

# Memoria Práctica 3. Visión Artificial

---

Antonio Delgado Bejarano

## 1. Implementación de funciones.

Para el desarrollo de la práctica se han creado una serie de funciones reutilizables en análisis posteriores con el fin de estructurar un código más general.

### a) FiltroMediana.m

```
function fMedian=FiltroMediana(f,radio)

[M,N]=size(f);

% Creamos la mascara segun el radio pasado como parametro:
NpixelsMask=(2*radio+1)^2;

fMedian=zeros(size(f));

% Aplicamos el filtro de mediana:
for i=radio+1:N-radio
    for j=radio+1:M-radio
        fMedian(i,j)=median(reshape(f(i-radio:i+radio,j-radio:j+radio),...
            NpixelsMask,1));
    end
end

% Convertimos el valor a uint8 para devolverlo:
fMedian=uint8(fMedian);
```

Esta función recibe la imagen y el radio para crear la máscara, devolviendo la imagen ya filtrada. Primero crea la máscara en función del radio y luego recorriendo la imagen con un doble bucle for crea la imagen filtrada. Antes de devolverla la convierte a entero de 8 bits.

### b) ExpansionContraste.m

El objetivo de esta función es la expansión del rango significativo del histograma de la imagen, es decir, transformar un nivel significativo mínimo de gris en el nivel 0, y un nivel máximo significativo en 255. De esta manera, habrá más píxeles repartidos entre los valores 0 y 255, aumentando el contraste.

El criterio para conocer dichos valores significativos viene dado por un *porcentajeResidual* indicado por el usuario de la función en tanto por ciento. Tras diversas pruebas, el porcentaje usado para mejorar el contraste significativamente sin perder información es de un 8%. A continuación, se expone el proceso seguido en la función.

```

18
19 - [M,N]=size(f); Npixels=M*N;
20
21 - %Obtenemos el histograma de la imagen
22 - [counts, r]=imhist(f);
23
24 - %Variables auxiliares para comprobar el porcentaje y tomar el nivel de
25 - %intensidad
26 - integral=0; idx=0;
27
28 - %Se obtiene el nivel de intensidad minimo a tener en cuenta
29 - while(integral<=porcentajeResidual*Npixels/100)
30 -     idx=idx+1;
31 -     integral=cumsum(counts(1:idx));
32 - end
33
34 - %Valor minimo hasta el cual los nuevos valores de intensidad seran 0
35 - rmin=r(idx);

```

En primer lugar, se obtiene de la imagen *f* el número de píxeles para aplicarle el porcentaje indicado y se calcula el histograma haciendo uso de la función *imhist*, tomando como parámetros, para la transformación posterior, las variables *counts* (número de píxeles con un nivel determinado) y *r* (niveles de la imagen). Se usa un bucle *while* que suma el número de

```

37 -     integral=0; idx=length(counts);
38
39 -     %Se obtiene el nivel de intensidad maximo a tener en cuenta
40 -     while(integral<=(porcentajeResidual*Npixels/100))
41 -         integral=cumsum(counts(idx:end));
42 -         idx=idx-1;
43 -     end
44
45 -     %Valor maximo a partir del cual los nuevos valores de intensidad seran 255
46 -     rmax=r(idx);

```

píxeles de cada nivel hasta alcanzar el porcentaje y se asigna el valor de *rmin* al último nivel de gris sumado según la posición *idx*.

En el caso del nivel *rmax*, es decir, el nivel máximo a partir del cual se realiza la transformación, se procede de igual modo. La diferencia radica en que esta vez se recorre el vector *counts* desde el final, hasta que la alcance el valor del porcentaje residual.

```

48 - %Se hace uso de la funcion contrasStretching para obtener la lut de
49 - %transformaci?n lineal a trozos
50 - lut=ContrastStretching(r,rmin,0,rmax,255);
51
52 - %Aumentamos el contraste haciendo uso de la lut
53 - fMayorContraste=lut(f+1);
54
55 - end

```

Finalmente, para la expansión del histograma se recurre a la función *ContrastStretching*, que se expone en el siguiente apartado. Dicha función devuelve una lut (look-up table) para

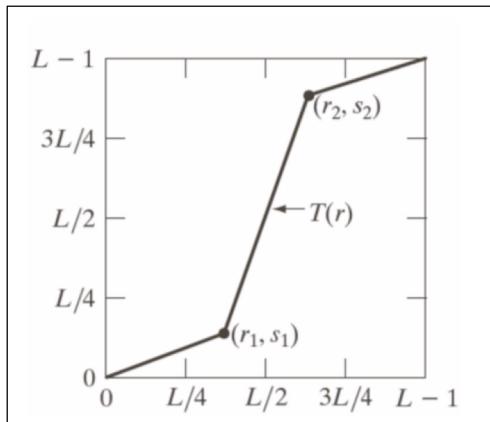
agilizar a nivel computacional la transformación y aumentar el contraste. La llamada a la función parte del rango dinámico original, los valores significativos a transformar en 0 y 255 respectivamente.

Para cada píxel de la imagen se toma el valor nuevo como el contenido de la posición, valor a cambiar, en el vector con la expresión  $lut(f+1)$ .

### c) ContrastStretching.m

Esta función se encarga de realizar una tabla que implemente la transformación lineal a trozos del histograma de una imagen. Los pasos seguidos son sencillos.

A partir de un rango dado  $r$ , se definen tres tramos según los valores inicial y final del tramo. Según la gráfica:



En el caso que nos concierne, la llamada realizada a la función implica un primer tramo con pendiente 0 ( $s_1=0$ ). Y un último tramo con pendiente 0 también ( $s_2=255$ ). El tramo del medio es una transformación lineal con pendiente

$$m = \frac{s_2 - s_1}{r_2 - r_1}$$

Que en la llamada para el análisis de la imagen de la práctica resulta:  $m = \frac{255}{r_{max} - r_{min}}$

A continuación, se presenta la implementación en Matlab de la transformación según la gráfica.

```

26 %Primer tramo. Por debajo de r1
27 if r<=r1
28     s0=0;r0=0;m=(s1-s0)/(r1-r0);
29 %Tercer tramo. Por encima de r2
30 elseif r>=r2
31     s0=s2;r0=r2;m=(255-s0)/(255-r0);
32 %Segundo tramo. Entre r1 y r2
33 else
34     s0=s1;r0=r1;m=(s2-s0)/(r2-r1);
35 end
36
37 %Ecuaci?n punto-pendiente de la transformaci?n
38 s=uint8(s0+m*(r-r0));
39
40 end

```

#### d) CalculaUmbralOtsu.m

Esta función se ha diseñado para aplicar el método de Otsu a la umbralización del histograma. Siguiendo el algoritmo definido, que parte del fundamento que define dos funciones de probabilidad  $p(n)$  escaladas para el objeto a estudiar y el fondo, buscamos el punto  $U=n$  en el que se igualan. Para ello, haremos uso de la varianza de la función  $p$ .

```
15 - %Obtenemos la funcion de probabilidad del histograma
16 - p = double(h)/sum(h);
17 -
18 -
19 - %Variable auxiliar que almacena la varianza de la funcion p segun el nivel
20 - %de gris que separa las dos funciones del metodo de Otsu.
21 - tabVarIntraclase = zeros(256,1);
22 -
```

En primer lugar, se crea una variable que almacenará la varianza de la función  $p(n)$  para cada dos funciones de probabilidad de la iteración. Se calcula también la función de probabilidad del histograma atendiendo a la definición de función de probabilidad:

$$\sum_{i=0}^n p(n) = 1$$

El grueso de la función consta de un bucle *for* que, para cada nivel de gris, separa en dos el histograma y obtiene la media y la varianza de las dos funciones resultantes.

```
25 - for T = 0:255
26 -     indT = T+1;
27 -     %Se calculan las constantes que escalan las dos funciones desde las que
28 -     %se parte en cada iteracion
29 -     A1 = sum(p(1:indT));
30 -     A2 = sum(p(indT+1:256));
31 -
32 -     %Se calculan la medias de ambas funciones de probabilidad
33 -     mu1 = 0;
34 -     mu2 = 0;
35 -     for r=0:T
36 -         mu1 = mu1 + r*p(r+1);
37 -     end
38 -     for r=T+1:255
39 -         mu2 = mu2 + r*p(r+1);
40 -     end
41 -
42 -     mu1 = mu1/A1;
43 -     mu2 = mu2/A2;
44 -
```

La obtención de la media consiste en aplicar la expresión general en discreto:

$$m = \sum_{i=0}^N n * p(n)$$

Como segunda parte, se obtienen las varianzas de las funciones aplicando la expresión general de la varianza y puesto que se tratan de funciones independientes, la suma de ambas, escalada, coincide con la varianza de la función histograma. Sencillamente las escalamos por A1 y A2 para obtener la final.

$$\text{var} = \sum_{i=0}^N (n - m)^2 * p(n) ; \quad \text{var}(p) = \text{var}(f1) + \text{var}(f2)$$

Finalmente, se almacenan todas las varianzas y se toma la de menor valor, será aquella en la que se verifique la condición del método:

$$K_1 * p_{\text{objeto}}(n) = K_2 * p_{\text{fondo}}(n)$$

```

44 %Se calculan la varianzas de ambas funciones de probabilidad
45 var1 = 0;
46 var2 = 0;
47 for r=0:T
48     var1 = var1 + (r-mu1)^2*p(r+1);
49 end
50 for r=T+1:255
51     var2 = var2 + (r-mu2)^2*p(r+1);
52 end
53 var1 = var1/A1;
54 var2 = var2/A2;
55
56 %Tabla de varianza de la funcion p para cada par de funciones segun el
57 %nivel de gris
58 tabVarIntraclase(indT) = A1*var1+A2*var2
59
60 end
61
62 %Obtenemos la posición del minimo de la varianza, nivel en el que las
63 %funciones coinciden
64 [varMin,indMin] = min(tabVarIntraclase)
65
66 %El umbral coincidirá con la posición (1:256) de dicho mínimo transformado
67 %a nivel de gris(0:255)
68 umbral = indMin-1;
69

```

### e) apertura.m

```
function fOpen=apertura(f,mask)
    % Numero de filas de la imagen:
    M = size(f,1);

    % Aplicamos un erosionado y al resultado un dilatado:
    fOpen = images.internal.morphop(f,mask,'erode','notpacked',M);
    fOpen = images.internal.morphop(fOpen,mask,'dilate','notpacked',M);

end
```

Recibimos la imagen y la máscara a aplicar. Para conseguir el efecto de apertura primero realizamos un erosionado a la imagen con la máscara proporcionada, y a continuación realizamos un dilatado a la imagen resultante del previo erosionado.

### g) cierre.m

```
function fClose = cierre(f,mask)
    % Numero de filas de la imagen
    M = size(f,1);

    % Aplicamos un dilatado y al resultado aplicamos un erosionado:
    fClose = images.internal.morphop(f,mask,'dilate','notpacked',M);
    fClose = images.internal.morphop(fClose,mask,'erode','notpacked',M);

end
```

Misma idea que en la función de apertura, solo que en este caso primero se realiza el dilatado y posteriormente el erosionado al resultado de la dilatación (todo esto con una máscara dada).

### h) calculaPropGeom.m

Esta función pretende calcular las diferentes propiedades geométricas de los objetos que conforman la imagen. En la llamada se le puede exigir qué propiedades se quieren calcular (área, perímetro, circularidad...).

```

%Recorremos todos los objetos:
for o=1:cc.NumObjects

    pixelList= cell2mat(cc.PixelIdxList(o));
    mask=zeros(M,N); mask(pixelList)=1; obj=mask.*I;

    % Definicion de Momentos que se usaran para determinadas propiedades
    m00 = sum(sum(obj));
    m10 = 0; m01 = 0; m11= 0;
    m20 = 0; m02 = 0;
    for x=1:N
        for y=1:M
            m10 = m10 + x * obj(y,x);
            m01 = m01 + y * obj(y,x);
            m11 = m11 + x * y * obj(y,x);
            m20 = m20 + x^2 * obj(y,x);
            m02 = m02 + y^2 * obj(y,x);
        end
    end

```

Primero hacemos un bucle para recorrer todos los objetos de la imagen, y definimos los momentos para las diferentes propiedades:

```

% Bucle para calcular todas las propiedades pedidas en la llamada de la
% funci?n:
for k=1:length(varargin)
|
% Asignaci?n del ?rea:
if (strcmp(varargin{k}, 'Area'))
    props(o).Area = m00;
end

% Asignaci?n del centro:
if (strcmp(varargin{k}, 'Centro'))
    xCM = m10/m00;
    yCM = m01/m00;
    props(o).Centro=[xCM,yCM];
end

% Asignaci?n del per?metro:
if (strcmp(varargin{k}, 'Perimetro'))
    props(o).Perimetro=per(o).Perimeter;
end

% Asignaci?n de la circularidad:
if (strcmp(varargin{k}, 'Circularidad'))
    props(o).Circularidad=1/(4*pi)*(per(o).Perimeter^2)/props(o).Area;
end

```

```

% Asignaci?n de la orientaci?n:
if(strcmp(varargin{k}, 'Orientacion'))
    u11= m11 - (m10*m01/m00);
    u20= m20 - (m10^2/m00);
    u02= m02 - (m01^2/m00);

    %C?lculo del ?ngulo:
    props(o).Orientacion =(1/2)*atan(2*u11/(u20-u02))*180/pi;
end

% Asignaci?n del BoundingBox llamando a una funci?n externa:
if(strcmp(varargin{k}, 'BoundingBox'))
    rect=BoundingBox(obj);
    props(o).BoundingBox=rect;
end

end

```

Por ?ltimo hacemos otro bucle para recorrer todos los par?metros de llamada de la funci?n que representan las diferentes propiedades a calcular, realizando el c?lculo de cada una de ellas y asign?ndolo en la estructura de datos que se va a crear para almacenar los datos. Una vez calculados, la informaci?n se guarda en un struct de este tipo:

Fields	Area	Centro	Orientacion	Circularidad	Perimetro	Tipo	Tamano
1	113051 [623.207...]	23.5463	1.0074	1.1963e+03	'Moneda'	3	
2	84857 [626.342...]	16.1567	0.9998	1.0325e+03	'Moneda'	2	
3	161362 [1.0632e...]	-3.1963	2.1473	2.0867e+03	'Otro'	0	
4	45118 [1.1197e...]	-35.2285	1.0012	753.4420	'Moneda'	1	
5	47715 [1.1878e...]	-35.1796	0.9957	772.6700	'Moneda'	1	
6	76193 [1.4752e...]	-27.4267	0.9972	977.1180	'Moneda'	2	
7	49874 [1.4678e...]	-13.0931	1.0089	795.1920	'Moneda'	1	
8	80135 [1.9155e...]	-17.8925	1.0152	1.0111e+03	'Moneda'	2	
9	73740 [2.1583e...]	-31.2227	0.9962	960.7760	'Moneda'	2	
10	66064 [2.5465e...]	-19.8880	0.9978	910.1460	'Moneda'	1	
11	133626 [2.7819e...]	-36.0225	2.0484	1.8546e+03	'Otro'	0	

### i) Etiquetado.m

Esta funci?n pretende añadir una peque?a descripci?n sobre cada objeto en la imagen. En el caso de las monedas se añadir? un peque?o texto indicando su tama?o, mientras que para los dem?as objetos se representar? una recta en su eje mayor. Para ello recorremos con un bucle todos los objetos, y seg?n el tama?o (1, 2 o 3 para las monedas) calculado anteriormente se representar? el mensaje que corresponda y en el color que corresponda (amarillo para las peque?as, magenta para las medianas y rojo para las grandes).

```

function [fEtiq,nEtiqs] = Etiquetado(f,nMaxEtiqs)

% Almacenamos en fEtiq la imagen original:
fEtiq = f;

% Variable para ir contando el n?mero de etiquetas:
nEtiqs = 0;

% Bucle para recorrer todos los objetos:
for i=1:length(nMaxEtiqs)

    % Seg?n el tama?o representa el texto necesario con el color
    % correspondiente al tama?o de cada moneda:
if (nMaxEtiqs(i).Tamanio == 1)
    fEtiq = insertText(fEtiq,nMaxEtiqs(i).Centro,'Moneda Peque?a','FontSize',50, ...
    'BoxOpacity',0,'TextColor','y');

    % Cada vez que inserta una etiqueta incrementa el valor de la variable:
nEtiqs = nEtiqs + 1;

elseif (nMaxEtiqs(i).Tamanio == 2)
    fEtiq = insertText(fEtiq,nMaxEtiqs(i).Centro,'Moneda Mediana','FontSize',50, ...
    'BoxOpacity',0,'TextColor','m');
    nEtiqs = nEtiqs + 1;

elseif (nMaxEtiqs(i).Tamanio == 3)
    fEtiq = insertText(fEtiq,nMaxEtiqs(i).Centro,'Moneda Grande','FontSize',50, ...
    'BoxOpacity',0,'TextColor','r');
    nEtiqs = nEtiqs + 1;

```

En el caso de los objetos que no son monedas, se pretendía representar en verde una línea en su eje mayor, pero no hemos conseguido implementar esta funcionalidad. Hemos optado por representar un cartel con la palabra ‘Otro’.

```

else
    fEtiq = insertText(fEtiq,nMaxEtiqs(i).Centro,'Otro','FontSize',50, ...
    'BoxOpacity',0,'TextColor','g');
    nEtiqs = nEtiqs + 1;

```

#### j) indicaTamanio.m

Esta función pretende distinguir tres tipos de monedas según su área: monedas grandes, medianas y pequeñas. Para ello obtendrá del struct el área de la moneda más grande y la de la moneda más pequeña, definiendo así un intervalo dividido en tres partes en el cual los extremos son estos dos valores:

```

% Obtenemos las áreas de todos los objetos que están en el struct:
area=struct(props(ids).Area);

% Obtenemos el área máxima y mínima:
amax = max(area);
amin = min(area);

% Obtenemos el intervalo pedido, dividido en tres partes:
intervalo=(amax-amin)/3;

% Determinamos los valores límites que tendrán las áreas para ser
% consideradas grandes o pequeñas:
a_gran = amax-intervalo;
a_peq = amin + intervalo;

```

Definimos también dos valores (a\_gran y a\_peq) los cuales serán los límites para los cuales un área se considerará máxima o mínima. Ahora recorremos todos los objetos mediante un bucle y comparamos el área de cada objeto con estos dos valores. Si es mayor que a\_gran será una moneda grande, si es menor que a\_peq será una moneda pequeña, y si está en una zona intermedia del intervalo será una moneda mediana.

```

% Bucle para recorrer todos los objetos:
for o=1:length(props)

    % Solo nos interesan las monedas:
    if(strcmp(props(o).Tipo, 'Moneda'))

        % Si el área es menor que a_peq le asignamos tamaño 1 (moneda
        % pequeña):
        if(props(o).Area < a_peq)
            props(o).Tamanio=1;

        % Si el área es mayor que a_gran le asignamos tamaño 3 (moneda
        % grande):
        elseif(props(o).Area > a_gran)
            props(o).Tamanio=3;

        % Si no es ni uno ni otro tenemos una moneda mediana:
        else
            props(o).Tamanio=2;
    end
end

```

## k) indicaTipo.m

Esta función discrimina qué objetos son monedas y qué objetos no. Para ello recorrerá el struct que contiene la información de las propiedades de todos los objetos, y si la circularidad de un objeto es aproximadamente igual a uno lo catalogará como tipo “Moneda”, mientras que si la circularidad no cumple esta condición se catalogará como tipo “Otro”. Además se cuentan cuantas monedas se han encontrado mediante la instrucción find:

```

function [props,id_monedas,id_otro] = indicaTipo(props)

id=zeros(1,length(props));

% Recorremos todos los objetos:
for o=1:length(props)

    % Si la circularidad es aproximadamente 1 asignamos ese objeto como
    % moneda, si no lo asignamos como otro:
    if(round(props(o).Circularidad) == 1)
        props(o).Tipo='Moneda';

        % A los objetos Moneda le asignamos un identificador igual a 1:
        id(o)=1;
    else
        props(o).Tipo='Otro';
        % A los objetos Otro le asignamos un identificador igual a 0:
        id(o)=0;
    end
end

% Contamos cuantas monedas hemos detectado:
id_monedas=find(id ~= 0);
id_otro=find(id~=1);
end

```

## 2.- Programa principal:

El programa empieza leyendo la imagen a tratar y representándola tal cual. Posteriormente se le realiza un filtrado de mediana y se vuelve a representar. Por último, se realiza una expansión de contraste mediante el análisis del histograma de la imagen y se vuelve a representar:

```

%% 1. Lectura y filtrado

%Lectura y conversi?n a escala de grises:
I=im2double(imread('monedasLlaves.jpg'));I=rgb2gray(I);

figure(1), title(strcat('Imagen monedasLlaves')), imshow(I), shg

%Filtro de mediana:
I_Mediana=uint8(255*medfilt2(I,[5 5]));
%Imediana=FiltroMediana(I, radioMask);
figure(2), title(strcat('Imagen monedasLlaves filtrada')),...
imshow(I_Mediana), shg

input('Pulse cualquier tecla para continuar')

%Expansi?n de contraste mediante an?lisis de histograma:
I_MayorContraste=ExpansionContraste(I_Mediana,8);

figure(3), title(strcat('Imagen monedasLlaves MayorContraste')),...
imshow(I_MayorContraste), shg

```

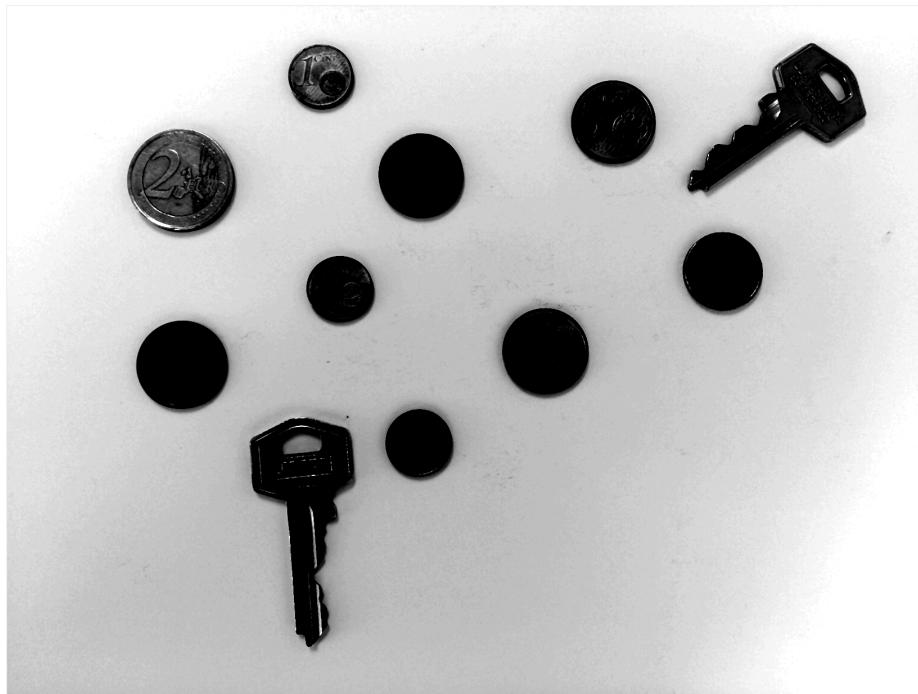
Imagen original:



Imagen tras filtrado de mediana:



Imagen tras expansión de contraste:



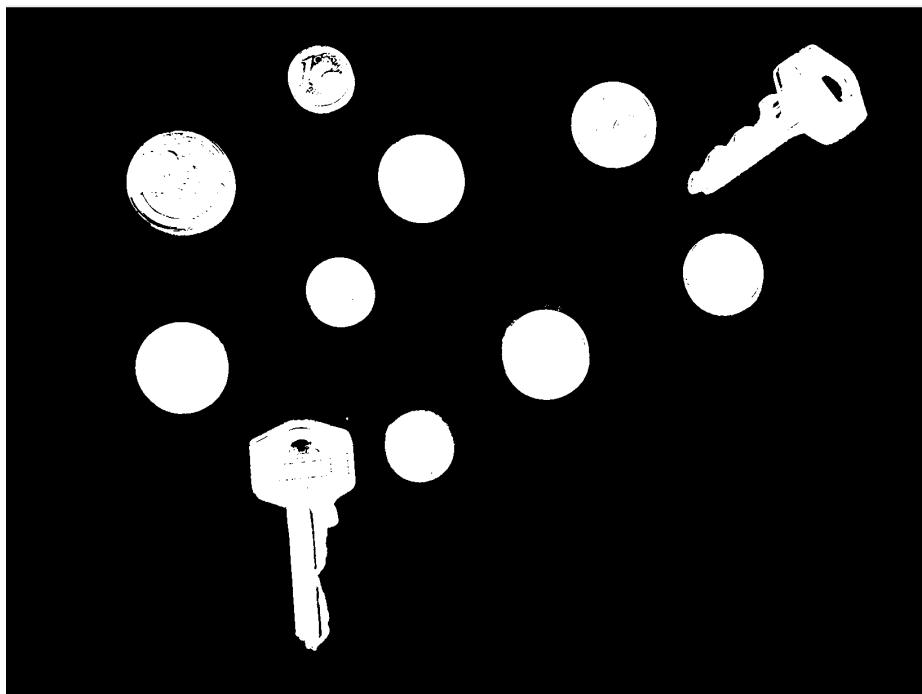
La segunda sección corresponde con la binarización de la imagen, para lo cual vamos a obtener su histograma y le vamos a calcular su umbral por el método de Otsu. Tras esto binarizamos la imagen de forma que los valores que queden por debajo del umbral se considerarán negro y los que queden por encima se considerarán blanco:

```
%% Binarización de la imagen
%Obtenemos el histograma:
[counts, ~]=imhist(I_MayorContraste);

%Calculamos el umbral de Otsu:
umbral= CalculaUmbralOtsu(counts);

%Binarización de la imagen y representación de la imagen binarizada:
I_binarize=(I_MayorContraste<=umbral); figure(4), imshow(I_binarize)
```

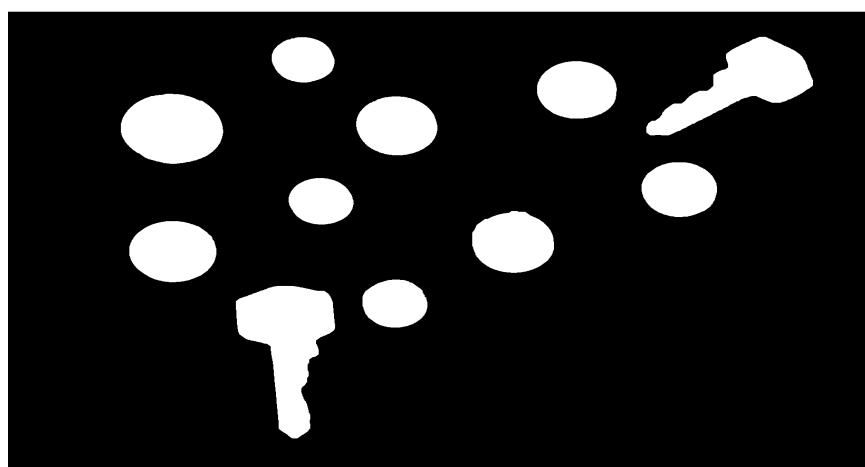
Imagen binarizada:



Ahora vamos a mejorar la imagen de siluetas anterior. Primero vamos a aplicarle una máscara y una función de cierre y otra de apertura. Tras esto, le calculamos los parámetros que nos interesan de cada objeto (área, centro, perímetro, BoundingBox, circularidad, orientación, excentricidad...). Además, hemos de identificar si cada objeto es una moneda o no, y en caso de serlo, hemos de determinar su tamaño. Para todo esto nos ayudaremos de las funciones anteriores:

Primero aplicamos los métodos de apertura y cierre:

Imagen mejorada tras apertura y cierre:



Ahora obtenemos los parámetros deseados (el BoundingBox lo obtenemos directamente desde regionprops):

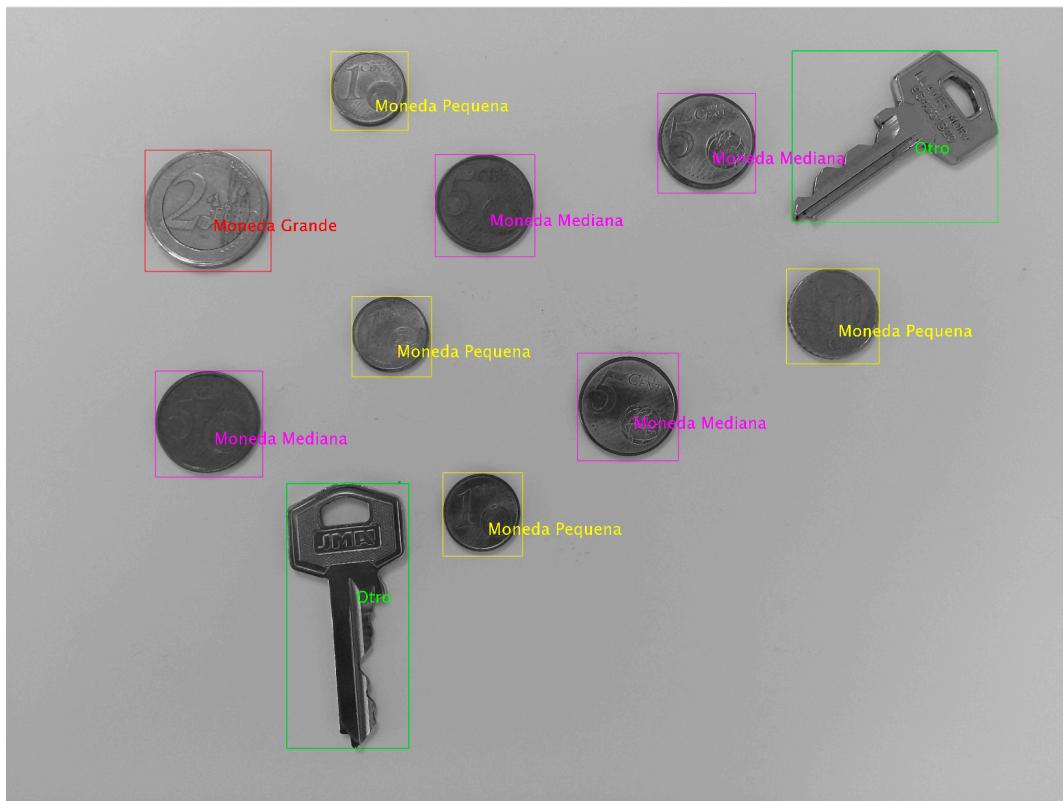
```
%Obtencion del BoundingBox con regionprops:  
stats=regionprops(IOpened, 'BoundingBox');  
  
%Obtencion de los parametros de interes con calculaPropGeom:  
[props,obj]=calculaPropGeom(IOpened,'Area','Centro','Orientacion', 'Circularidad',...  
'Perimetro');
```

```
%Identificamos el tipo de cada objeto (moneda u otro):  
[props,idMonedas] = indicaTipo(props);  
  
%Identificamos el tamano de cada moneda (grande, mediana o pequena):  
props = indicaTamanio(props,idMonedas);  
  
%Obtenemos el eje mayor de los objetos para representarlo  
Eje_Mayor=regionprops('table',obj,'MajorAxisLength');
```

Por último, gracias a la función etiquetado obtenemos la imagen con los títulos correspondientes, y ahora representamos los rectángulos pedidos en función del tamaño (color amarillo para las monedas pequeñas, magenta para las medianas y rojo para las grandes). Para los objetos que no son monedas tenemos un rectángulo verde:

```
%% Representacion  
  
%Funcion de etiquetado para añadir los nombres o lneas a cada objeto:  
title = Etiquetado(I,props);  
  
%Representacion del rectngulo (BoundingBox) de cada objeto:  
imshow(title),hold on  
for i=1:length(stats)  
  
% Segn el tamano de cada moneda le asigna un color al rectngulo. Para los  
% objetos que no son monedas se aplica el color verde:  
if (props(i).Tamanio == 1)  
    rectangle('Position',stats(i).BoundingBox,'EdgeColor','y');  
  
elseif (props(i).Tamanio == 2)  
    rectangle('Position',stats(i).BoundingBox,'EdgeColor','m');  
  
elseif (props(i).Tamanio == 3)  
    rectangle('Position',stats(i).BoundingBox,'EdgeColor','r');  
  
else  
    rectangle('Position',stats(i).BoundingBox,'EdgeColor','g');  
  
end  
  
end  
  
hold off
```

Esta es la imagen obtenida tras todo el proceso:



Por último se nos pide demostrar que funciona el procedimiento al rotar la imagen:

