

MEMORIAS DE LAS PRÁCTICAS DE

VISIÓN POR COMPUTADOR

Grupo:
Jose David Fernández Rodríguez
Cristóbal Carnero Liñán

1 PRÁCTICA 1

1.1 Apartado 1

Leemos la imagen lily.tif:

```
I = imread('lily.tif');
```

Convertimos la matriz de int8 [0..255] en color a double [0..1] en escala de grises:

```
r = double(I(:,:,1))./256;  
g = double(I(:,:,2))./256;  
b = double(I(:,:,3))./256;  
J = (r+g+b)/3;
```

Mostramos el histograma y la guardamos como escala de grises en formato TIFF:

```
imhist(J);  
imwrite(J,'lily_gris.tif', 'TIFF');
```

1.2 Apartado 2

Leemos y mostramos la imagen:

```
I = imread('lily_gris.tif');  
tam=size(I);  
subplot(2,2,1);  
imshow(I);
```

Calculamos y mostramos el umbral:

```
subplot(2,2,2);  
BW=I>100;  
imshow(BW);
```

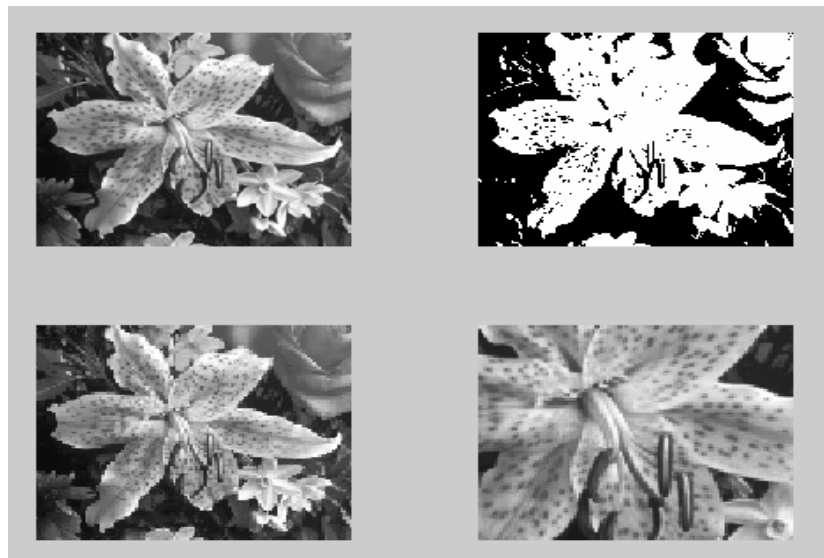
Calculamos la imagen con la mitad de resolución, tomando sólo filas y columnas alternas:

```
subplot(2,2,3);  
R=I(1:2:tam(1),1:2:tam(2));  
imshow(R);
```

Tomamos el recuadro central de la imagen con la mitad de tamaño:

```
subplot(2,2,4);  
Z1 = I((tam(1)/4):(3*tam(1)/4), (tam(2)/4):(3*tam(2)/4));  
imshow(Z1);
```

La salida del programa es la siguiente:



2 PRÁCTICA 2

2.1 Apartado 1 y 2

```
function practica2(file);
```

Definimos el filtro gaussiana (con el tamaño de la plantilla según lo especificado en los apuntes) y el media:

```
sigma = 0.5;  
c = 2*sqrt(2*sigma);  
w = ceil(3*c);  
gaus = fspecial('gaussian',2*w+1,sigma);  
media = fspecial('average', 3);
```

Leemos la imagen y la pasamos a double en el rango [0..1] y la mostramos:

```
im = double(imread(file))./256;  
imshow(im); title('imagen original');
```

Aplicamos ruido de sal y pimienta, y a su resultante, el filtro gaussiana, mediana y media:

```
im11 = imnoise(im,'salt & pepper', 0.02);  
im12 = filter2(gaus, im11);  
im13 = medfilt2(im11, [3 3]);  
im14 = filter2(media, im11);
```

A continuación mostramos los resultados:

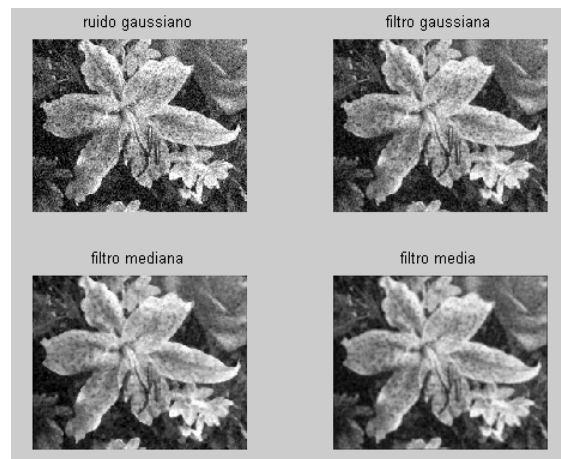
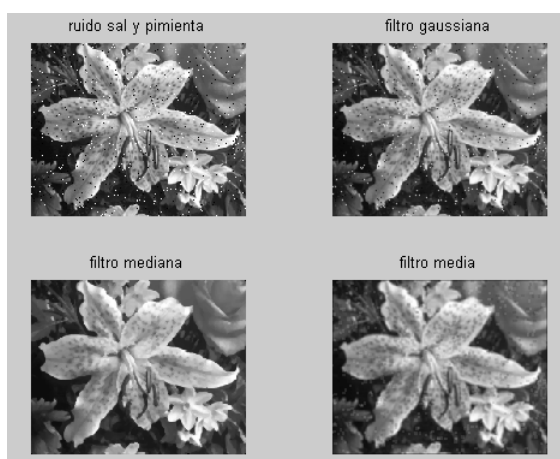
```
figure;  
subplot(2,2,1); imshow(im11); title('ruido sal y pimienta');  
subplot(2,2,2); imshow(im12); title('filtro gaussiana');  
subplot(2,2,3); imshow(im13); title('filtro mediana');  
subplot(2,2,4); imshow(im14); title('filtro media');
```

Ahora hacemos el mismo procedimiento pero añadiendo ruido gaussiano:

```
im21 = imnoise(im,'gaussian');  
im22 = filter2(gaus, im21);  
im23 = medfilt2(im21, [3 3]);  
im24 = filter2(media, im21);  
figure;  
subplot(2,2,1); imshow(im21); title('ruido gaussiano');  
subplot(2,2,2); imshow(im22); title('filtro gaussiana');  
subplot(2,2,3); imshow(im23); title('filtro mediana');  
subplot(2,2,4); imshow(im24); title('filtro media');
```

2.1.1 Conclusiones

Como se puede observar en la salida del script, para un ruido de tipo “sal y pimienta” el filtro más apropiado para su eliminación es el filtro mediana. Sin embargo, para ruido gaussiano, el propio filtro gaussiano obtiene mejores resultados, pero difumina la imagen.



3 PRÁCTICA 3

3.1 Apartado 1

```
im=imread('rice.tif');
```

Leída la imagen, la mostramos junto a su histograma:

```
%(apartado 1)
subplot(2,4,1); imshow(im); title('imagen original');
subplot(2,4,5); imhist(im); title('histograma original');
```

Observando el histograma, apreciamos que la mayor parte de los colores de la imagen están en el rango [40..230], por lo tanto realizamos la imagen expandiendo dicho rango:

```
%(apartado 2)
%a ojo, este parece un buen realce
subplot(2,4,2);
ima=imadjust(im,[40/255; 230/255],[0; 1]);
imshow(ima); title('imagen realzada (rango 40 230)');
subplot(2,4,6); imhist(ima); title('histograma realzado');
```

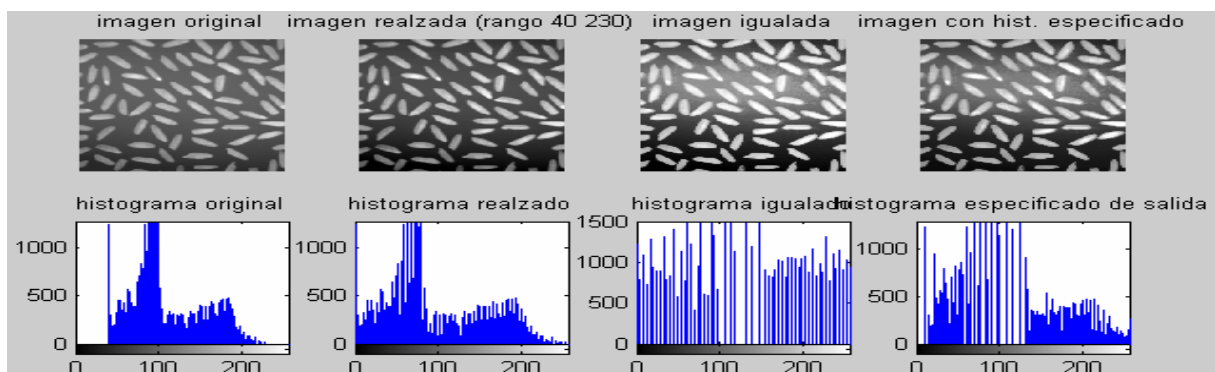
A continuación igualamos el histograma de la imagen (uniformizando así la distribución de colores en la medida que permite esta técnica):

```
%(apartado 3)
imb = histeq(im);
subplot(2,4,3); imshow(imb); title('imagen igualada');
subplot(2,4,7); imhist(imb); title('histograma igualado');
```

Ahora, leemos otra imagen para obtener su histograma, y modificamos nuestra imagen para adecuarlo al histograma dado (también mostramos la imagen y el histograma que usamos como entrada para la última de las modificaciones):

```
%(apartado 4)
lili = imread('lily_gris.tif');
histesp = imhist(lili);
imc = histeq(im, histesp);
subplot(2,4,4); imshow(imc); title('imagen con hist. especificado');
subplot(2,4,8); imhist(imc); title('histograma especificado de salida');
figure;
subplot(1,2,1); imshow(lili);
subplot(1,2,2); imhist(lili); title('histograma especificado de entrada');
```

A continuación se presenta una de las salidas del programa:



3.2 Apartado 2

```
function realce_local(file, K);
im=double(imread(file))./256;
[h v]=size(im);
imout(h,v) = 0; %para preasignar espacio a la matriz de salida
M=mean2(im);
```

Para cada píxel de la imagen, calculamos la media y desviación típica de una ventana en torno a dicho píxel. Si la desviación típica no es nula (ventana no uniforme), entonces aplicamos la fórmula de realce:

```
for j=2:v-1
    for i=2:h-1
        %entorno 3x3
        entorno = im(i-1:i+1, j-1:j+1);
        media=mean2(entorno);
        destipica=std2(entorno);

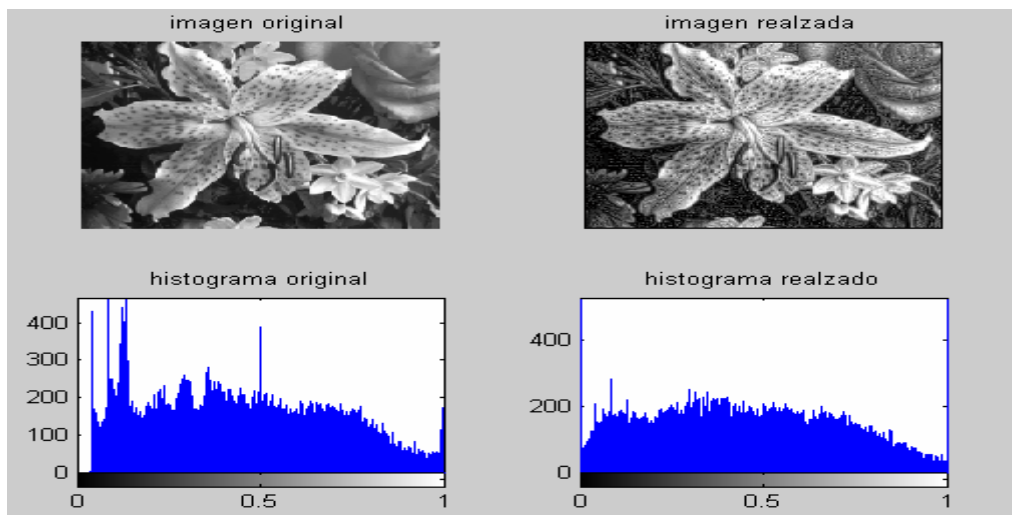
        if (destipica==0) A=1;
        else A=K*M/destipica;
        end;

        imout(i,j)=A*(im(i,j)-media)+media;
    end
end
```

Y mostramos los resultados:

```
figure;
subplot(2,2,1); imshow(im); title('imagen original');
subplot(2,2,3); imhist(im); title('histograma original');
subplot(2,2,2); imshow(imout); title('imagen realzada');
subplot(2,2,4); imhist(imout); title('histograma realzado');
```

Esta es la salida del algoritmo usando un factor de realce de 0.3:



4 PRÁCTICA 4

4.1 Apartado 1

```
function angulos=extraer_bordes_sobel(filename, umbral);
im = double(imread(filename))./255;
```

Con la función “fespecial” obtenemos el filtro de **Sobel** horizontal. De su transpuesta, obtenemos el filtro vertical. Aplicamos ambos filtros a la imagen obteniendo dos imágenes:

```
sobelh = fespecial('sobel');
sobelv = sobelh';
imv = abs(conv2(im, sobelv, 'same'));
imh = abs(conv2(im, sobelh, 'same'));
```

Calculamos los bordes a partir de los resultados de la aplicación del filtro de Sobel:

```
imbordes = (imv+imh)>umbral;
```

Y se lo añadimos a la imagen original, pero solamente en la componente verde.

```
imrgb(:,:,1) = im;
imrgb(:,:,2) = im+imbordes;
imrgb(:,:,3) = im;
```

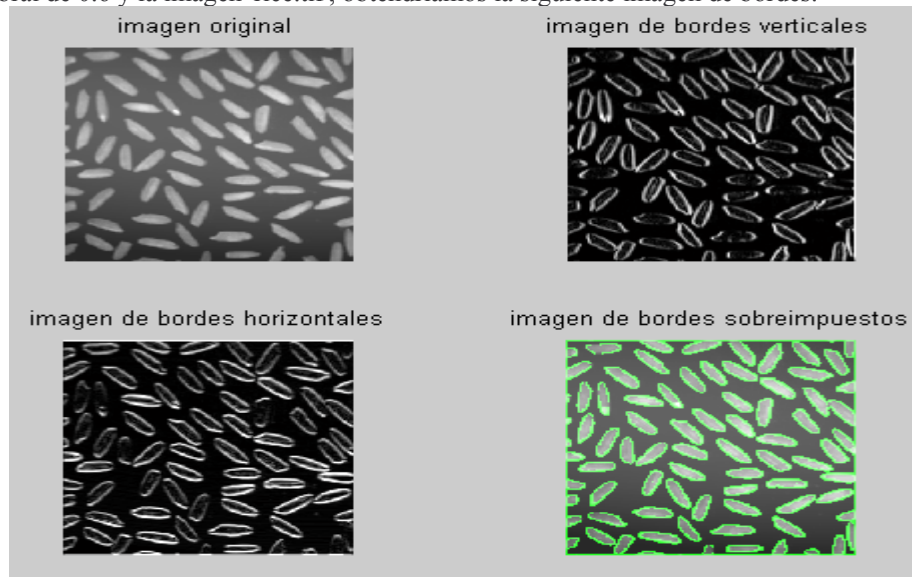
Mostramos los resultados:

```
subplot(2,2,1); imshow(im); title('imagen original');
subplot(2,2,2); imshow(imv); title('imagen de bordes verticales');
subplot(2,2,3); imshow(imh); title('imagen de bordes horizontales');
subplot(2,2,4); imshow(imrgb); title('imagen de bordes sobreimpuestos');
```

Por último calculamos la fase de los bordes:

```
angulos = atan(imv./imh);
```

Usando un umbral de 0.6 y la imagen 'rice.tif', obtendríamos la siguiente imagen de bordes:



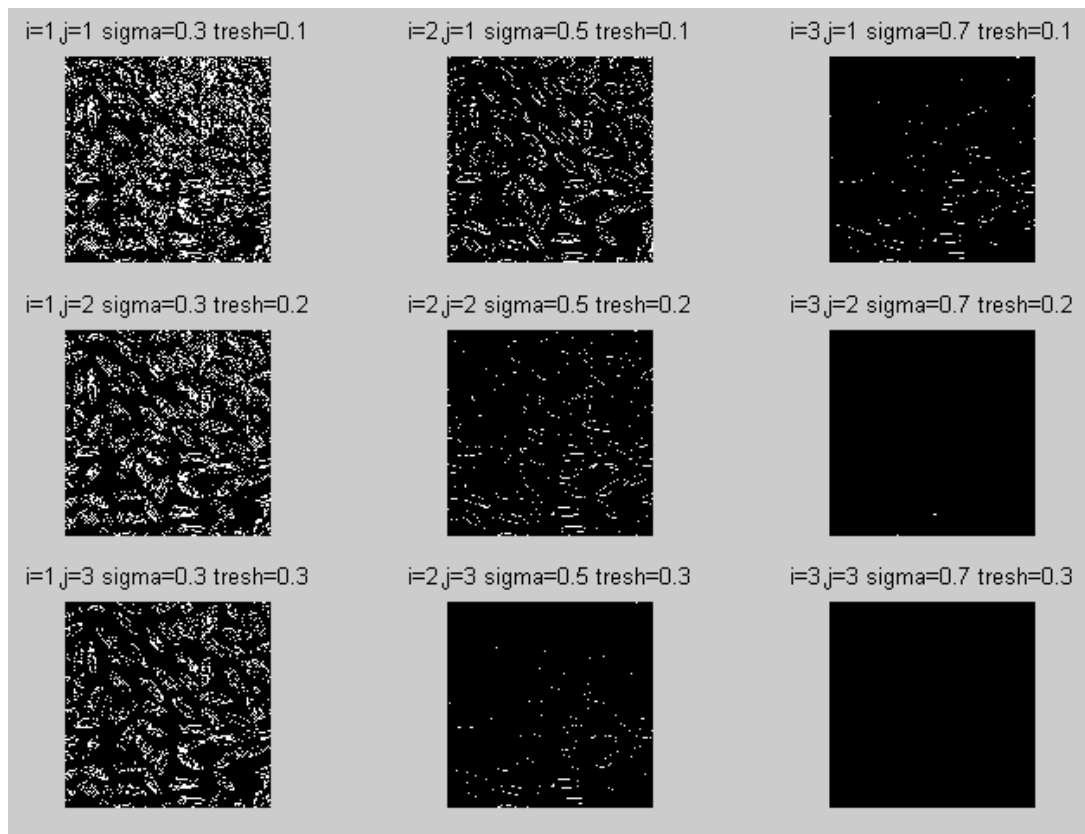
4.2 Apartado 2

Usamos el siguiente código para comparar la influencia de los distintos parámetros:

```
im=double(imread('rice.tif'))./255;

sigma=[0.3 0.5 0.7];
tresh=[0.1 0.2 0.3];
[dummy size_sigma] = size(sigma);
[dummy size_tresh] = size(tresh);
count=1;
for j=1:size_tresh
    for i=1:size_sigma
        subplot(size_tresh,size_sigma, count);
        count=count+1;
        imshow(edge(im,'log',tresh(j),sigma(i)));
        title(sprintf('i=%d,j=%d sigma=%g tresh=%g',i,j,sigma(i),tresh(j)));
    end;
end;
```

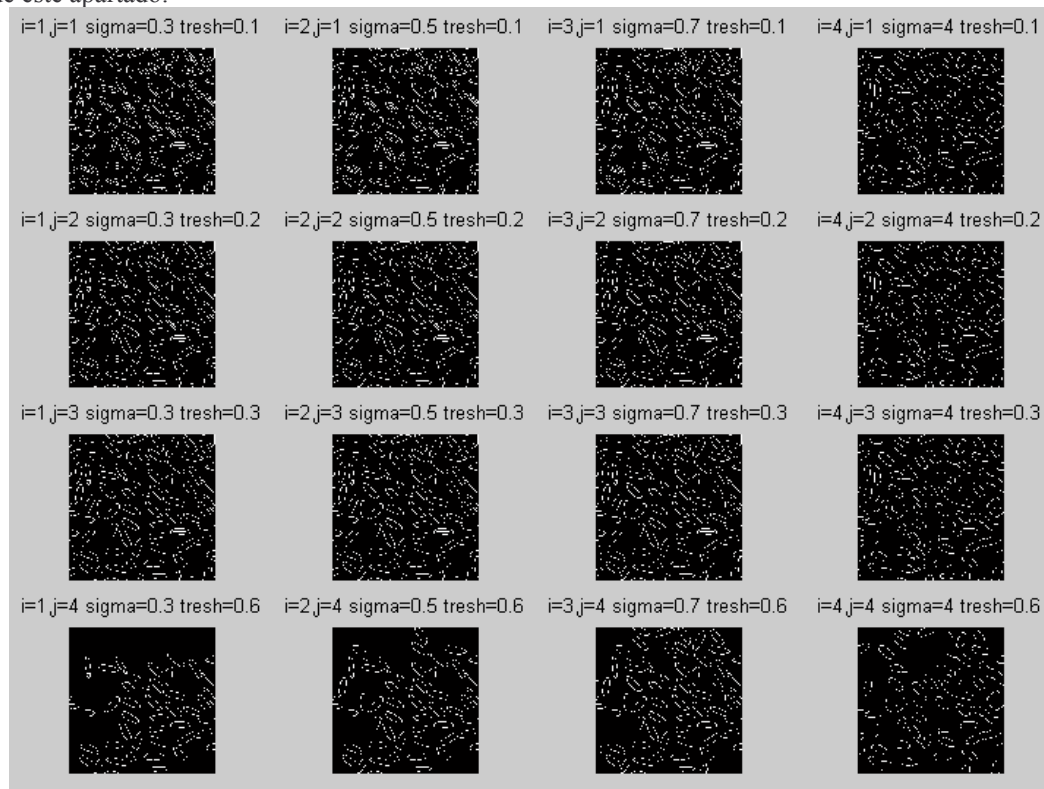
Obtenemos la siguiente salida:



De estos resultados deducimos que, conforme aumenta σ , y al suavizarse más la imagen antes de la detección de bordes, ante un *threshold* fijo, se van detectando menos bordes, al estar estos cada vez más difuminados. También observamos que conforme aumenta *threshold*, para un σ dado, se requiere que la intensidad mínima para detectar un borde sea mayor, por lo que se detectan menos bordes.

4.3 Apartado 3

Salida de este apartado:



Esto se consigue usando un código similar al anterior:

```
im=double(imread('rice.tif'))./255;

sigma=[0.3 0.5 0.7 4];
tresh=[0.1 0.2 0.3 0.6];
[dummy size_sigma] = size(sigma);
[dummy size_tresh] = size(tresh);
count=1;
for j=1:size_tresh
    for i=1:size_sigma
        subplot(size_tresh,size_sigma, count);
        count=count+1;
        imshow(edge(im, 'canny', tresh(j), sigma(i)));
        title(sprintf('i=%d,j=%d sigma=%g tresh=%g', i, j, sigma(i), tresh(j)));
    end;
end;
```

Conforme aumenta *threshold*, para un *sigma* dado, se requiere que la intensidad mínima para detectar un borde sea mayor, por lo que se detectan menos bordes. Ahora bien, distintos valores de *sigma*, para un *threshold* dado, detectan distintos rangos de frecuencias. Cuando es pequeña, se detectan bastantes bordes espurios. Conforme se hace mayor, detecta bordes más suaves pero también deja de detectar otros.

5 PRACTICA 5

5.1.1 Detección de rectas

La función toma los siguientes parámetros:

- nombre del fichero con la imagen (debe estar en escala de grises)
- umbral del modulo del gradiente para considerar los píxeles de los bordes
- umbral de votos para delimitar las regiones del espacio de parámetros que se corresponden a rectas.
- umbral de tamaño (número de contadores individuales) para considerar que una región del espacio de parámetros se corresponde con una recta (o sea, una región conectada cuyos contadores tienen todos un número de votos superior al umbral establecido)

Ejemplo de uso: *houghl('esc512.bmp',0.06,20,10)*

```
function imh = houghl(filename, umbralgrad, umbral_histeresis, umbral_size)
global ang_min; % == 1
global ang_max; % == máximo índice de la matriz de votos en ángulos
global mod_min; % == 1
global mod_max; % == máximo índice de la matriz de votos en módulos
global modulomax; % máxima magnitud del módulo del gradiente
```

Leemos la imagen, obtenemos el gradiente, y a partir del mismo, los módulos y ángulos del gradiente en cada píxel. Nótese que el gradiente se obtiene con ATAN, no con ATAN2, debido a que así distintos puntos pueden votar a la misma recta si yacen en ella, aunque sus ángulos de gradiente estén separados por π radianes (o sea, una recta que en unos segmentos separe una región oscura de otra clara, y en otros segmentos lo haga al revés, como ocurre en un tablero de ajedrez). También obtenemos la imagen umbralizada de módulos:

```
im = double(imread(filename))./255;
[sx sy] = size(im);
sizim = [sx sy];
[fx, fy] = gradient(im);
imgr=sqrt(fx.^2+fy.^2);
imbin=imgr>umbralgrad;
dirgr=atan(fx./fy);
```

Dimensionamos la matriz que contiene el espacio paramétrico. Antes de continuar, advertimos que al final se definen funciones de conversión entre el espacio de índices de la matriz y el espacio de parámetros, y que esta conversión es lineal, pero no 1:1

```
ang_min=1;
ang_max=360/2;
modulomax = sqrt(sx^2+sy^2);
mod_min = 1;
mod_max = round(modulomax/4);
contador_hough = zeros(ang_max, mod_max);
sizhough = size(contador_hough);
```


Obtenemos las coordenadas de los píxeles que superan el umbral estipulado del gradiente, y a continuación iteramos para cada uno de dichos puntos:

```
[r c] = find(imbin);  
for ind=1:length(r)  
    i = r(ind);  
    j = c(ind);
```

Obtenemos la dirección del gradiente del punto actual, que sería la de la correspondiente recta. Esto puede parecer raro, ya que el gradiente es perpendicular al ángulo de la recta con el eje de abscisas, pero es que estamos buscando el ángulo paramétrico de la recta, que se corresponde al de un segmento perpendicular a ésta:

```
angulo_grad = dirgr(i,j);
```

Pero no vamos a usar solamente un ángulo, sino un rango alrededor suyo, debido a la posible variabilidad local del ángulo. Será un rango $[-\pi/32, +\pi/32]$. También obtenemos los índices matriciales correspondientes a los límites del rango:

```
angulo_grad_min = angulo_grad - pi/32; %valor real del angulo minimo  
ang_grad_min = angle2index(angulo_grad_min); %indice del angulo minimo  
angulo_grad_max = angulo_grad + pi/32;  
ang_grad_max = angle2index(angulo_grad_max);
```

El espacio angular es modular, de modo que el rango de índices puede ir del menor al mayor ángulo o también al revés:

```
if (ang_grad_min < ang_grad_max) %rango normal  
    ang_indices = ang_grad_min:ang_grad_max;  
else %rango complementario  
    ang_indices = [ang_min:ang_grad_max, ang_grad_min:ang_max];  
end;
```

Así, obtenemos un rango de índices matriciales, cada uno de los cuales se corresponde a un ángulo discreto. Obtenemos dichos ángulos:

```
angulos = index2angle(ang_indices);
```

A partir de cada uno de esos ángulos, obtenemos los módulos de las rectas que tienen dicho ángulo y pasan por el punto del borde que estamos procesando. También traducimos dichos módulos a índices del espacio matricial:

```
modulos = i*cos(angulos)+j*sin(angulos);  
mod_indices = modulo2index(modulos);
```

Y así, aumentamos en 1 el nº de votos de cada celda del espacio matricial paramétrico correspondiente a una recta que pasa por el punto procesado, y terminamos el código que procesa cada uno de estos puntos (*end* del bucle *for*):

```
indices = sub2ind(sizhough, ang_indices, mod_indices);  
contador_hough(indices)=contador_hough(indices)+1;  
end;
```

Ya hemos terminado la primera fase del procesamiento. Mostramos los resultados obtenidos hasta ahora: la imagen original, la imagen de bordes (aplicando una LUT para que sean más visibles), la imagen de bordes umbralizada, y el espacio de parámetros, que puede ser visualizado como una imagen en escala de grises:

```
imh = contador_hough./max(max(contador_hough));  
subplot(2,2,1); imshow(im); title('imagen original');  
subplot(2,2,2); imshow(imadjust(imgr)); title('imagen de bordes');  
subplot(2,2,3); imshow(imbin); title('bordes superiores al umbral');  
subplot(2,2,4); imshow(imh); title('espacio paramétrico');
```

A continuación usaremos una matriz de contadores binarios para llevar cuenta de las rectas detectadas. Dicha matriz se corresponderá con el espacio paramétrico, y sus elementos a 1 indicarán los parámetros de las rectas detectadas. Por eficiencia, usaremos una matriz dispersa:

```
rectas_detectadas = sparse(sizhough(1), sizhough(2));
```

Umbralizamos la matriz de votos del espacio paramétrico según el umbral de histéresis, y etiquetamos los conjuntos conectados de contadores de votos que superen dicho umbral. Cada uno de dichos conjuntos se corresponde con una posible recta detectada:

```
imhisteresis = contador_hough > umbral_histeresis;
[imrectas numrectas] = bwlabel(imhisteresis,8);
```

Iteramos para cada posible recta detectada:

```
for numrecta=1:numrectas
```

Para cada posible recta detectada, determinamos si su área es igual o superior al tamaño estipulado por el parámetro `umbral_size`. Esto sirve para eliminar muchos conjuntos espúreos pequeños que se sitúan alrededor de los conjuntos principales de gran tamaño.

```
    mascara = (imrectas==numrecta);
    indices_rectan = find(mascara);
    if length(indices_rectan) < umbral_size
        continue;
    end;
```

Superado el filtro anterior, encontramos los contadores con máximo número de votos de todo el conjunto. Si son varios, calculamos los parámetros medios. Obsérvese que en todo momento estamos trabajando con índices matriciales más que con puntos del espacio de parámetros, pero dado que las funciones de transformación son lineales, no hay problema:

```
    puntos_rectan = contador_hough(indices_rectan);
    [maximales indices_maximales] = max(puntos_rectan);
    [r_max c_max] = ind2sub(sizzhough, indices_rectan(indices_maximales));
    row = round(mean(r_max));
    column = round(mean(c_max));
```

Finalmente, marcamos la posición de la recta detectada en el espacio paramétrico y terminamos el bucle *for*:

```
    disp(sprintf('aceptada recta del grupo nº %d', numrecta));
    rectas_detectadas(row,column)=1;
end;
```

Finalmente, obtenemos los ángulos y módulos de las rectas detectadas a partir de sus índices matriciales:

```
[r_max c_max] = find(rectas_detectadas);
angulos = index2angle(r_max);
modulos = index2modulo(c_max);
numrectas = length(angulos);
```

A continuación obtenemos 4 puntos para cada recta, que son los puntos de corte con

- 1: eje X
- 2: eje Y
- 3: eje X desplazado en `SIZE_X` píxeles (límite X de la imagen)
- 4: eje Y desplazado en `SIZE_Y` píxeles (límite Y de la imagen)

```
senos = sin(angulos);
cosenos = cos(angulos);
eje_x1(1:numrectas)=0;
eje_y1=modulos./senos;
eje_x2=modulos./cosenos;
eje_y2(1:numrectas)=0;
eje_x3(1:numrectas)=sx; eje_x3 = eje_x3';
eje_y3=(modulos-eje_x3.*cosenos)./senos;
eje_y4(1:numrectas)=sy; eje_y4 = eje_y4';
eje_x4=(modulos-eje_y4.*senos)./cosenos;
```

Para terminar, mostramos el espacio paramétrico de dos formas:

- como imagen en escala de grises con los puntos correspondientes a las rectas detectadas marcados en rojo
- como imagen coloreada en la que se muestran todos los conjuntos conectados de contadores de votos (en esta imagen se puede apreciar que se generan numerosos conjuntos pequeños espúreos, que se corresponden a regiones limitrofes de conjuntos más grandes)

```
figure;
imhough(:, :, 1) = imh+rectas_detectadas;
imhough(:, :, 2) = imh-rectas_detectadas;
imhough(:, :, 3) = imh-rectas_detectadas;
imshow(imhough); title('espacio paramétrico original');
figure;
imshow(label2rgb(imrectas)); title('espacio paramétrico con clusters');
```

También mostramos la imagen original con los bordes umbralizados y las rectas detectadas sobreimpresas. Para asegurarnos de que son visibles, hemos de dibujar dos segmentos, [(x1,y1) (x3,y3)] y [(x2,y2) (x4,y4)]:

```
figure;
imtotal(:,:,1) = im-imbin;
imtotal(:,:,2) = im+imbin;
imtotal(:,:,3) = im-imbin;
imshow(imtotal); title('líneas detectadas en el espacio cartesiano');
for ind=1:numrectas
    line([eje_y1(ind) eje_y3(ind)], [eje_x1(ind) eje_x3(ind)]);
    line([eje_y1(ind) eje_y4(ind)], [eje_x1(ind) eje_x4(ind)]);
    line([eje_y2(ind) eje_y3(ind)], [eje_x2(ind) eje_x3(ind)]);
    line([eje_y2(ind) eje_y4(ind)], [eje_x2(ind) eje_x4(ind)]);
end;
```

Por ultimo, tenemos las funciones de conversión índices_matriciales <-> espacio_de_ángulos e índices_matriciales <-> espacio_de_módulos (4 en total):

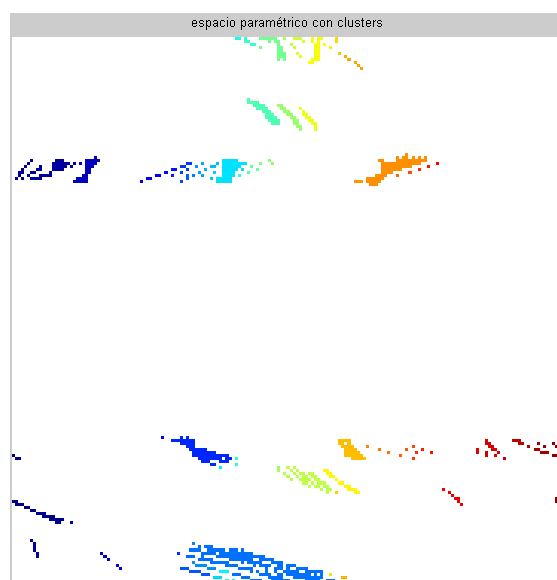
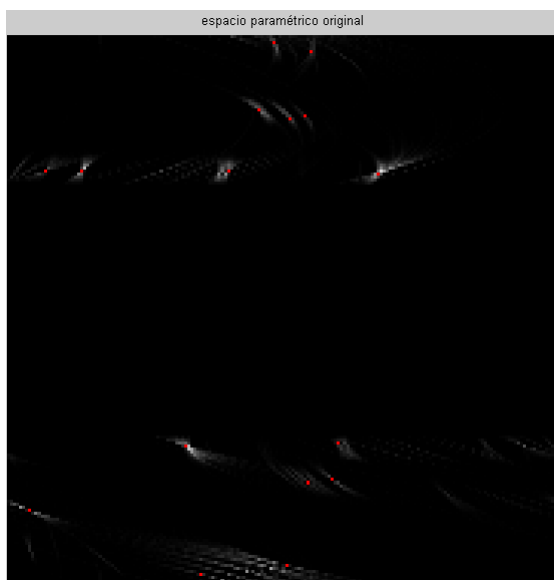
```
%de ángulo 0..2*pi a índice matricial 1..ang_max
function ang_ind = angle2index(angulo)
global ang_min; global ang_max;
angulo = mod(angulo, 2*pi); %reducimos el espacio angular al rango [0 2*pi]
ang_ind = round((angulo*(ang_max-ang_min)/(2*pi))+ang_min);
```

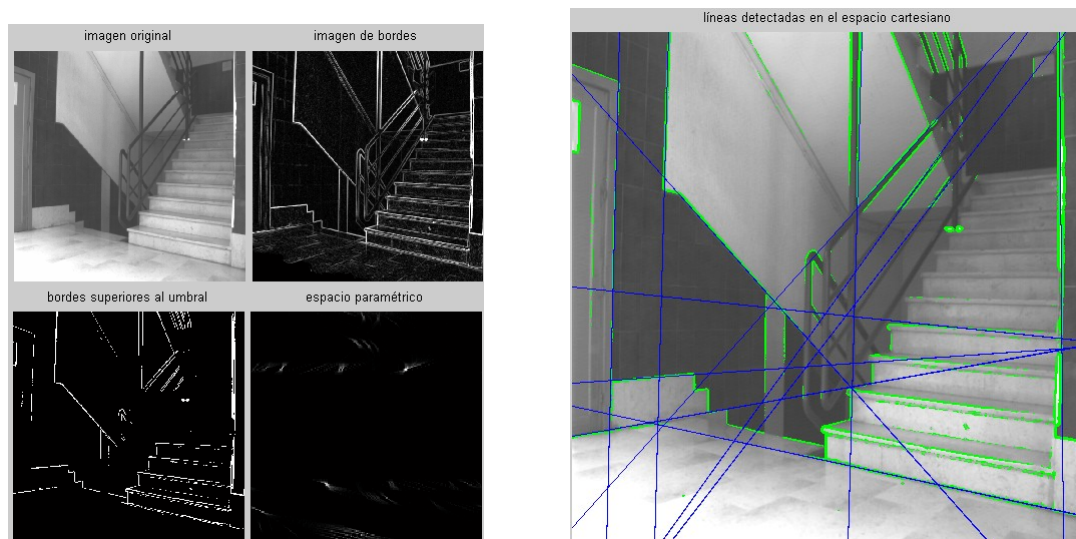
```
%de índice matricial 1..ang_max a ángulo 0..2*pi
function angulo = index2angle(ang)
global ang_min; global ang_max;
angulo = ((ang-ang_min)*2*pi/(ang_max-ang_min));
```

```
%de modulo 0..modulomax a índice matricial 1..mod_max
function mod_ind = modulo2index(modulo)
global mod_min; global mod_max; global modulomax;
modulo = mod(modulo, modulomax);
mod_ind = round((modulo*(mod_max-mod_min)/modulomax)+mod_min);
```

```
%de índice matricial 1..mod_max a modulo 0..modulomax
function modulo = index2modulo(mod)
global mod_min; global mod_max; global modulomax;
modulo = (mod-mod_min)*modulomax/(mod_max-mod_min);
```

Mostramos ahora la salida para la ejecución de *houghl('esc512.bmp',0.06,20,10)*:





5.2 Detección de círculos

Para la detección de círculos usaremos una metodología similar a la de las rectas. La función toma los siguientes parámetros:

- nombre del archivo de imagen (debe estar en escala de grises)
- umbral gradiente mínimo para que un punto del borde pueda votar
- radio mínimo de los círculos detectados.
- radio máximo (conviene que el rango de estos dos parámetros sea pequeño).
- umbral de votos para delimitar las regiones del espacio de parámetros que se corresponden a círculos.
- umbral de tamaño (número de contadores individuales) para considerar que una región del espacio de parámetros se corresponde con un círculo (o sea, una región conectada cuyos contadores tienen todos un número de votos superior al umbral establecido).

Ejemplo de uso: *houghc('blood1.tif', 0.15, 15, 25, 20, 15)*

```
function imh = houghc(filename, umbralgrad, radiomin, radiomax,
                    umbral_histeresis, umbral_size)
global radio_minimo;
global radio_maximo;
radio_minimo = radiomin;
radio_maximo = radiomax;
```

Leemos la imagen, calculamos gradiente, módulos y ángulos del gradiente, y mostramos la imagen con los bordes umbralizados sobreimpuestos:

```
im = double(imread(filename))./255;
[sx sy] = size(im);
sizim = [sx sy];
[fx, fy] = gradient(im);
imgr=sqrt(fx.^2+fy.^2);
imbin=imgr>umbralgrad;
dirgr=atan(fx./fy);

figure;
imtotal(:,:,1) = im-imbin;
imtotal(:,:,2) = im+imbin;
imtotal(:,:,3) = im-imbin;
imshow(imtotal); title('círculos detectados en el espacio cartesiano');
```

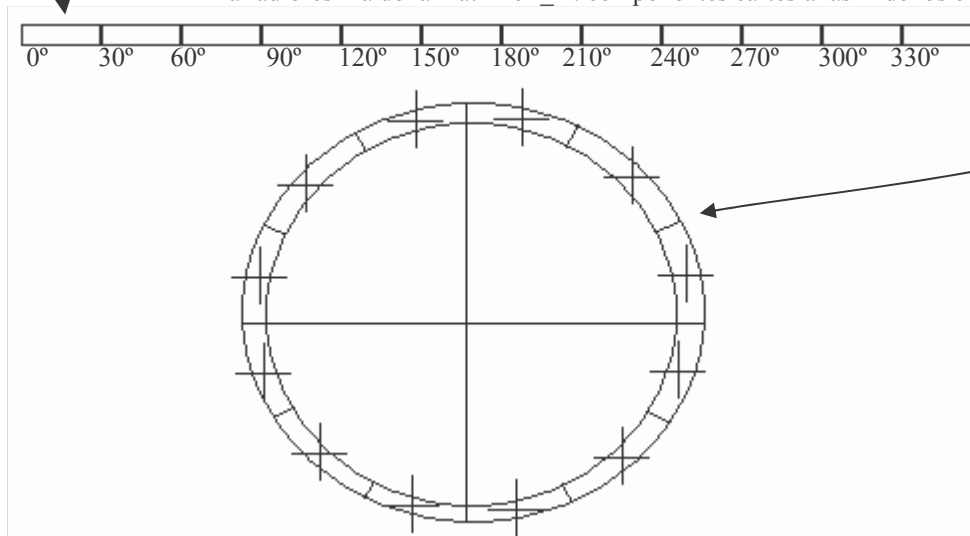
El espacio paramétrico será tridimensional, con dos coordenadas cartesianas (para caracterizar centros de círculos) y otra para caracterizar los radios. La correspondencia entre el tamaño del espacio de parámetros será 1:1. Por ello, el tamaño en las dimensiones cartesianas será $TAMAÑO_IMAGEN+2*RADIO_MAXIMO$, ya que los centros de los círculos pueden estar ubicados fuera de la imagen:

```
a_min = -radiomax;
a_max = sx+radiomax;
b_min = -radiomax;
b_max = sy+radiomax;
```

Vamos a generar un círculo de coordenadas cartesianas centradas en el origen para cada posible radio a detectar. Cada uno de estos círculos tendrá el número de píxeles necesarios para cubrir todo el círculo sin dejar huecos.

¿Cómo generamos dichos círculos? Muy sencillo:

- Para cada radio, tenemos una longitud en píxeles para el círculo correspondiente.
- Mapeamos un espacio angular $0..2\pi$ en un vector de dicha longitud.
- Cada componente de dicho vector se hace corresponder con una coordenada polar cuyo ángulo es el contenido de la componente, y su módulo el radio correspondiente.
- Traducimos dichas coordenadas polares a cartesianas, con lo que obtenemos, efectivamente, un círculo de píxeles de radio estipulado.
- Guardamos cada uno de estos conjuntos en:
 - componente radio-ésimo del vector `cir_npoints`: contiene el número de píxeles de un círculo para cada radio.
 - fila radio-ésima de la matriz `cir_x`: componentes cartesianas X de los círculos, para cada radio.
 - fila radio-ésima de la matriz `cir_y`: componentes cartesianas Y de los círculos, para cada radio.



¿Qué conseguimos con esto? Una forma rápida de incrementar los elementos de la matriz de contadores de votos: si el punto (x,y) forma parte del borde, todos los centros de los círculos de un determinado radio a los que podría pertenecer yacen en los píxeles $(x+cir_x, y+cir_y)$. Además, y es lo más interesante, mantenemos dichos píxeles indexados según su ángulo respecto al centro de su respectivo círculo:

```
maxnumpoints = ceil(2*pi*radiomax)+1;
numradios = radiomax-radiomin+1;
cir_x = zeros(numradios, maxnumpoints);
cir_y = zeros(size(cir_x));
cir_npoints = zeros(1, numradios);
for ind_radio=1:numradios
    [x y npoints] = circulo(index2radio(ind_radio));
    cir_npoints(ind_radio) = npoints;
    cir_x(ind_radio,1:npoints) = x;
    cir_y(ind_radio,1:npoints) = y;
end;
```

Inicializamos la matriz de contadores:

```
contador_hough = zeros(coord2index(a_max), coord2index(b_max), numradios);
sizough = size(contador_hough);
```

Obtenemos las coordenadas de los puntos de los bordes:

```
[r c] = find(imbin);
```

Procesamos cada punto del borde que supere el umbral:

```
for ind=1:length(r)
    i = r(ind);
    j = c(ind);
```

Convertimos las coordenadas del píxel del borde a índices de la matriz de contadores, y obtenemos el ángulo de su gradiente:

```

coord_i = coord2index(i);
coord_j = coord2index(j);
angulo_grad = dirgr(i,j);

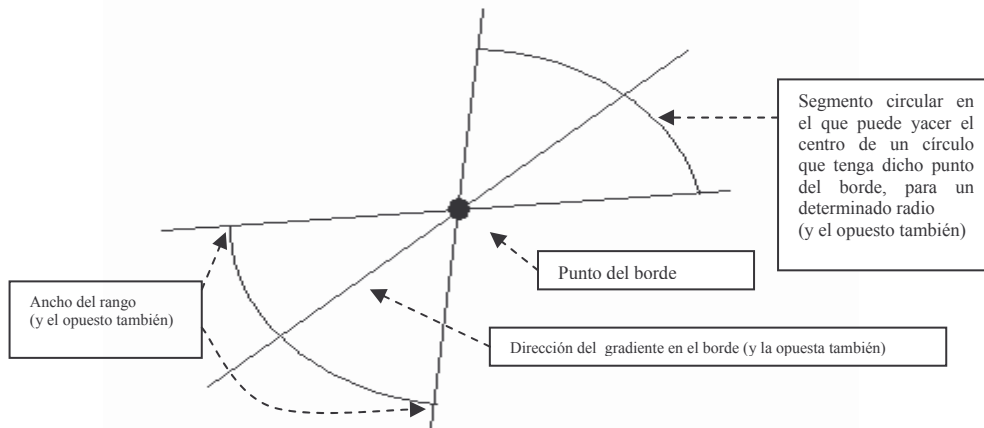
```

Ahora, para cada radio de búsqueda de círculos:

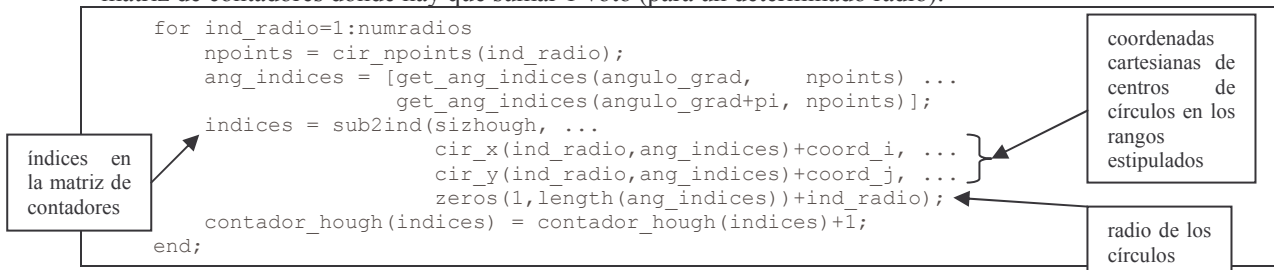
-Obtenemos el número de píxeles del círculo.

-No buscamos los centros solo en la dirección del gradiente, ya que ésta puede variar mucho localmente. En vez de ello, tomamos un rango de posibles ángulos en torno al ángulo del gradiente. Un aspecto importante es que, como el gradiente depende de si se separa una región oscura de una blanca y viceversa, hay que buscar también en la dirección opuesta a la del gradiente (sumándole PI radianes).

-Dichos rangos de ángulos se traducen a índices en los vectores de coordenadas cartesianas de píxeles de círculos, de modo que para obtener los puntos de dicho círculo que están en dichos rangos de ángulos, simplemente tomamos las coordenadas cartesianas de los píxeles que corresponden a cada rango de ángulos.



-Y una vez obtenemos las coordenadas cartesianas de cada posible centro (para un determinado radio), las desplazamos respecto a las coordenadas del píxel del borde, y de este modo tenemos las coordenadas de la matriz de contadores donde hay que sumar 1 voto (para un determinado radio):



Y así con lo anterior, ya hemos finalizado el bucle *for* que procesa los píxeles del borde:

```
end;
```

Ya hemos terminado la fase electoral (de votos). Ahora, para detectar los círculos, procedemos de la misma manera que en el caso de la detección de rectas: umbralizamos la matriz de contadores, buscamos conjuntos conectados de contadores, desechamos los conjuntos que sean más pequeños que el umbral de tamaño estipulado, y para el resto de conjuntos los centros de círculos detectados están en los máximos locales de cada conjunto:

```

imhisteresis = contador_hough > umbral_histeresis;
[imcircuitos numcircuitos] = bwlabeln(imhisteresis,26);
circuitos_detectados = zeros(sizhough);
for numcirculo=1:numcircuitos
    mascara = (imcircuitos==numcirculo);
    indices_circulon = find(mascara);
    if length(indices_circulon) < umbral_size; continue; end;
    puntos_circulon = contador_hough(indices_circulon);
    [maximales indices_maximales] = max(puntos_circulon);
    [row_max col_max radio_max] = ind2sub(sizhough, indices_circulon(indices_maximales));
    row = round(mean(row_max));
    column = round(mean(col_max));
    radio = round(mean(radio_max));
    circuitos_detectados(row,column,radio)=1;
end;
[indices] = find(circuitos_detectados);

```

Para mostrar los círculos sobreimpresos a la imagen original, pintamos aproximaciones poligonales tomando uno de cada cinco píxeles de cada círculo:

```

[a b r] = ind2sub(sizhough, indices);
i = index2coord(a);
j = index2coord(b);

for ind=1:length(i)
    indices = 1:5:cir_npoints(r(ind));
    line(cir_y(r(ind),indices)+j(ind), ...
        cir_x(r(ind),indices)+i(ind));
end;

```

Y para visualizar el espacio paramétrico usamos *contourslice*. En la captura se puede apreciar que las regiones que representan círculos detectados se corresponden con superficies cónicas con el eje en la dimensión de los radios, en cuyo centros se encuentran las coordenadas paramétricas de los círculos detectados. Estas superficies se configuran así debido a que el centro del círculo de un determinado radio se configura como una región con gran cantidad de votos, mientras que para círculos de radios mayores o menores, los posibles centros caen en circunferencias concéntricas:

```

imh = contador_hough/max(max(max(contador_hough)));
figure;
phandles = contourslice(imh,[],[],1:numradios);
view([-70 70]); axis tight
return;

```

Ahora mostramos las funciones auxiliares. La primera calcula las coordenadas cartesianas de los píxeles de un círculo de radio estipulado, siguiendo el método ya expuesto:

- usamos un vector cuya longitud es el número de píxeles del círculo
- particionamos el espacio angular $0..2\pi$ en pasos discretos para dicho vector
- transformamos las coordenadas polares en las que cada punto tiene de módulo el radio estipulado y de ángulo el correspondiente a cada componente del vector, en coordenadas cartesianas:

```

function [x,y,numpoints] = circulo(radio)
numpoints = ceil(2*pi*radio)+1;
angulos = index2angle(1:numpoints, numpoints);
modulos = ones(1,numpoints) * radio;
[x,y] = pol2cart(angulos, modulos);
x=round(x);
y=round(y);

```

Las dos siguientes funciones auxiliares transforman ángulos en índices de vectores y viceversa, para una determinada longitud en píxeles del espacio angular (numpoints):

```

function ang_ind = angle2index(angulo, numpoints) %ángulo va de 0 a 2pi
angulo = mod(angulo, 2*pi);
ang_ind = round(angulo*(numpoints-1)/(2*pi))+1;

```

```

function angulo = index2angle(ang, numpoints)
angulo = (ang-1)*2*pi/(numpoints-1);

```

Las transformaciones de radio a índice de la matriz de contadores y viceversa son triviales:

<pre>function radio_ind = radio2index(r) global radio_minimo; radio_ind = r-radio_minimo+1;</pre>	<pre>function radio = index2radio(radio_ind) global radio_minimo; radio = radio_ind-1+radio_minimo;</pre>
---	---

También son sencillas las transformaciones de coordenadas X e Y de un posible centro a índices de la matriz de contadores, que únicamente han de tener en cuenta que la matriz de contadores está ampliada respecto al tamaño de la imagen:

<pre>function coord_ind = coord2index(coord) global radio_maximo; coord_ind = coord+radio_maximo+1;</pre>	<pre>function coord = index2coord(coord_ind) global radio_maximo; coord = coord_ind-1-radio_maximo;</pre>
---	---

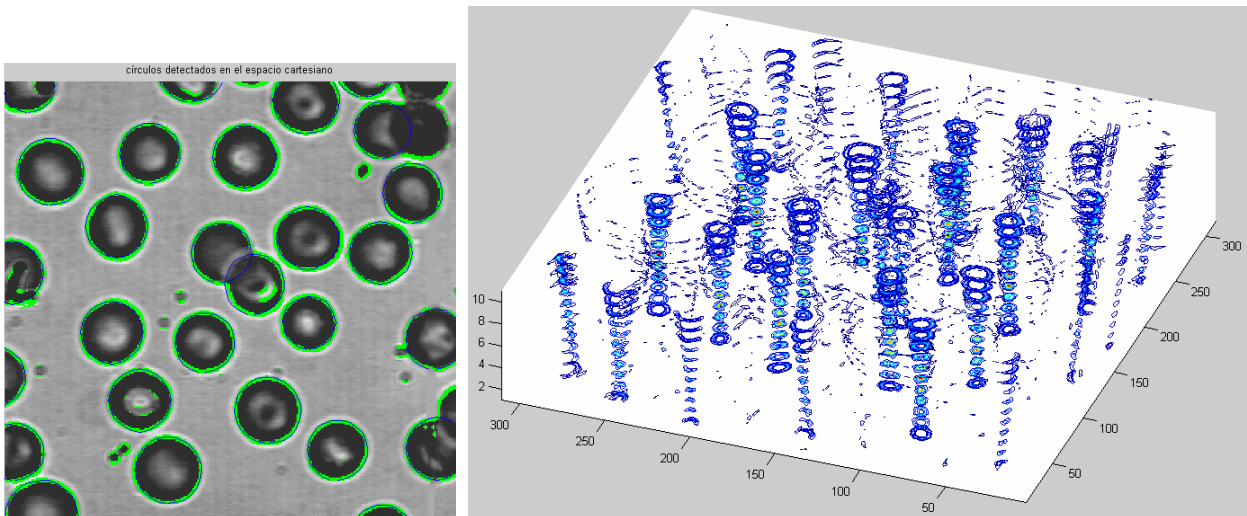
Por último, esta auxiliar sirve para obtener un rango de índices que se corresponden a los píxeles que pueden ser centros de un círculo. Para ello, toma 2 argumentos:

- un ángulo de gradiente (del píxel del borde que puede ser parte de un círculo)
- un número de píxeles (que es la longitud del vector de coordenadas de píxeles de un círculo con un determinado radio)

Con estos dos argumentos, la función calcula un rango de ángulos y lo traduce a un rango de índices dentro del vector de coordenadas de píxeles correspondiente (ya que los píxeles se pueden indexar por ángulos):

<pre>function ang_indices = get_ang_indices(angulo_grad, npoints) angulo_grad_min = angulo_grad - pi/16; %valor real del angulo minimo ang_grad_min = angle2index(angulo_grad_min, npoints); %indice del angulo minimo angulo_grad_max = angulo_grad + pi/16; ang_grad_max = angle2index(angulo_grad_max, npoints); if (ang_grad_min < ang_grad_max) %rango normal ang_indices = ang_grad_min:ang_grad_max; else %usamos el rango complementario ang_indices = [1:ang_grad_max, ang_grad_min:npoints]; end;</pre>

Veamos la salida para la entrada *houghc('blood1.tif', 0.15, 15, 25, 20,15)*:



6 PRÁCTICA 6

Definimos las variables globales que necesita "crec_recur", leemos la imagen y la mostramos:

<pre>global region; %region que va creciendo. Es una matriz binaria del tamaño de la imagen global media; %media (dinamica) de la region que va creciendo global points_in_region; %numero de puntos en la region que crece global im1; %imagen global t; %desviacion de nivel de gris sobre la media para añadir nuevo pixel im1 = double(imread('rice.tif'))./255; imshow(im1);</pre>

Obtenemos las dimensiones de la imagen y la semilla, que será elegida por el usuario:

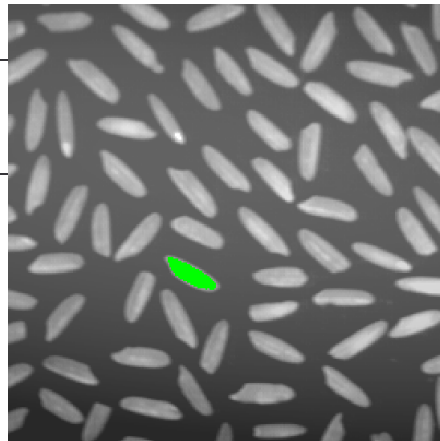

```
[sy sx] = size(im1);
[x y] = ginput(1);
x=round(x);
y=round(y);
```

Inicializamos las variables y ejecutamos el algoritmo de crecimiento recursivo:

```
region(1:sy,1:sx)=0;
media=im1(y,x);
points_in_region=0;
t = 0.1;
crec_recur(x,y);
```

Añadimos la información de la región a la imagen original en el canal verde y mostramos el resultado:

```
imtotal(:,:,1) = im1-region;
imtotal(:,:,2) = im1+region;
imtotal(:,:,3) = im1-region;
imshow(imtotal);
```



Y vemos la salida de la práctica:

7 PRÁCTICA 7

```
global region; %region que va creciendo. Es una matriz binaria del tamaño de la imagen
global media; %media (dinamica) de la region que va creciendo
global points_in_region; %numero de puntos en la region que crece
global im1; %imagen
global t; %desviacion de nivel de gris sobre la media para añadir nuevo pixel
im1 = double(imread('rice.tif'))./255;
imshow(im1);
[sy sx] = size(im1);
[x y] = ginput(1);
x=round(x);
y=round(y);
```

En primer lugar se calcula la región como en el apartado anterior:

```
region(1:sy,1:sx)=0;
media=im1(y,x);
points_in_region=0;
t = 0.1;
crec_recur(x,y);
imtotal(:,:,1) = im1-region;
imtotal(:,:,2) = im1+region;
imtotal(:,:,3) = im1-region;
```

Calculamos la suma de las coordenadas de todos los puntos pertenecientes a la región, así como el número de puntos que contiene. Con esos datos calculamos el centroide:

```

sumax=0;
sumay=0;
puntos=0;
for j=1:sy
    for i=1:sx
        if (region(j,i)>0)
            sumax=sumax+i;
            sumay=sumay+j;
            puntos=puntos+1;
        end;
    end;
end;

centroidx=sumax/puntos;
centroidey=sumay/puntos;

```

Usando los momentos centrales calculamos el ángulo de la región:

```

mu11 = 0;      mu20 = 0;      mu02 = 0;

for j=1:sy
    for i=1:sx
        if (region(j,i)>0)
            termx = (i-centroidx);
            termy = (j-centroidey);
            mu11 = mu11 + termx*termy; % *region(j,i)==1
            mu20 = mu20 + termx*termx; % *region(j,i)==1
            mu02 = mu02 + termy*termy; % *region(j,i)==1
        end;
    end;
end;

angulo = 0.5*atan2(2*mu11, (mu20 - mu02));

```

A continuación representamos los resultados obtenidos:

```

imshow(imtotal);

centroidx_p = round(centroidx);
centroidey_p = round(centroidey);

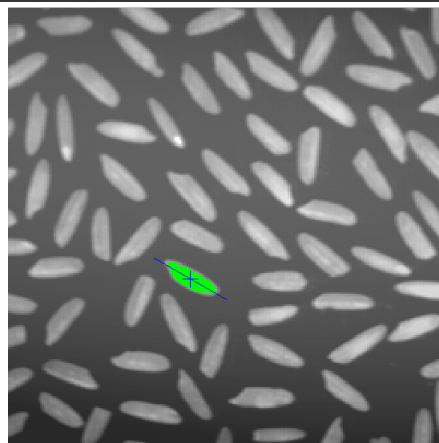
modulo = puntos/10;
eje_x1 = centroidx_p-modulo*cos(angulo);
eje_y1 = centroidey_p-modulo*sin(angulo);
eje_x2 = centroidx_p+modulo*cos(angulo);
eje_y2 = centroidey_p+modulo*sin(angulo);

line([centroidx_p-5 centroidx_p+5], [centroidey_p centroidey_p]);
line([centroidx_p centroidx_p], [centroidey_p-5 centroidey_p+5]);

line([eje_x1 eje_x2], [eje_y1 eje_y2]);

```

Y vemos la salida de la práctica,
que dibuja el centroide y el eje principal:



8 PRACTICA 8

8.1 APARTADO 1

```
function [detectados_l, detectados_r] = prac8_1(tresh, umbral_correlacion,
pixels_alto_búsqueda)
%devuelve los puntos detectados en ambas imágenes en coordenadas
%homogéneas: size(detectados_l) = size(detectados_r) = [3 N°Detectados]
```

Los parámetros que acepta la rutina implementada son los siguientes:

- *tresh*: umbral para el operador de Harris.
- *Umbral de correlación*.
- *pixels_alto_búsqueda*: indica la máxima distancia a la que se pueden encontrar los *corners*.

En primer lugar se inicializan las variables y se cargan las imágenes:

```
filename_r = 'pepsi_right.tif';
filename_l = 'pepsi_left.tif';

iml=double(imread(filename_l));
imr=double(imread(filename_r));

[srx sry] = size(imr);
[slx sly] = size(iml);

sigma=1;
radius = 2;
display=0;
```

Ahora se usa el operador de Harris que nos devolverá listas de *corners* de ambas imágenes:

```
[ciml, rl, cl] = harris(iml, sigma, tresh, radius, display);
[cimr, rr, cr] = harris(imr, sigma, tresh, radius, display);

disp(sprintf('Puntos de Harris: L=%d R=%d', length(rl), length(rr)));
```

A continuación se define el tamaño de las ventanas que se usará para calcular la correlación en un píxel. Además se crean imágenes con un margen (negro) de tamaño adecuado para poder calcular la correlación sin problemas:

```
size_ventana = 5;

iml_aumentado = zeros(size_ventana*2+slx, size_ventana*2+sly);
iml_aumentado((1:slx)+size_ventana, (1:sly)+size_ventana)=iml;

imr_aumentado = zeros(size_ventana*2+srx, size_ventana*2+sry);
imr_aumentado((1:srx)+size_ventana, (1:sry)+size_ventana)=imr;
```

Se inicializa una lista, que contendrá pares de *corners* detectados en ambas imágenes, así como un contador de la cantidad de puntos detectados:

```
detectados = zeros(4, length(rl));
ind_detectados = 1;
```

Por fin, el algoritmo que calcula correspondencias entre los *corners* de ambas imágenes. Para cada punto encontrado por el operador de Harris para la primera imagen, se busca su correspondiente en la segunda imagen (si existe).

```
for indl=1:length(rl);
    if (mod(indl,100)==0); disp(sprintf('Procesados %d puntos', indl)); end;
```

Para calcular la correspondencia con uno de los *corners*, se busca entre los *corners* de la otra imagen la más parecida, usando para ello la correlación. El algoritmo coge una ventana en el entorno del punto origen. A

continuación, para cada *corner* de la otra imagen se obtiene su ventana y se calcula la correlación. Nos quedamos siempre con el punto más “parecido”:

```
    ventana_l = iml_aumentado(rl(indl):rl(indl)+2*size_ventana,
    cl(indl):cl(indl)+2*size_ventana);

    % Para la búsqueda de la mejor correlacion:
    max_correlacion=-2;
    max_cor_ind=0;
    for indr=1:length(rr);

        if (abs(rl(indl) - rr(indr)) > pixels_alto_búsqueda);
            continue;
        end;

        ventana_r = imr_aumentado(rr(indr):rr(indr)+2*size_ventana,
        cr(indr):cr(indr)+2*size_ventana);

        %correlacion entre -1 y 1
        correlacion = corr2(ventana_l, ventana_r);

        if correlacion > max_correlacion
            max_correlacion=correlacion;
            max_cor_ind=indr;
        end;
    end;
end;
```

Finalmente, si la correlación entre los dos puntos es suficientemente grande (en comparación con un umbral) hemos encontrado un emparejamiento correcto para este *corner*.

```
    if (max_correlacion>=umbral_correlacion)
        detectados(:,ind_detectados) = ...
            [rl(indl) cl(indl) rr(max_cor_ind) cr(max_cor_ind)]';
        ind_detectados = ind_detectados+1;
    end;
end;
```

Para terminar mostramos los resultados representando en cada imagen los *corners* emparejados:

```
disp(sprintf('Concordancias detectadas: %d', ind_detectados-1));

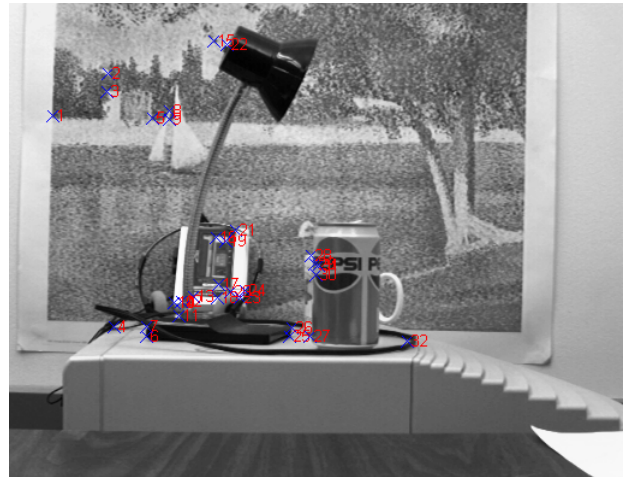
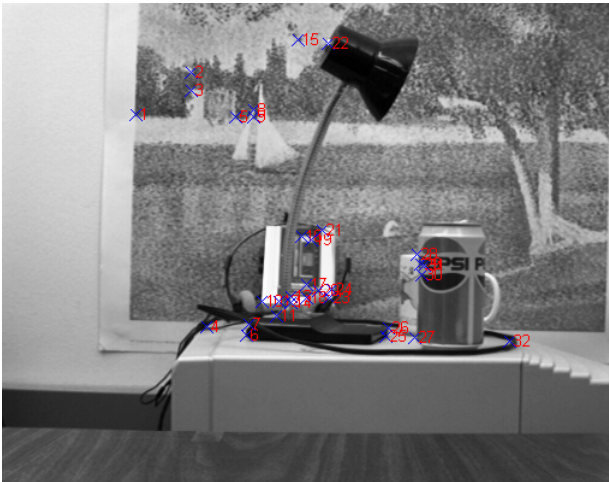
if (ind_detectados>1);
    detectados_l = ones(3, ind_detectados-1);
    detectados_r = ones(3, ind_detectados-1);
    detectados_l(1:2,:) = detectados(1:2, 1:ind_detectados-1);
    detectados_r(1:2,:) = detectados(3:4, 1:ind_detectados-1);
end;

figure;
imshow(iml./255);
for ind=1:ind_detectados-1;
    rl = detectados(1, ind);
    cl = detectados(2, ind);
    line([cl+5 cl-5], [rl+5 rl-5]);
    line([cl-5 cl+5], [rl+5 rl-5]);
    text(cl, rl,sprintf(' %d',ind), 'Color', 'red');
end;

figure;
imshow(imr./255);
for ind=1:ind_detectados-1;
    rr = detectados(3, ind);
    cr = detectados(4, ind);
    line([cr+5 cr-5], [rr+5 rr-5]);
    line([cr-5 cr+5], [rr+5 rr-5]);
    text(cr, rr,sprintf(' %d',ind), 'Color', 'red');
end;
```

Se muestra a continuación las imágenes de salida con la siguiente llamada al algoritmo:

```
prac8_1(8000, 0.8, 5)
```



8.2 APARTADO 2

Este apartado es muy parecido al anterior, excepto que toma como parámetros uno adicional:

- umbral para el operador de Harris
- umbral de correlación
- máxima distancia vertical entre los puntos entre los cuales se hace correspondencia
- máxima distancia horizontal entre los puntos entre los cuales se hace correspondencia (este valor debe ser elevado, pus las imágenes usadas presentan un alto grado de desplazamiento horizontal y pequeño en vertical)

Un ejemplo de uso es *prac8_2(8000, 0.8, 5, 150)*:

```
function [detectados_l, detectados_r] = prac8_2(tresh, umbral_correlacion,
        pixels_alto_busqueda, pixels_ancho_busqueda)
```

Obtenemos las imágenes:

```
filename_r = 'pepsi_right.tif';    filename_l = 'pepsi_left.tif';
iml=double(imread(filename_l));    imr=double(imread(filename_r));
[srx sry] = size(imr);             [slx sly] = size(iml);
```

Aplicamos el operador de Harris a la imagen izquierda:

```
sigma=1; radius = 2; display=0;
[ciml, rl, cl] = harris(iml, sigma, tresh, radius, display);
disp(sprintf('Puntos de Harris en la imagen izquierda: %d', length(rl)));
```

Usamos una ventana de $5*2+1 = 11$ píxeles de lado. Para aplicarla sin problemas, aumentamos el tamaño de las imágenes rellenando los bordes con ceros:

```
size_ventana = 5;
iml_aumentado = zeros(size_ventana*2+slx, size_ventana*2+sly);
iml_aumentado((1:slx)+size_ventana,(1:sly)+size_ventana)=iml;
imr_aumentado = zeros(size_ventana*2+srx, size_ventana*2+sry);
imr_aumentado((1:srx)+size_ventana,(1:sry)+size_ventana)=imr;
```

Reservamos espacio para los pares de correspondencia que podamos detectar:

```
detectados = zeros(4, length(rl));
ind_detectados = 1;
```

Iteramos para cada punto detecto por el operador de Harris, tomando una ventana alrededor suya:

```

tam = size(imr);
for indl=1:length(rl);
    ventana_l = iml_aumentado(rl(indl):rl(indl)+2*size_ventana, ...
                               cl(indl):cl(indl)+2*size_ventana);

```

Dentro del bucle anterior, iteramos para cada píxel de la imagen derecha que esté dentro de la ventana definida por los argumentos `pixels_alto_búsqueda` y `pixels_ancho_búsqueda`, buscando el píxel cuyo entorno presente la máxima correspondencia con el entorno del punto de Harris de la iteración actual:

```

    max_correlacion=-2;
    max_cor_r=0;
    max_cor_c=0;
    for rr=max(1,rl(indl)-pixels_alto_búsqueda):min(tam(1),rl(indl)+pixels_alto_búsqueda);
        for cr=...
            max(1,cl(indl)-pixels_ancho_búsqueda):min(tam(2),cl(indl)+pixels_ancho_búsqueda);
                ventana_r = imr_aumentado(rr:rr+2*size_ventana, cr:cr+2*size_ventana);
                correlacion = corr2(ventana_l, ventana_r);
                if correlacion > max_correlacion
                    max_correlacion=correlacion;
                    max_cor_r=rr;
                    max_cor_c=cr;
                end;
            end;
        end;
    end;

```

Finalizamos la iteración del bucle *for* para cada punto de Harris de la imagen izquierda, comprobando si el píxel de la imagen derecha con mayor correlación supera el umbral estipulado:

```

    if (max_correlacion>=umbral_correlacion)
        detectados(:,ind_detectados) = ...
            [rl(indl) cl(indl) max_cor_r max_cor_c]';
        ind_detectados = ind_detectados+1;
    end;
end;

```

Organizamos los argumentos de salida, que son puntos homogéneos (2 coordenadas cartesianas, 3 homogéneas), aunque no se usan debido a que no sabemos cómo usar *fundmatrix.m*

```

if (ind_detectados>1);
    detectados_l = ones(3, ind_detectados-1);
    detectados_r = ones(3, ind_detectados-1);
    detectados_l(1:2,:) = detectados(1:2, 1:ind_detectados-1);
    detectados_r(1:2,:) = detectados(3:4, 1:ind_detectados-1);
end;

```

Mostramos los resultados en cada imagen, pintado en cada punto de correspondencia una cruz y su índice:

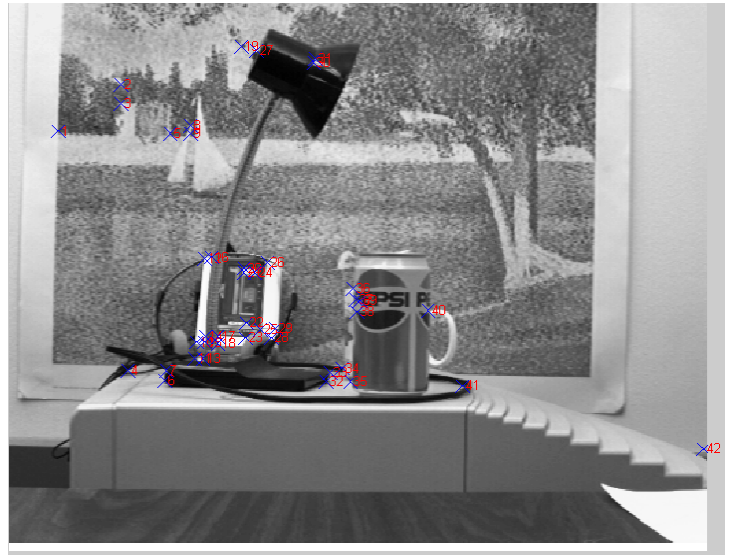
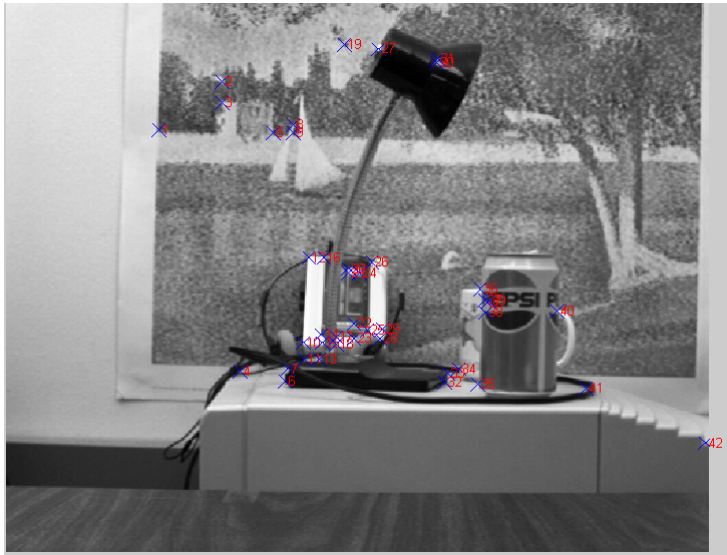
```

figure; imshow(iml./255);
for ind=1:ind_detectados-1;
    rl = detectados(1, ind); cl = detectados(2, ind);
    line([cl+5 cl-5], [rl+5 rl-5]); line([cl-5 cl+5], [rl+5 rl-5]);
    text(cl, rl,sprintf(' %d',ind), 'Color', 'red');
end;

figure; imshow(imr./255);
for ind=1:ind_detectados-1;
    rr = detectados(3, ind); cr = detectados(4, ind);
    line([cr+5 cr-5], [rr+5 rr-5]); line([cr-5 cr+5], [rr+5 rr-5]);
    text(cr, rr,sprintf(' %d',ind), 'Color', 'red');
end;

```

Veamos las salidas:



8.3 APARTADO 3

En lo que se refiere a optimizaciones para evitar procesos de comparación exhaustivos, ya se han realizado en los dos apartados anteriores (en el primero vía el argumento `pixels_alto_busqueda`, y en el segundo con `pixels_alto_busqueda` y `pixels_ancho_busqueda`), limitando la ventana en la imagen derecha dentro de la cual se buscan correspondencias (ya sea sólo de puntos de Harris en el primer apartado, o de cualquier píxel en el segundo) con puntos de Harris de la imagen izquierda.

En lo que se refiere a la medición, usamos el siguiente script, *prac8_3*:

```
inicial1 = cputime;
[dleft_1 dright_1] = prac8_1(8000, 0.8, 5);
tiempo1 = cputime - inicial1;
disp(sprintf(...
'Puntos detectados por prac8_1: %d. Tiempo de computo: %g', size(dleft_1, 2), tiempo1));

inicial2 = cputime;
[dleft_2 dright_2] = prac8_2(8000, 0.8, 5, 150);
tiempo2 = cputime - inicial2;
disp(sprintf(...
'Puntos detectados por prac8_2: %d. Tiempo de computo: %g', size(dleft_2, 2), tiempo2));
```

El resultado es:

```
Puntos de Harris: L=47 R=100
Concordancias detectadas: 32
Puntos detectados por prac8_1: 32. Tiempo de computo: 1.35938
Puntos de Harris en la imagen izquierda: 47
Concordancias detectadas: 42
Puntos detectados por prac8_2: 42. Tiempo de computo: 93.9531
```

O sea, en ambos casos se detectaron 47 puntos de Harris en la imagen izquierda (como no puede ser de otra manera, ya que ambos procesos de extracción son idénticos). De esos 47:

- en el primer caso se detectaron pares de correspondencia con 32 de los 100 puntos de Harris de la imagen derecha, con una eficacia de $32/47 * 100 = 68.0851\%$.
- en el segundo caso se detectaron pares de correspondencia con 42 de los píxeles de la imagen derecha. con una eficacia de $42/47 * 100 = 89.3617\%$

En el primer caso, todo el proceso duró menos de dos segundos, mientras que en el segundo caso, casi 94, una relación de $1.35938/93.9531 * 100 = 1.44687\%$

La principal diferencia operativa entre ambos métodos es la exhaustividad en la comparaciones dentro de la imagen derecha. Como vemos, la eficacia del segundo método es sustancialmente mayor que la del primero, pero a costa de un esfuerzo de cómputo desorbitadamente mayor (al menos con esta implementación).