CAPÍTULO 3. DESARROLLO

3.1 Breve descripción

El proceso de rastreo de objetos rígidos bidimensionales es llevado a cabo en tres pasos generales.

El primero consiste en la etapa de generación de la secuencia de imágenes que incluyen el objeto a rastrear. En ella, se tratan a las imágenes para que sean accesibles al resto del código y se pueda realizar el rastreo.

En la segunda etapa se realiza el procedimiento del cálculo de flujo óptico entre un par de imágenes, la cual incluye dos sub-etapas, ya que en primera instancia éste es calculado por medio del método de correlación, para un grupo de varios píxeles que se encuentran dentro de un radio definido por el usuario, y posteriormente se calcula nuevamente el flujo óptico dentro de ese grupo de píxeles para dar precisión del movimiento de cada uno de ellos. De esta manera, al término de esta etapa tenemos como resultado un par de matrices (*Vx, Vy*) que describen las velocidades de cada píxel en su dirección correspondiente (plano *'x'* o plano *'y'*). En *Vx* el signo positivo de un píxel indica que se mueve hacia su derecha, si este signo es negativo, se mueve a su izquierda. En *Vy* el signo positivo de un píxel indica que se mueve hacia abajo, si el signo es negativo, se mueve hacia arriba.

La tercera etapa consiste en el proceso de recolección de parámetros del campo de flujo óptico, es decir, la obtención de los desplazamientos lineales de cada píxel así

como su ángulo de rotación, según sea el caso. En esta fase está descrito el movimiento dado entre cada par de imágenes y en la secuencia completa, así podemos apreciar los cambios paso por paso y el resultado total de la suma de todos ellos.

3.2 Matlab

3.2.1 Matrices

Matlab es un programa desarrollado por Mathworks para el procesamiento de todo tipo de datos. Contiene diversas herramientas y funciones, útiles para resolver problemas de diversa índole. El nombre Matlab proviene de la contracción de los términos MATrix LABoratory ya que es un software complejo cuya unidad principal es la matriz que se utiliza para el análisis de sistemas de ecuaciones.

Las matrices son numéricas y rectangulares y poseen entradas posiblemente complejas; todas las variables representan matrices. A veces las matrices 1 x 1 se consideran escalares y las matrices con una sola fila o columna se consideran como vectores.

Existen diversas formas para introducir una matriz en Matlab, las principales son:

- Mediante la introducción de una lista explícita de elementos.
- Generándola mediante funciones y declaraciones.
- Creándola en un archivo de extensión **.**m
- Cargándola de un archivo de datos externo

Las entradas individuales de una matriz o un vector se pueden obtener poniendo los índices entre paréntesis y separados por comas.

Las operaciones con matrices se utilizan tanto en rectangulares como en escalares. En Matlab se utilizan las siguientes:

+ Adición
- Sustracción
* Multiplicación
^ Potenciación
´ Transpuesta
\ División izquierda
/ División derecha

Si los tamaños de las matrices son incompatibles para la operación, se obtiene entonces un mensaje de error (excepto cuando uno de los operandos sea escalar y el otro una matriz, donde se opera el escalar con cada término de la matriz).

3.2.2 Tipos de archivo

En Matlab se utilizan principalmente dos tipos de archivos:

a) Archivos de Escritura. Se consideran como secuencias de declaraciones de comandos, o sub-programas de Matlab. Las variables de este tipo de archivos son globales y modifican el valor de las variables que poseen el mismo nombre pero en el ambiente de la sesión actual de Matlab.

b) Archivos de Función. Se utilizan para extender Matlab. Con este tipo de archivos es posible crear funciones nuevas específicas a un problema y que puedan ser utilizadas en otras funciones de Matlab. Las variables que se utilizan en una función pueden ser locales o pueden extenderse globalmente. Es necesario que el archivo de función comience con una fórmula que al inicio contenga nombre, argumentos de entrada y argumentos de salida. Por ejemplo:

Function [variable(s) de salida]= nombre del archivo (variable(s) de entrada)

Para los dos tipos de archivos, el nombre de archivo corresponde al nombre del comando que se ingresa en la ventana de comandos de Matlab para llevar a cabo la rutina.

3.2.3 Formatos de imagen soportados

Para las pruebas de esta tesis se requiere trabajar con imágenes previamente diseñadas que se deben cargar y procesar por Matlab. En nuestro caso, usamos imágenes con formato .png, sin embargo, Matlab posee la capacidad de leer distintos formatos de imagen, lo que le confiere la característica de ser un programa con amplia accesibilidad. Entre los formatos que el software identifica se encuentran:

- PNG: (*Portable Network Graphics)* es un formato gráfico basado en un algoritmo de compresión sin pérdida para bitmaps que permite almacenar imágenes con una mayor profundidad de color y mayor calidad.
- BMP: (*Windows Bitmap)* es un formato de imagen sin compresión. En el archivo se almacena la totalidad de los píxeles que componen la imagen, por lo que su tamaño suele ser alto. Así mismo, permite diferentes profundidades de colores.
- PCX: (*Windows Paintbrush)* es un formato de archivos gráficos que comprime los datos del gráfico de tipo RLE.
- ICO: (*Windows Icon Resources)*
- TIFF: (*Tagged Image File Format)* es un formato de archivo de imágenes que contiene, además de los datos de la imagen propiamente dicha, "etiquetas" en las

que se archiva información sobre las características de la imagen, que sirve para su tratamiento posterior.

- HDF: (*Hierarchical Data Format)* nos permite compactar el tamaño de las imágenes además de manipular la información sin ningún problema independientemente de la plataforma empleada para escribir los datos.
- JPEG: (*Joint Photographic Experts Group)* es un algoritmo diseñado para comprimir imágenes con 24 bits de profundidad o en escala de grises, es también el formato de fichero que utiliza este algoritmo para comprimir imágenes. Sólo trata imágenes fijas.
- GIF: (*Graphics Interchange Format)* es un formato de imagen utilizado ampliamente en la World Wide Web, tanto para imágenes como para animaciones. No tiene pérdida de calidad.
- XWD: *(X Window Dump)* este formato de archivo se usa para grabar y restaurar imágenes de pantalla.
- CUR : *Windows Cursor Resources*

3.3 Descripción del algoritmo

3.3.1 Generación de imágenes

Con dos variantes, para probar la efectividad del sistema:

a) Por algoritmo matricial

1. Petición del número de imágenes de la secuencia.
2. Solicitud de dimensiones de la imagen en píxeles.
3. Solicitud de dimensiones del objeto (rectangular) en píxeles.

4. Posicionado del objeto dentro de la imagen.
5. Desarrollo de la imagen inicial.
6. Petición de parámetros de movimiento para cada imagen subsecuente.
7. Desarrollo de las imágenes subsecuentes.

b) Por lectura de archivo

1. Lectura de la imagen.
2. Conversión de la imagen RGB a imagen de intensidad.
3. Determinación de valores de variables.

3.3.2 Cálculo del flujo óptico

Procesamiento de variables

1. Chequeo y ajuste de variables de entrada.
2. Verificación de correspondencia entre variables (mensajes de error en caso correspondiente).
3. Suavizado de la imagen.
4. Cálculo de constantes necesarias.
5. Pre-ajuste para penalización en distancias mayores.
6. Ampliado de los límites de la imagen. (Para movimientos que se desarrollan cerca o sobre éstos).
7. Cálculo del gradiente.

Flujo óptico

1. Algoritmo de correlación
    a) Definición de regiones de píxeles
    b) Cálculo de la diferencia de desplazamiento entre tramas.

2. Algoritmo de Lucas Kanade
    a) Definición de los límites de la región de búsqueda
    b) Cálculo del gradiente para el píxel determinado.
    c) Identificación del píxel correspondiente
    d) Cálculo de la distancia entre puntos para el eje correspondiente.
    e) Graficado del campo de flujo óptico

3.3.3 Obtención de parámetros

1. Identificación del tipo de movimiento (traslación o rotación)
    a) Traslación
        1. Cálculo del movimiento en el eje *'x'*.
        2. Cálculo del movimiento en el eje *'y'*.
    b) Rotación
        1. Definición de límites y centro del campo de flujo.
        2. Determinación de la dirección con mayor contenido de información. (Los campos de flujo no son simétricos, por ello se elige una dirección derecha o izquierda a partir del centro para el cálculo del ángulo buscado).
        3. Cálculo del ángulo promedio.
2. Resultados de desplazamientos en los ejes o ángulo de rotación entre el par de imágenes.
3. Resultados de desplazamientos y rotación totales.

3.4 Descripción del código

3.4.1 Descripción por niveles

Para comprobar el desempeño del programa, el usuario debe ingresar en la ventana de comandos de Matlab el comando *'ex#'* donde # corresponde al número del ejemplo que se desea probar (*ex1.m* realiza un ejemplo usando imágenes creadas con *generasec.m* y *ex2.m* lee imágenes .png previamente diseñadas por medio de *lectura.m*). De esta manera, se ejecuta el programa *ex#.m*, que a su vez lleva a cabo varias tareas y algunas funciones, como se observa en la figura 3.1.

El procedimiento descrito por este diagrama revela el orden en que se ejecutan las funciones necesarias para el rastreo de movimiento de objetos. Es importante mencionar que este proceso se lleva a cabo 'n-1' veces, siendo 'n' el número total de imágenes de la secuencia, pues se calcula el flujo entre cada par de ellas.

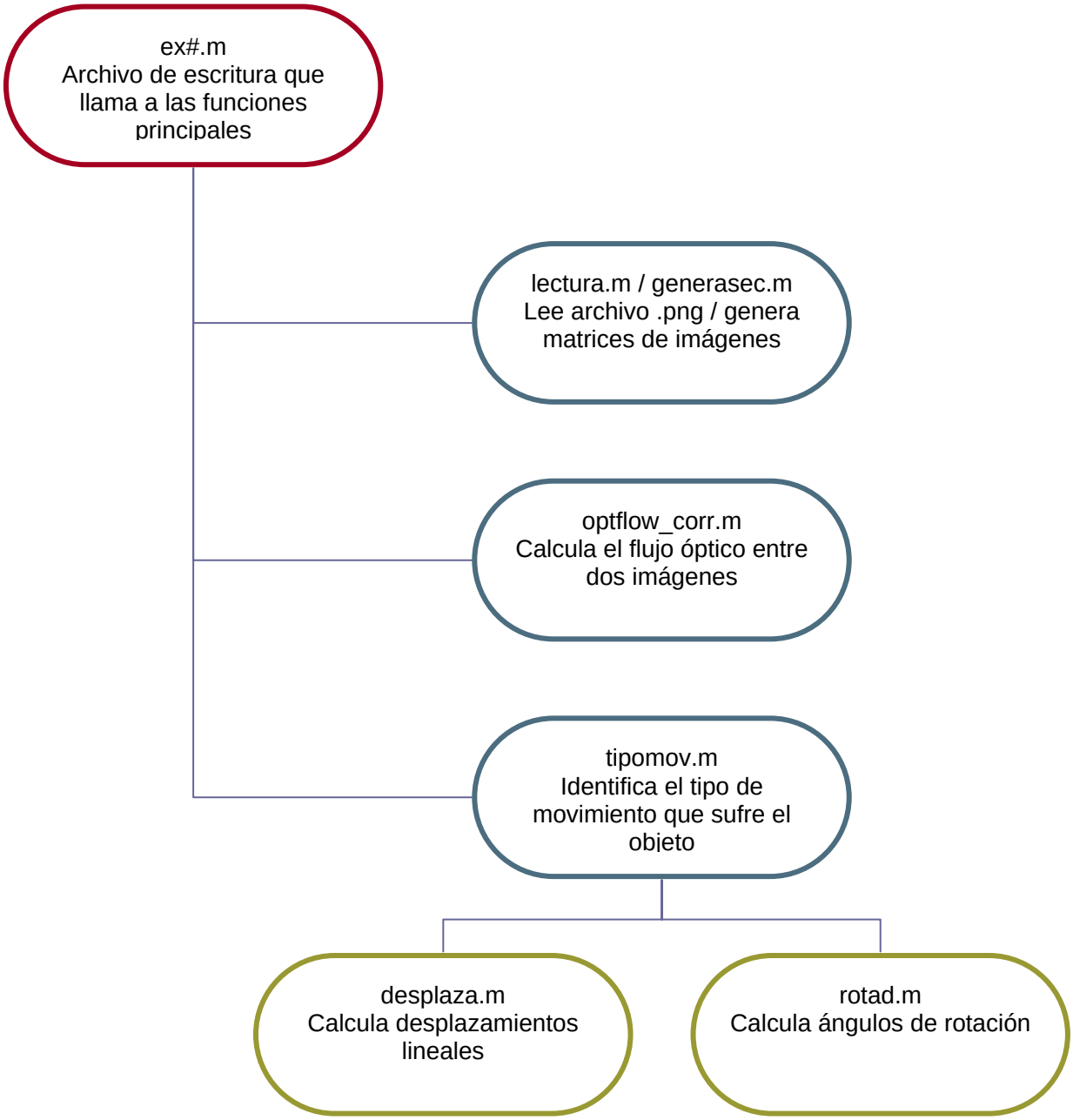


Figura 3.1 Niveles del programa de rastreo de movimiento

3.4.1 Funcionamiento específico

Para dar una comprensión detallada de cada uno de los pasos que el algoritmo propuesto sigue, es necesario conocer y explicar el código implementado. En esta sección las líneas de código se identifican con la tipografía en itálicas. Como en muchos lenguajes de programación, los mensajes posteriores a un signo % actúan como comentarios anexados al código.

La ejecución del programa comienza cuando el usuario ingresa *'ex#'* en la ventana de comandos de Matlab, llamando al programa correspondiente *'ex#.m'*. Así, el primer grupo de instrucciones y el que manda a llamar al resto de las otras funciones da inicio al proceso de rastreo de objetos.

clear all

[num,sec] = generasec();

ó

[I1, I2, …] = lectura() ;

En estas líneas de código se borran los contenidos de todas las variables mediante el uso del comando *clear all*, y posteriormente se manda a llamar a la función encargada de la lectura de las imágenes. Dependiendo del ejemplo que se ejecute, el tipo de imagen con el que se va a trabajar es distinto, por lo que su lectura involucra funciones distintas, las cuales son explicadas a detalle más adelante.

Las variables que recibimos de *generasec* son *num,* que determina el número de imágenes de la secuencia que desea el usuario y *sec,* una matriz tridimensional que

dentro de ella posee el conjunto de imágenes bidimensionales de la secuencia. Para la función *lectura()*, (*I1, I2,…*), son las imágenes correspondientes a una misma secuencia, descritas como matrices de intensidad de grises y con las cuales se trabajará en todo el programa.

fprintf ('\n');

[Vx,Vy,reliab] = optflow_corr(I1, I2, patch_r, search_r, sigma, thr, show);

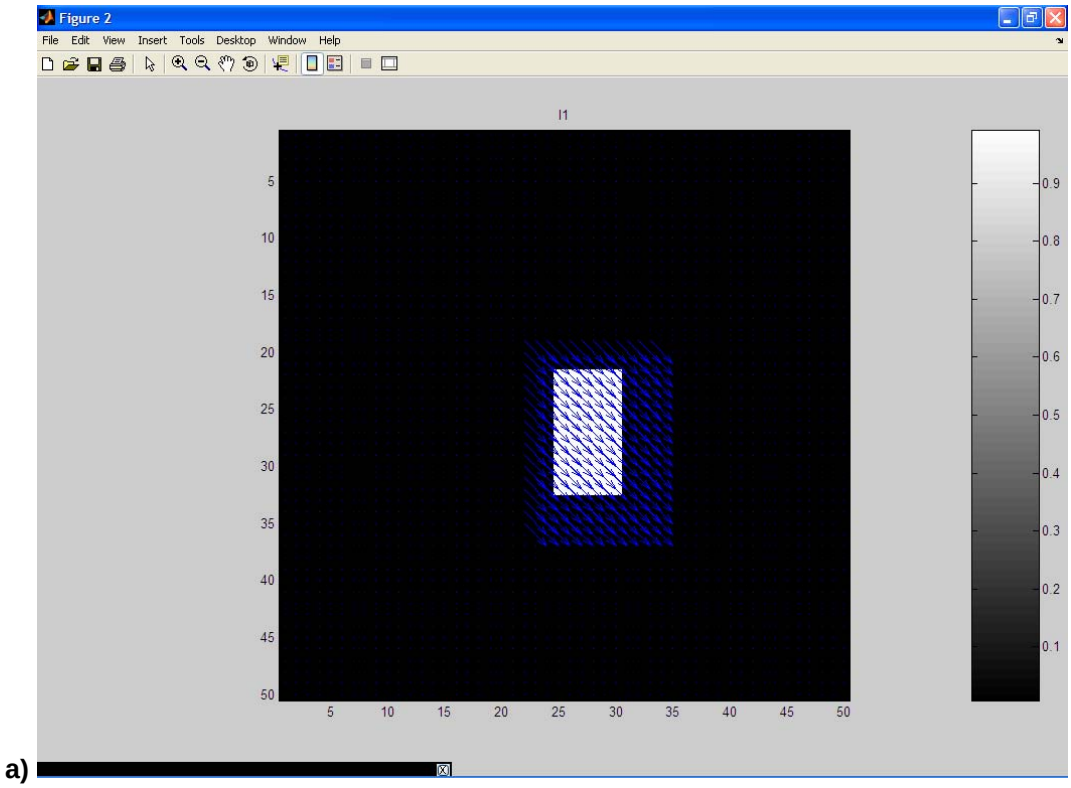
fprintf ('\n'); salta a la siguiente línea en la ventana de comando de Matlab, esto simplemente es parte del formato de presentación de los resultados. *[Vx,Vy,reliab] = optflow_corr(I1, I2, patch_r, search_r, sigma, thr, show);* es el llamado a la función correspondiente al archivo *optflow_corr.m*, que como se mencionó en el capítulo de introducción, forma parte de la librería para procesamiento de imágenes y video desarrollada por Piotr Dollar [18].

Esta función lleva a cabo el cálculo del campo de flujo óptico entre dos imágenes, respaldado por el marco teórico del capítulo 2. Este algoritmo de flujo óptico basa su funcionamiento en la técnica de correlación en primera instancia, para calcular el movimiento de un grupo de píxeles dentro de la imagen, y posteriormente aplica la técnica propuesta por Lucas Kanade para obtención del movimiento de cada píxel dentro del grupo. Estos resultados son almacenados para nuestra posterior apreciación y manejo. Además, entrega como parte de sus resultados una imagen en la que se puede apreciar el campo de flujo óptico sobre el objeto en movimiento (ver figura 3.2a y 3.2b). Las variables que recibimos de ella son *Vx, Vy* y *reliab*. *Vx* es una matriz cuyo tamaño es igual a las dimensiones en píxeles de la imagen, y describe la magnitud de la velocidad y la

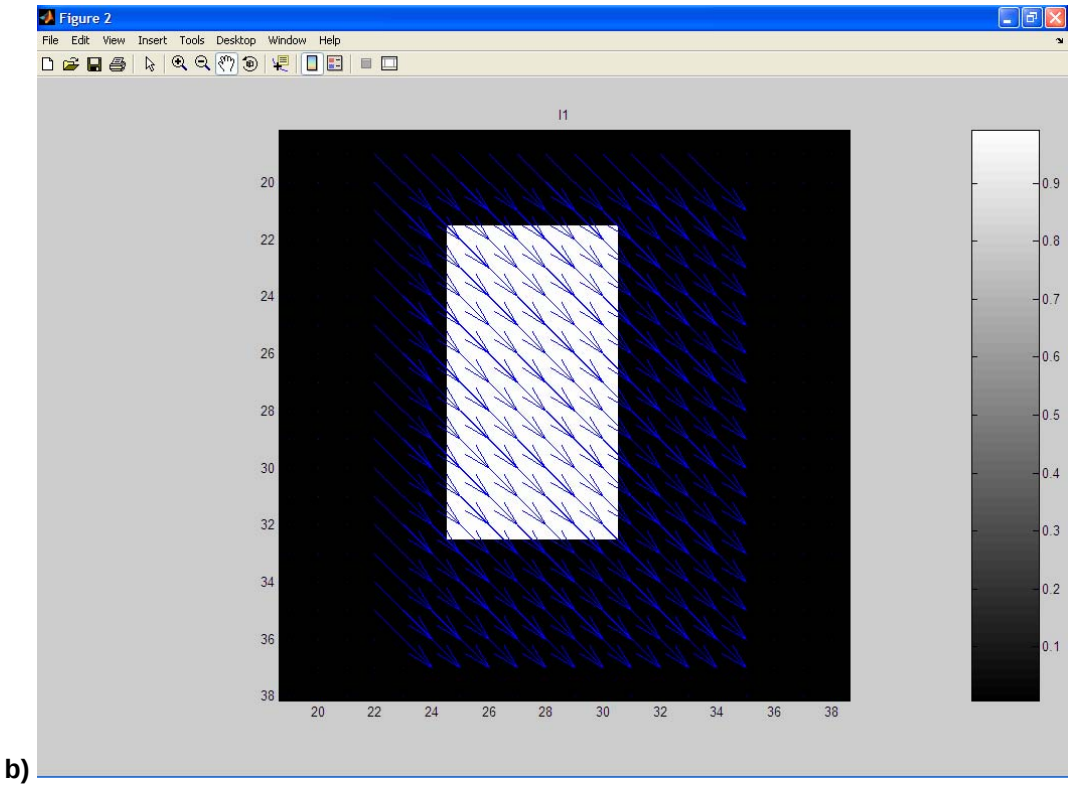
dirección de movimiento de éstos entre dos imágenes en el eje *'x'* (si el valor observado es positivo, el píxel se mueve a la derecha; si es negativo se mueve hacia la izquierda), mientras que *Vy*, que tiene exactamente las mismas dimensiones, describe éstos valores para el eje *'y'* (si el valor observado es positivo, el píxel se mueve hacia abajo; si es negativo, se mueve hacia arriba). *reliab* es el valor de la confiabilidad del resultado que se tiene para cada píxel, por lo que sus dimensiones también son iguales a las de *Vx* y *Vy*.

Los valores de variables que nosotros proporcionamos a la función son los siguientes:

- *I1* e *I2*, que son las matrices de intensidad de grises de la imagen inicial (*I1*) y la subsiguiente (*I2*), entre las que se calcularán los parámetros de desplazamiento o rotación.
- *patch_r*, que determina el tamaño en píxeles de la región para búsqueda por correlación.
- *search_r*, define el radio de búsqueda para la región correspondiente.
- *sigma* (valor opcional), establece el suavizado que se hará de la imagen, para mejorar los resultados. Este valor puede ser cero, si no se requiere ningún suavizado.
- *thr* (valor opcional), es el umbral de la confiabilidad relativa. Este valor sirve como un filtro de confianza que determina cuales valores del campo de flujo óptico son lo suficientemente correctos ser ingresados a las matrices *Vx* y *Vy*. Por default su valor es .01.
- *show* (valor opcional), número asignado a la figura por desplegar. Si su valor es cero, ésta no es mostrada.



a)



b)

Figura 3.2 a) Ventana desplegada en pantalla con el campo de flujo óptico en sus dimensiones originales, **b)** Acercamiento a la zona del campo de flujo óptico.

[promx, promy, ang] = tipomov(Vx, Vy);

Aquí se hace el llamado a la función *tipomov*, la cual se encarga de distinguir si el movimiento que estamos detectando es lineal, o si se trata de una rotación. Las variables de salida que apreciamos son *promy, promx, ang*, que corresponden a los parámetros de movimiento a los que llega como resultado el sistema, y en los cuales ahondaremos posteriormente. Los valores que tiene como entradas son *Vx* y *Vy*, pues son la única fuente de información que le permite saber el tipo de movimiento que está ocurriendo.

total1=total1+promx;

total2=total2+promy;

angt=angt+ang;

Estas líneas de código simplemente se encargan de hacer el conteo total en los desplazamientos en *'x'* y *'y'*, y el ángulo de rotación.

El proceso demostrado hasta el momento, debe repetirse a partir del llamado de la función *optflow_corr*, tantas veces como se desee, pues dependiendo del número de imágenes que posea la secuencia y de las que se requiera calcular sus transformaciones en el plano *'x,y'*, se debe realizar este proceso nuevamente para cada par de frames. En el caso de usar la lectura de imágenes por medio de la función *generasec()* se utiliza un ciclo for que repite todos los pasos anteriores 'num-1' veces (es decir el número total de transiciones).

fprintf ('Movimiento total: \n desplazamiento en X: %i, en Y: %i \n', total1, total2);

fprintf ('Rotacion: %i grados \n', ang);

En este par de instrucciones finales, se muestran en la ventana de comandos de Matlab los resultados finales del movimiento registrado, a manera de suma de todos los desplazamientos y rotaciones que sufrió el objeto a lo largo de la secuencia computada, bajo las categorías de desplazamiento total en eje *'x'*, en el eje *'y'*, y ángulo de rotación total.

Esto se hace con el fin de demostrar la capacidad del algoritmo en el contexto de rastreo de movimiento para el cálculo de los parámetros de movimiento del objeto, pues en una aplicación real, puede no interesarnos la manera en que se desarrollaron los cambios de posición del objeto a lo largo de toda una secuencia, pero sí resulta relevante conocer la posición final de éste para poder interactuar con él de la manera que se desee.

function [I1, I2, …] = lectura()

I1=imread('TT1.png');

I2=imread('TT2.png');

…

Al dar inicio a la función *lectura*, se leen las imágenes (archivos .png) que se encuentran en la misma carpeta donde está el código, ya que de no estar ahí, es necesario hacer la aclaración al hacer su llamado y proporcionar la ruta donde éstas se localizan. La cantidad de veces que se lleva a cabo este procedimiento es igual al número de archivos que se deseen leer como parte de la secuencia, tomando en cuenta que para

hacer el cálculo del flujo óptico entre ellos se requieren, al menos, dos imágenes pertenecientes a la misma serie.

La figura 3.3 muestra el contenido del archivo TT1.png. Los tipos de archivos que Matlab es capaz de reconocer son especificados en la sección 3.2 de este capítulo.

I1=rgb2gray(I1);

figure

imshow(I1);

Para llevar a cabo el rastreo de movimiento bidimensional, se requieren matrices definidas en dos dimensiones, por ello, las imágenes leídas en RGB (descritas en tres dimensiones), deben ser transformadas a matrices de intensidad de gris (descritas en dos dimensiones). El comando *rgb2gray* es uno de los más comunes para llevar a cabo esta tarea. Todas las matrices con las que se trabajen deben pasar por este proceso, ya que de no hacerlo, el número de sus dimensiones no permitirá que el código se ejecute y *optflow_corr* desplegará un mensaje de error indicando el incumplimiento de esta condición.

imshow(I1); se encarga de desplegar en una nueva ventana el contenido de la imagen *I1*, es decir, la apreciación visual de la matriz de intensidades de gris correspondiente, finalizando esta función. Este paso no es necesario para la correcta ejecución del código, sin embargo, permite al usuario conocer las imágenes con las que se está trabajando en caso de no haberlas visto con anterioridad. En la figura 3.4 podemos observar la ventana que este comando despliega.

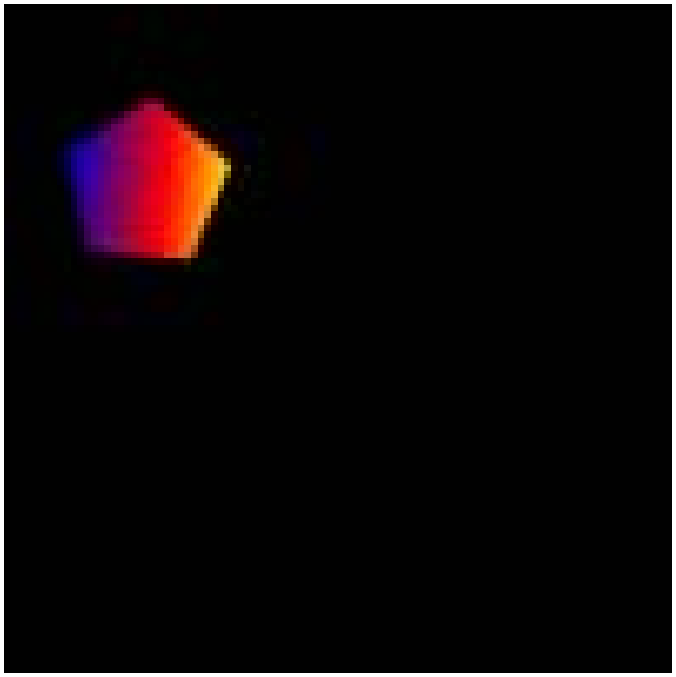


Figura 3.3 TT1.png

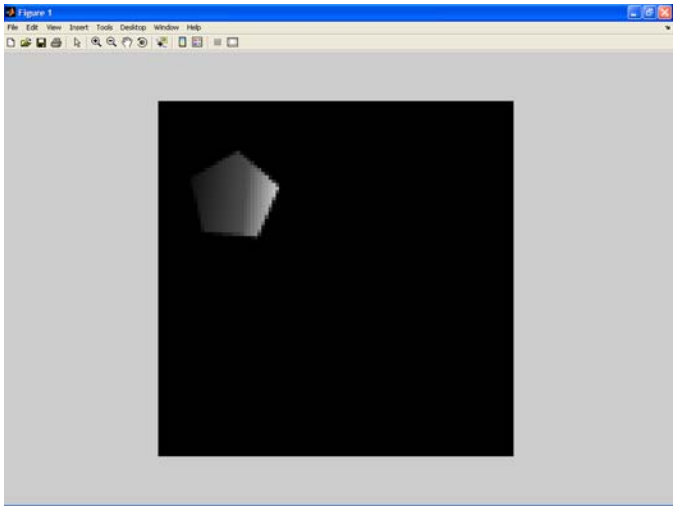


Figura 3.4 Imagen de intensidad correspondiente a TT1 mostrada por el comando *imshow*

En caso de ocupar la otra función para la lectura de la secuencia de imágenes (*generasec)* el siguiente conjunto de instrucciones es llamado a ejecución:

```
function [num, sec] = generasec()
clear all;
num=0;
lx=0; ox=0; px=0;
ly=0; oy=0; py=0;
dxx=1000; dyy=1000; ra=0;
tip='a';
```

Las variables de salida de esta función es el número de imágenes de la secuencia (*num*) y una matriz tridimensional (*sec*) que contiene dentro de ella, el conjunto de imágenes bidimensionales que conforman la escena.

El resto de las instrucciones sólo dan inicialización a las variables que se ocupan dentro de este código. Es importante mencionar que todas éstas son variables 'dobles', excepto *tip* que es una variable de tipo caracter.

```
while ((num<2)||(num>10))
   num = input('\nNúmero de imágenes de la secuencia (2-10): ');
end
```

En este conjunto de instrucciones se hace la petición al usuario del número de imágenes que desea que formen parte de la secuencia. Como podemos apreciar se usa un ciclo *while*, el cual funciona a manera de filtro, para que la respuesta que obtengamos

del usuario cumpla los límites de 2 a 10 imágenes. Esto se debe a que como mínimo se necesitan dos imágenes para poder hablar de una secuencia, y para no hacer muy lento y tedioso el análisis del movimiento se toma un límite máximo de 10.

```
%dimensiones imagen
fprintf('\nProporciona las características de la imagen inicial \n');
while ((lx<50)||(lx>150)||(ly<50)||(ly>150))
   fprintf('\nDimensiones de la imagen en píxeles: (50:150),(50:150)')
   ly = input('\n  para x: ');
   lx = input('  para y: ');
end
```

En este momento comienza la generación de la imagen inicial, la cual dará la mayoría de las características necesarias para las imágenes posteriores. El conjunto de instrucciones superior, se encarga de definir el tamaño de la imagen. De la misma forma que con el ciclo *while* anterior, éste es usado como un filtro para que la imagen a crear sea mayor a los 50 píxeles y menor a los 150. Como se puede notar, es una imagen pequeña en realidad, esto es hecho para reducir los tiempos de procesamiento de la imagen y así poder comprobar el funcionamiento de la función de rastreo en un menor tiempo. Así, el usuario procede a delimitar la imagen con valores para el eje *'x'* y el eje *'y'*, cuyos valores son guardados en *ly* y *lx*, respectivamente. Es importante mencionar que estas dimensiones se mantendrán para todas las imágenes de la secuencia.

```
%dimensiones objeto
while ((ox<5)||(ox>15)||(oy<5)||(oy>15))
```

```
   fprintf('Dimensiones del objeto rectangular en píxeles: (5:15),(5:15)')
   oy = input('\n  para x: ');
   ox = input('  para y: ');
end
obj = rand(ox,oy);
[lxx,lyy] = size (obj);
dx = lx-ox+1; dy = ly-oy+1;
xx = round(ox/2); yy = round(oy/2);
```

Ahora, se procede a crear el objeto rígido que se mantendrá en movimiento durante la secuencia. Para ello, se piden sus dimensiones para el eje *'x'* y el eje *'y'* en píxeles. Nuevamente usamos un filtro para mantener el tamaño del objeto entre 5 y 15 píxeles de longitud para cada eje. De esta forma podemos saber que se trata de un objeto rectangular para el cual definimos su base y su altura. Los valores que proporciona el usuario son guardados en las variables *oy* y *ox*, respectivamente. Después de definir las dimensiones, se crea una matriz *obj* de valores aleatorios que representan píxeles de varias tonalidades en escala de grises y que mantiene las dimensiones dadas para el objeto.

Posteriormente, con *[lxx,lyy] = size (obj);* se toman las medidas de esta matriz que posteriormente serán necesarias; *dx = lx-ox+1; dy = ly-oy+1;* son límites del objeto dentro de la imagen completa, es decir, es la distancia a mínima (respecto a cada eje) en la que puede estar posicionado el objeto dentro de la imagen para que no salga de los límites de ella, con la finalidad de que con el siguiente conjunto de instrucciones el usuario conozca estos límites.

xx = round(ox/2); yy = round(oy/2); son valores que se determinan para posteriormente colocar el objeto en el punto designado por el usuario.

```
%localización objeto
while ((px<xx)||(px>dx)||(py<yy)||(py>dy))
   fprintf('Posición inicial del objeto (centro) en coordenadas \n');
   fprintf('  para x (%d:%d)',yy, dy);
   py = input(' ');
   fprintf('  para y (%d:%d)', xx,dx);
   px = input(' ');
end
```

Como se había dicho, al ejecutarse estas instrucciones, el usuario puede ahora definir una posición inicial para el objeto en cuestión sin el riesgo de colocarlo fuera de la imagen y con ello causar un error de programa. Así, el ciclo *while* obliga al usuario a respetar los límites de posicionamiento del objeto para que éste proporcione valores adecuados que son guardados en *py* y *px*.

```
%crear imagen inicial
sec = zeros(lx,ly,num);
for x=1:ox;
   for y=1:oy;
      sec(px+x-xx, py+y-yy, 1) = obj(x,y);
   end
end
```

figure;

imshow(sec(:,:,1));

Tras recibir todas las especificaciones anteriores, procedemos a crear la imagen que será usada como la primera parte de la secuencia.

sec = zeros(lx,ly,num); crea una matriz tridimensional de 'ceros(0)' (que se aprecian como píxeles negros gráficamente) que tiene por dimensiones *lx, ly y num. lx* y *ly* refieren al tamaño de la imagen, y *num* es el total de imágenes de la secuencia. Se podría decir que *sec* es una matriz que se encarga de almacenar '*num*' matrices bidimensionales que corresponden a imágenes en escala de grises.

for x=1:ox; for y=1:oy; son un par de ciclos usados para realizar el proceso de llenado de la matriz de 'ceros' *sec,* con el contenido de *obj*, por medio de un recorrido en sus dimensiones por *'x'* y *'y'*. *sec(px+x-xx, py+y-yy, 1) = obj(x,y);* es precisamente la instrucción que se encarga de colocar el contenido de *obj* en la posición que le corresponde de acuerdo a los valores de posición inicial, tamaño de la imagen y tamaño del objeto que definió el usuario. De esta forma, la imagen inicial (*num=1*) es creada y a partir de ella se harán las modificaciones en la posición y rotación del objeto para generar su movimiento a lo largo de la secuencia.

figure; imshow(sec(:,:,1)); se encargan de mostrar en una nueva ventana la imagen inicial para que de esta forma el usuario pueda percibir el resultado de todo lo que ha ejecutado y pueda saber hacia donde puede mover al objeto.

```
for (k=2:num)
   fprintf('\nImagen subsecuente número %d', k);
   while ((tip~='D')&&(tip~='R')&&(tip~='d')&&(tip~='r'))
      tip = input('\n  Tipo de movimiento: \n  Desplazamiento (D), Rotación (R) ','s');
   end
```

Aquí se inicia un ciclo que se encargará de generar las imágenes subsecuentes de la escena. El programa se encarga de informar al usuario qué imagen subsecuente está desarrollando y pregunta por el tipo de movimiento que se desea realizar, teniendo como opciones desplazamiento (lineal) o rotación, identificados cada uno por la letra ‘D’ y ‘R’. Como se observa se ha colocado otro ciclo *while* para evitar errores en el programa, en el caso de que el caracter dado por el usuario sea distinto a ‘D’ o a ‘R’.

```
if ((tip=='D')||(tip=='d'))
      while ((px+dxx+ox>lx+1)||(py+dyy+oy>ly+1)||(px+dxx<xx)||(py+dyy<yy))
         dyy = input('\n  Desplazamiento en x: ');
         dxx = input('  Desplazamiento en y: ');
         pxx = px+dxx; pyy = py+dyy;
         if ((pxx+ox>lx+1)||(pyy+oy>ly+1)||(pxx<xx)||(pyy<yy))
            fprintf('ERROR: Objeto fuera de las dimensiones de la imagen \n');
            dxx=1000; dyy=1000;
         end
      end
```

Si el usuario decide desplazar al objeto linealmente *((tip=='D')||(tip=='d'))*, entonces se entra a este conjunto de instrucciones, para solicitar los valores del movimiento del objeto para el eje *'x'* y para el eje *'y'*, los cuales son almacenados en *dyy* y *dxx* respectivamente. Para evitar que el objeto se desplace fuera de la imagen, son colocados varios filtros que se encargan de hacer el cálculo de los límites de la posición final del objeto, y en los que se verifica que éstos no excedan a los del tamaño de la imagen. En el caso de que así suceda, entonces el programa se encargará de mostrar el mensaje de error que especifica que el objeto está fuera de las dimensiones de la imagen, y se vuelve a solicitar que se dé un desplazamiento válido.

```
px = px+dxx; py = py+dyy;
xx = round(lxx/2);
yy = round(lyy/2);
for x=1:lxx;
   for y=1:lyy;
      sec(px+x-xx, py+y-yy, k) = obj(x,y);
   end
end
figure;
imshow(sec(:,:,k));
```

Una vez comprobado que el desplazamiento es válido, entonces se procede a realizar una acción similar a la hecha en la fase de creación de la imagen inicial. Primeramente se calcula la nueva posición del objeto (*px = px+dxx; py = py+dyy;*), y se calcula un par de variables *xx* y *yy* que se usan para ordenar la forma en que se vacían

los datos que contiene la matriz *obj*. Entonces, se ejecutan un par de ciclos *for* que hacen el barrido de los ejes *'x'* y *'y'* de la matriz *obj* y con la instrucción *sec(px+x-xx, py+y-yy, k) = obj(x,y);* se copia sobre la matriz *sec* en la posición indicada, los valores que se encuentren en las coordenadas respectivas de *obj*. Es importante resaltar el hecho de que el copiado de datos sobre *sec* se hace en una 'nueva' imagen, distinta a la inicial, definida por el valor de *k*, es decir, las siguientes imágenes bidimensionales de la escena se guardan en nuevas posiciones conforme k va creciendo, y de esta forma, el resultado que tenemos tras finalizar el proceso de desarrollo de la secuencia, es un conjunto de imágenes almacenadas en *sec*.

Una vez más por medio de los comandos *figure; imshow(sec(:,:,k));* se muestra en pantalla de manera gráfica el contenido de la matriz *sec* en la posición dada, tras haber sido desplazada.

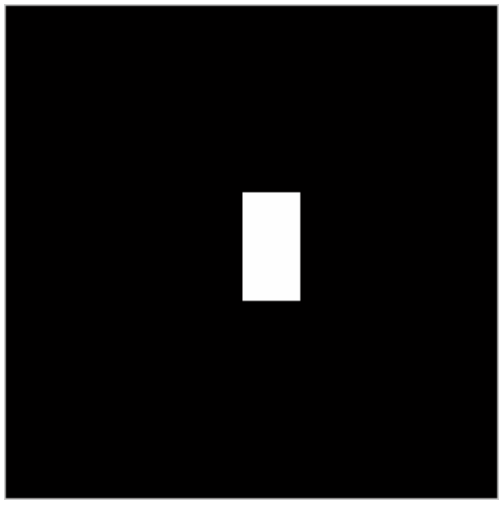
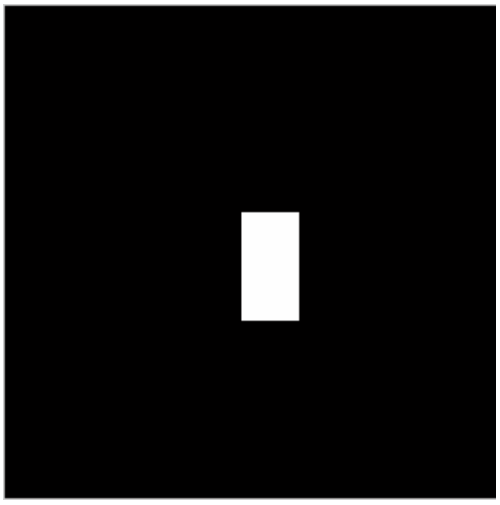
a)  **b)** 

Figura 3.5 a) Imagen inicial sec(:,:,1), **b)** Imagen subsecuente sec(:,:,2).

```
else
    ra = input('\n  Ángulo de rotación deseado: ');
    obj = imrotate(obj,-ra,'bil');
    [lxx,lyy] = size(obj);
    xx = round(lxx/2);
    yy = round(lyy/2);
    for x=1:lxx;
       for y=1:lyy;
          sec(px+x-xx, py+y-yy, k) = obj(x,y);
       end
    end
    figure;
    imshow(sec(:,:,k));
 end
```

En caso de que se desee realizar una rotación del objeto, entonces la anterior serie de instrucciones es llevada a cabo, la cual es muy similar a la de desplazamiento, con la diferencia de que la petición ahora es para recibir un ángulo de rotación para el objeto.

La rotación en sí, es hecha por medio de la instrucción *obj = imrotate(obj, -ra,'bil');* la cual se encarga de rotar la matriz *obj* con un ángulo *'-ra'* (el signo negativo es usado sólo para que la rotación sea en sentido de las manecillas del reloj) y con la especificación *'bil'* que refiere al suavizado de la imagen, para que se pierda la menor cantidad posible de información del objeto tras modificarlo. De esta forma, tras rotarlo, el objeto es copiado

sobre la matriz *sec* en la posición subsecuente de la misma forma que cuando se realiza un desplazamiento lineal, y tras hacer esto, ésta imagen es mostrada en pantalla para que el usuario pueda observar las transformaciones de la escena tras llevar a cabo este paso.

```
   tip='a';
   dxx=1000; dyy=1000;
   ra=0;
end
```

Finalmente, *generasec()* asigna valores a ciertas variables para evitar errores cuando se vuelve a ejecutar, y hecho esto, manda los contenidos de *num* y *sec* para que se pueda proseguir con el resto del programa.

Posteriormente, se manda a llamar a la función *optflow_corr*, como se explicó anteriormente, y se obtienen las matrices *Vx* y *Vy* que definen la velocidad de movimiento en el eje correspondiente. Así, se procede a analizarlas para detectar el tipo de movimiento que se está presentando.

```
function [promx, promy, ang] = tipomov(Vx, Vy)
Vxx = round(Vx);
Vyy = round(Vy);
Vxx = Vx - Vxx;
Vyy = Vy - Vyy;
```

La función tipomov se encarga de hacer esta detección de la siguiente manera: *Vxx = round(Vx);, Vyy = round(Vy);* crean un par de matrices con el mismo contenido que

Vx y *Vy*, pero realizando el redondeo de todos sus elementos, mientras que *Vxx = Vx - Vxx;, Vyy = Vy - Vyy;* calcula la diferencia entre la matriz original y la matriz correspondiente a sus valores redondeados.

Cuando se detecta un movimiento lineal, los valores de las matrices *Vx* y *Vy*, son enteros, y si el movimiento es rotacional presentan fracciones, por lo que al hacer el redondeo de sus elementos y calcular la diferencia entre ellos, somos capaces de distinguir entre movimientos lineales y movimientos rotacionales.

```
if ((Vxx<=0.01)&(Vyy<=0.01))
   [promx, promy] = desplaza( Vx, Vy );
   ang=0;
else
   [ang] = rotad( Vx, Vy );
   promx=0; promy=0;
end
```

Este conjunto de instrucciones es el encargado de hacer la diferenciación mencionada. *if ((Vxx<=0.01)&(Vyy<=0.01))* es la condición en la cual se analiza el contenido de *Vxx* y *Vyy*.

Si el movimiento fue lineal, los valores de *Vx* y *Vy* son enteros o muy cercanos a éste, que al ser redondeados dan como resultado una matriz con exactamente los mismos números, y al calcular la diferencia entre ellos el resultado es cero o un valor próximo. De esta forma sabemos que el tipo de movimiento que se está presentando se desarrolla sin

rotación alguna, y procedemos a calcular los valores de los desplazamientos en *'x'* y *'y'* llamando a la función *desplaza*, la cual nos dará estos resultados dentro de las variables *promx* y *promy*. Como sabemos que, para este caso, no existe rotación alguna entonces ajustamos el valor del ángulo a cero usando *ang=0;*.

En caso de que el movimiento sea rotacional, al hacer la resta entre *Vx*, *Vy* y sus matrices redondeadas correspondientes, obtendremos un resultado diferente de cero, por lo que se procederá a realizar el cálculo del ángulo de rotación llamando a la función *rotad*, que guardará este valor dentro de la variable *ang*. Los valores de *promx* y *promy* serán cero, al no existir movimiento lineal.

```
function [promx,promy] = desplaza( Vx, Vy )
promx=0; promy=0;
```

Declara la función *desplaza*, que toma los valores de *Vx* y *Vy*, y comienza igualando a cero los valores de *promx* y *promy* que posteriormente serán calculados.

```
[i,j,indx]=find(Vx);
lx=length(indx);
indx=sum(indx);
```

find es una función de Matlab, que encuentra los índices de un vector, o de una matriz como es este caso. Esto refiere a que crea dos vectores, *i* y *j ,* donde *i* determina el renglón y *j* la columna correspondiente de la matriz en cuestión, en donde se encuentran valores diferentes a cero, además de crear un nuevo vector *indx* que guarda todos los

valores de la matriz distintos a cero. La finalidad de esta instrucción dentro del algoritmo, es conservar únicamente los valores que presentan información de movimiento, pues los valores de *Vx* que son cero, expresan estática en el campo de flujo óptico.

length permite medir el tamaño de un vector, en este caso de *indx* y guarda el resultado en *lx*.

sum hace la sumatoria de todos los valores del vector especificado (*indx)*, y para este caso, guarda el resultado de esta operación nuevamente dentro de *indx*, reduciendo su tamaño de un vector de longitud *lx*, a una variable de una sola entrada.

```
if (indx==0)
   fprintf ('Desplazamiento detectado en X: 0 ');
else
   promx=indx/lx;
   fprintf ('Desplazamiento detectado en X: %i ',promx);
end
```

La condición *if* presentada, revisa el contenido de la variable *indx*, para saber si se presentó algún movimiento en el eje *'x'*. Si el resultado de la sumatoria de los elementos de la matriz *Vx* es igual a cero, quiere decir que no hubo desplazamientos en este eje, y por consiguiente despliega esta deducción en la ventana de comandos de Matlab. Sin embargo, si el valor de *indx* es distinto a cero, entonces estaremos apreciando desplazamientos dentro de este eje, por lo que se prosigue a la etapa *else* de la condición y se calcula el promedio del desplazamiento apreciado. Esto se logra dividiendo la *indx*

que contiene la suma de todos los desplazamientos en *'x'*, y dividiendo entre el total de elementos que presentan este desplazamiento. Ahora, el resultado mostrado en la ventana de comandos de Matlab será el contenido de la variable *promx*.

```
[k,l,indy]=find(Vy);
ly=length(indy);
indy=sum(indy);
if (indy==0)
   fprintf ('Desplazamiento detectado en Y: 0 \n');
else
   promy=indy/ly;
   fprintf ('Desplazamiento detectado en Y: %i \n',promy);
end
```

De la misma forma en que se siguieron los pasos para el cálculo de desplazamientos en el eje *'x'* se calculan los del eje *'y'*, y con ello queda finalizada la función *desplaza*.

```
function [ang] = rotad( Vx, Vy )
ang=0; ds=0;
sum1=0; sum2=0; ci=0; cd=0;
```

ang es la función que se ejecuta en caso de que *tipomov* detecte un movimiento rotacional, y el resultado que obtenemos de ella es el ángulo de rotación que presenta el objeto entre dos imágenes. Las variables que se utilizan para su cálculo son inicializadas con valor de cero (*ang, ds, sum1, sum2, ci, cd*).

```
[i,j]=find(Vx);
d1=min(i);
d2=max(i);
d3=min(j);
d4=max(j);
```

El primer paso en el cálculo del ángulo de rotación es identificar los límites del campo de flujo óptico para posteriormente determinar las coordenadas que identifican su centro. Por ello se usa *find* nuevamente y con las funciones *min* y *max* se identifican los valores máximos y mínimos para los renglones y columnas de la matriz *Vx*. Se debe mencionar que este proceso sólo se lleva a cabo para *Vx* ya que por las propiedades del cálculo de flujo óptico en los casos de rotación, los límites en la matriz *Vy* son exactamente iguales y por ello la repetición de éste mismo resulta innecesaria.

```
dy=round(d1+((d2-d1)/2));
dx=round(d3+((d4-d3)/2));
```

Una vez obtenidos los límites del campo de flujo óptico, procedemos a calcular las coordenadas del centro, lo que resulta sencillo pues éstos describen a una pequeña matriz rectangular con dimensiones *d2-d1, d4-d3*. Entonces en *dx* y *dy* se guardan los puntos intermedios entre estos límites, para las coordenadas de renglones y columnas, y así se determina un punto como el centro de la matriz. El redondeo que se hace de estos valores es necesario para los casos en que no existe un punto intermedio exacto entre los límites calculados, y por ello se considera a la siguiente coordenada como la adecuada. Por las características que presenta el campo de flujo óptico, este ajuste no altera de

manera considerable el valor del ángulo de rotación y por ello no es necesario aumentar la complejidad del código.

El cálculo del centro se lleva a cabo con el fin de establecer un punto de referencia en la coordenada horizontal y vertical del plano. El ángulo que se presenta como resultado de la rotación del objeto, es calculado por medio de la pendiente de la recta que toma las coordenadas del centro para establecer en ese punto la intersección de los ejes. De esta forma, se mide la pendiente de la recta que va de este punto hacia otro punto definido como la posición final de un píxel localizado en la misma coordenada vertical en el campo de flujo óptico (valor de *dy* constante) luego de su rotación.

ds=round((d4-d3)/10);

ds es una variable que se obtiene para evitar calcular ángulos muy cercanos al centro, ya que éstos presentan ángulos muy elevados que no aportan información confiable. Por ello, se define como la distancia horizontal mínima a partir de la cual se toman en cuenta los píxeles en movimiento para iniciar el proceso de cálculo de la pendiente.

El campo de flujo óptico no posee características de simetría en la mayoría de los casos. En algunas ocasiones éste presenta mayor precisión en algunas regiones, que aportan información sustancial y confiable sobre el movimiento que se está presentando, por ello, es necesario determinar hacia qué dirección, derecha o izquierda, se han de calcular las pendientes de recta para determinar el ángulo.

```
for y=d3:dx
   sum1=sum1+abs(Vx(dy,y));
   sum2=sum2+abs(Vy(dy,y));
end
aa=sum1+sum2;
```

Este ciclo *for* inicia el proceso de distinción de hacia qué lado se ha de calcular la rotación del objeto, haciendo la sumatoria de los elementos de *Vx* y *Vy* que se encuentran en mismo renglón que el centro (en el eje horizontal) desde el límite de la izquierda (*d3*) hasta el centro mismo (*dx*). Se toma en cuenta el valor absoluto de estos valores por que lo que nos interesa en este momento es su magnitud, no su dirección, ya que ésta nos ayudará a considerar si hay cambios sustanciales o no. En *aa* se guarda el resultado de la suma de todas las velocidades, para llegar a una cifra de contenido de información final.

```
sum1=0; sum2=0;
for y=dx:d4
   sum1=sum1+abs(Vx(dy,y));
   sum2=sum2+abs(Vy(dy,y));
end
bb=sum1+sum2;
```

El mismo proceso es llevado a cabo nuevamente, pero ahora se calcula el nivel de contenido de información desde el centro (*dx*) hacia la derecha. El resultado es guardado en la variable *bb*.

```
if (aa>bb)
   x=round(dx-ds);
   for y=x:-1:d3
      a=dy+Vy(dy,y);
      b=y+Vx(dy,y);
      angs=atan((dy-a)/(dx-b))*180/pi;
      ang=ang+angs;
      ci=ci+1;
   end
```

Esta condición *if* compara los valores de *aa* y *bb*, para determinar si el primero es mayor que el segundo, y con ello definir hacia qué lado a partir del centro encontramos mayor cantidad de información que nos permita calcular un ángulo de rotación más preciso.

En el caso de que *aa* sea mayor, entonces el cálculo de los ángulos de rotación se hará hacia la izquierda del centro del campo de flujo óptico. Entonces, a partir de *ds* elementos a la izquierda de éste se calcula la pendiente definida como:

$$m = \frac{y_1 - y_2}{x_1 - x_2}$$

Como se aprecia en los límites del ciclo, el valor que va cambiando es el de las columnas, pues el renglón, como ya ha sido explicado, se mantiene constante.

El cálculo del ángulo obtenido para cada recta, es guardado en *angs*, La fórmula que define el resultado de esta variable hace el cálculo de la pendiente que, por funciones trigonométricas, corresponde al arco tangente del ángulo de inclinación en radianes. Para

una apreciación más práctica el resultado se entrega en grados, por lo que el valor obtenido hasta el momento es multiplicado por 180 y dividido entre Pi. (ver figura 3.6).

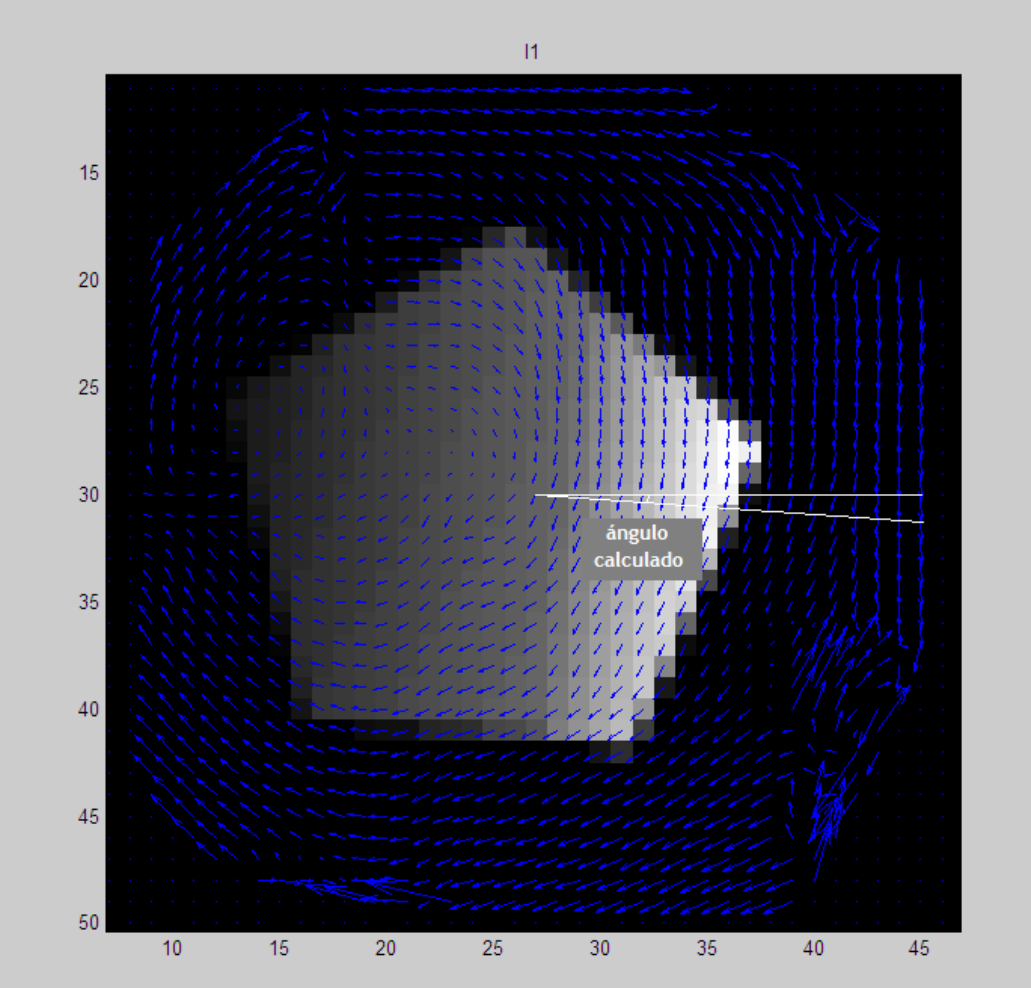


Figura 3.6 Cálculo de un ángulo que forma parte del campo de flujo óptico

ang guarda la suma de todos los ángulos calculados (uno por cada punto a la izquierda de *dx*, a partir de la distancia *ds*, hasta llegar al límite horizontal del campo de flujo óptico *d3*). *ci* es el encargado de contar el total del número de estos elementos.

```
else
   x=round(dx+ds);
```

```
   for y=x:d4
      a=dy+Vy(dy,y);
      b=y+Vx(dy,y);
      angs=atan((dy-a)/(dx-b))*180/pi;
      ang=ang+angs;
      cd=cd+1;
   end
end
```

En caso de que el lado derecho posea mayor información que el izquierdo, se sigue el mismo proceso del cálculo de los ángulos correspondientes, solo que en esta ocasión, el píxel de inicio a partir del cual se comienzan a calcular las pendientes de la recta que entre éste y el centro forman, será el encontrado en el renglón *dy*, y la columna *dx+ds*. Se seguirán calculando ángulos hasta llegar al límite del campo de flujo óptico en el eje horizontal, que en este caso es *d4*.

```
if ci==0
   ang=ang/cd;
else
   ang=ang/ci;
end
```

Como *ang* hasta el momento sólo es una suma de ángulos debe ser promediado entre el total de número de ángulos existente (*ci o cd*). Por ello, si el cálculo se hizo hacia la derecha del centro, *ci* es igual a cero y *cd* posee un valor entero, por lo que se obtiene

el ángulo promedio dividiendo *ang* entre *cd*. En caso de no ser así, *cd* posee entonces un valor igual a cero y *ci* tiene un valor entero, por lo que el promedio de los ángulos se define por la división de *ang* entre *ci*.

fprintf ('Rotación detectada de: %i grados \n', ang);

Finalmente, se despliega en la ventana de comandos de Matlab el ángulo de rotación calculado para la transición entre las dos imágenes activas.

3.5 Descripción de las imágenes en Fireworks

Los ejemplos del algoritmo de rastreo de movimiento implementado en esta tesis, obtienen los parámetros de desplazamiento y rotación entre dos imágenes subsecuentes. Estas imágenes son generadas de dos maneras, de acuerdo al ejemplo que se estudie.

Una de ellas es por medio del uso de un algoritmo que genera una matriz de ceros (fondo de la imagen) y posteriormente sitúa valores de 1 en las coordenadas que describen al objeto en movimiento. Este algoritmo ya ha sido explicado en la sección anterior.

La otra manera de generar las imágenes es por medio del software de Macromedia: Fireworks MX en su versión 6.0. Esta sección se ocupa de explicar el proceso de su creación.

Fireworks es un programa que permite diseñar, editar y optimizar gráficos para su uso en diversas aplicaciones, que van desde páginas web hasta diseño gráfico. Con esta

herramienta es posible dividir las imágenes y de esta forma obtener una mayor calidad de las mismas.

Macromedia Fireworks trabaja en conjunto con otros programas que permiten cargas previas, intercambio y control de las imágenes de una forma más eficiente. Esto lo cataloga como un software indispensable para el usuario que desarrolle este tipo de tecnología. Por su gran accesibilidad y facilidad de manejo se decidió emplear este software en el proceso de generación de imágenes de una secuencia. Este proceso consistió en una serie de pasos, descritos a continuación:

En primer lugar al iniciar el programa, se seleccionó la opción de realizar un archivo nuevo, consecutivamente se escogió el color de fondo y las dimensiones (en número de píxeles) en las que se desea el tamaño de la imagen. Para llevar a cabo el propósito de esta tesis, se eligió el tamaño de 100 x 100 píxeles, ya que a pesar de ser un tamaño relativamente pequeño, permite la ejemplificación correcta del algoritmo de rastreo implementado en Matlab, además de reducir los tiempos de procesamiento de la imagen.

Una vez realizada la selección de dimensiones, se utilizó una herramienta conocida como *vector*, en sus variantes de formas predeterminadas, con la que se realizó el diseño de figuras como polígonos y elipses que a su vez fueron modificadas para crear los objetos a utilizar en la secuencia. Por otro lado, a dichas figuras se les añadió una textura de relleno que permite que el campo de flujo óptico sea calculado de una forma más sencilla y eficaz.

Posteriormente, el objeto ya diseñado se colocó en una posición determinada en el plano de la imagen, y el archivo fue nombrado con la siguiente nomenclatura:
'letra, letra, número de imagen dentro de la secuencia'. Para el caso de ésta primera imagen, el número es el 1.

El archivo **.**PNG hace referencia a un formato de imágenes comprimidas conocido como *Gráficos Portátiles de Red* (Portable Network Graphics, por sus siglas en inglés), dicho formato, comprime las imágenes de forma que no haya pérdida de información (lo que significa que toda la información de la imagen queda restaurada cuando el archivo se

descomprime para su análisis), lo que permite al usuario realizar ajustes entre el tamaño de la imagen y su calidad.

Una vez nombrado el archivo, el objeto fue desplazado dentro de la imagen, trazando siempre un movimiento lineal. Hecho esto, se guardó nuevamente el archivo, pero ahora con el número 2, para hacer referencia a este como una imagen subsiguiente a la del primer archivo guardado.

Posteriormente se realizaron rotaciones del objeto mediante el uso de una herramienta de Fireworks conocida como Numeric Transform y con ella se estableció en grados, el ángulo de la rotación deseado, permitiéndonos definir exactamente este valor para su comparación posterior.

Por último, la imagen se guarda nuevamente como parte de la secuencia anterior, es decir, con el número de imagen 3. La complejidad y longitud de las secuencias es decisión del usuario. Para los ejemplos comprobados en esta tesis, se usaron secuencias de tres imágenes, ya que son suficientes para demostrar el cálculo de los parámetros que el algoritmo es capaz de medir.