Sistemas de Percepción

Práctica 2. Segmentación por color

Trabajar con espacios de color, ayudándonos de ellos para la segmentación de los diferentes objetos presentes en una imagen

Nombre: Sergio León Doncel y Álvaro García Lora

Curso: 2022/ 2023

Titulación: 4º Curso, GIERM

MEMORIA. PRÁCTICA 2

INTRODUCCIÓN

El objetivo consiste en desarrollar en Matlab un algoritmo de procesamiento de imágenes capaz de realizar la segmentación de los objetos presentes en ella y analizar sus características, representando el centro de éste y los marcos que encajonan sus dimensiones (bounding boxs).

VERSIÓN BÁSICA

En primer lugar, se realiza la versión básica en donde se recurren a las funciones de alto nivel de Matlab de la toolbox Image Processsing:

-strel(): Crea la estructura morfológica necesaria para generar una plantilla con la forma y el tamaño especificados.

-imerode(): Realiza la erosión (método morfológico fundamental que reduce el tamaño de los objetos) a través de la imagen y plantilla que le son proporcionados.

-imdilate(): Realiza la dilatación (método morfológico fundamental que aumenta el tamaño de los objetos) a través de la imagen y plantilla que le son proporcionados.

-imopen(): Aplica erosión y posteriormente dilatación. De esta forma el tamaño de los objetos en la imagen se conserva (se recupera en el segundo paso) pero se consigue abrir los “poros” (píxeles dispersos en la imagen).

-imclose(): Aplica dilatación y posteriormente erosión. De esta forma el tamaño de los objetos en la imagen se conserva (se recupera en el segundo paso) pero se consigue cerrar los “poros” (píxeles dispersos en la imagen).

-bwlabel(): Devuelve la imagen etiquetando los objetos (le asigna a cada píxel un valor asociado a cada uno de los objetos encontrados).

-regionprops(): Analiza las características de los distintos objetos que detecta, en función de los parámetros que se le pidan (centroide, área, boundingBox…)

Se puede abordar dicho algoritmo recurriendo a la herramienta Color Thresholder para encontrar los valores de umbralización apropiados para separar por colores los objetos. En ella, encontramos 2 caminos distintos: utilizar canales RGB o HSV.

*RGB BÁSICO

Inicializamos vectores donde almacenar los distintos umbrales (mínimos y máximos) obtenidos anteriormente como se mencionaba, para cada uno de los colores que se pueden encontrar en la imagen, con el siguiente orden: Azul, Verde, Amarillo, Rojo, Negros, Naranja.

Además, definimos los valores de radio que se usarán en las plantillas de las aperturas y cierres que se realicen posteriormente en el código, así como el área aproximada de los objetos que queremos detectar (“lacasitos”).

PARÁMETROS

```
% Umbrales maximos y minimos RGB
% (Orden colores: Azul, Verde, Amarillo, Rojo, Negros, Naranja)

channelRedMin = [0,0,166,89,201,0];
channelRedMax = [85,133,255,215,255,99];
channelGreenMin = [51,119,154,0,0,0];
channelGreenMax = [207,255,255,88,198,112];
channelBlueMin = [105,0,0,12,0,0];
channelBlueMax = [252,177,72,255,101,109];

% Radios para aplicar erosionado y dilatado
radio_open = 2;
radio_close = 5;

% Área para eliminación de objetos no lacasitos
areaLacasito = 800;
```

DIVIDIR CANALES DE COLORES

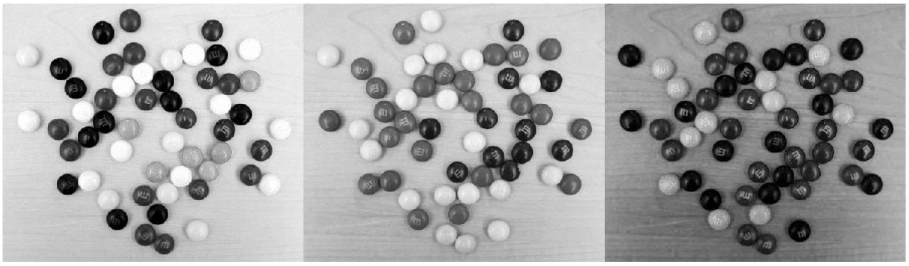
Cargamos la imagen original con imread(), guardamos su tamaño como matriz en M,N,C, siendo estos número de píxeles en una columna, número de píxeles en una fila, y número de canales de la imagen.

Como estamos tratando una imagen RGB, el primer canal se corresponde al color rojo, el segundo al verde, y el tercero al azul.

```
f = imread('imagenDePartida.png');
figure(1); imshow(f);
```

```
[M,N,C] = size(f);

fRojo  = f(:,:,1);
fVerde = f(:,:,2);
```

```
fAzul  = f(:,:,3);

figure(2); imshow([fRojo,fVerde,fAzul]);
```



CLASIFICACIÓN E IDENTIFICACIÓN

A través de la variable color, se recorre un bucle en el cuál se va seleccionando cada uno de los colores de los objetos que se encuentran en la imagen para procesarlos por separado, aunque siguiendo la misma técnica y criterios en todos ellos. Así, el algoritmo obtenido no es el que mejor resultado pueda dar para esta imagen en particular (para ello se podría implementar un vector de tamaños de radio para las máscaras de apertura y cerrado), es más genérico por lo que al cambiar a otra imagen similar el mismo código o con pequeñas adaptaciones puedan servir para lograr el objetivo, lo que aporta versatilidad en su uso.

```
Ncolor=6; %Número de colores en la imagen a clasificar
for color=1:Ncolor
```

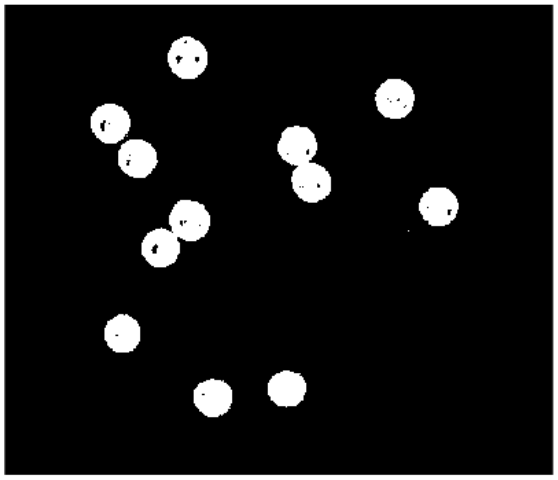
BINARIZACIÓN POR UMBRALES

Se procede a obtener imágenes binarizadas comparando los canales con los umbrales definidos, para posteriormente utilizar éstas como plantillas que contengan los “lacasitos” del color seleccionado en la variable *color*.

Básicamente, consiste en ver si el valor del píxel en todos los canales está dentro de los rangos establecidos. En caso afirmativo, su valor pasa a ser 1, en caso negativo 0.

```
    fRojoPlantilla = (channelRedMin(color)<=fRojo & fRojo<=channelRedMax(color));
    fVerdePlantilla = (channelGreenMin(color)<=fVerde & fVerde<=channelGreenMax(color));
    fAzulPlantilla = (channelBlueMin(color)<=fAzul & fAzul<=channelBlueMax(color));
    fPlantilla = fRojoPlantilla & fVerdePlantilla & fAzulPlantilla;


    figure(3); imshow(fPlantilla);
```

Si se representa gráficamente el resultado obtenido, se puede ver la plantilla obtenida que se utlizará a posteriori para procesar y conseguir resultados.



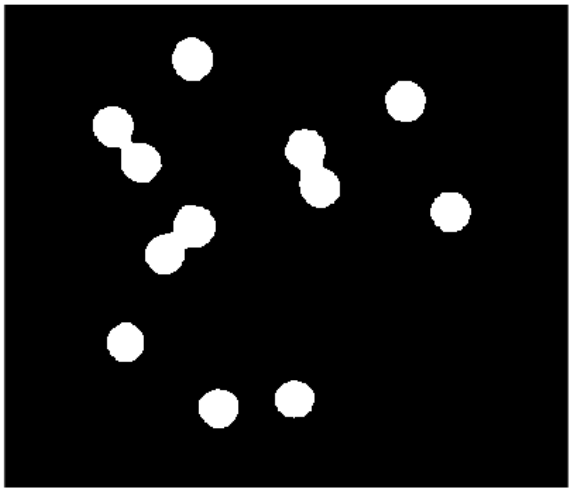
APLICAR DILATACIONES Y EROSIONES

Con la apertura, se consigue eliminar los pequeños píxeles dispersos que no interesa que formen parte de la plantilla ya que no se corresponden a ningún objeto de interés.

Seguidamente, aplicando el cierre se consigue que aquellos agujeros que tenga la plantilla (los cuáles sí pertenecen a los objetos) desaparezcan o se minimicen, de cara a obtener mejores resultados cuando se analice las propiedades del objeto más adelante.

```
se_open = strel('disk',radio_open);
se_close = strel('disk',radio_close);

fPlantillaAfterOpen = imopen(fPlantilla,se_open);
fPlantillaAfterClose = imclose(fPlantillaAfterOpen,se_close);


figure(4); imshow(fPlantillaAfterClose);
```

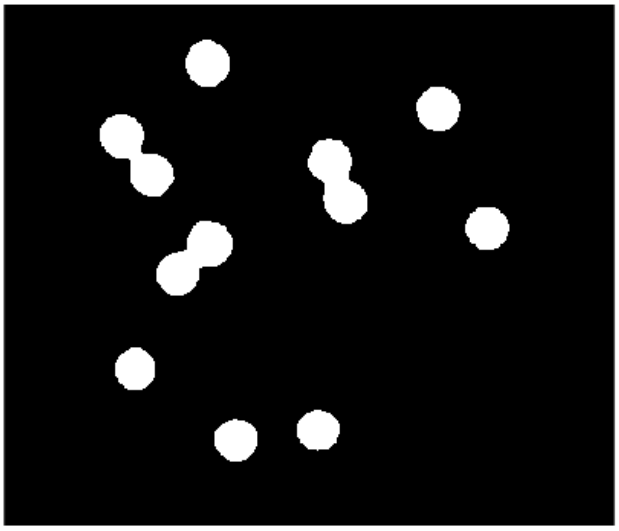
En la siguiente figura se puede ver como el resultado de este procesamiento morfológico mejora la plantilla:



APLICAR FILTRO DE ÁREA

Como en ciertos casos, dado que aplicamos un tamaño de plantilla para la apertura y cierre fijos para todos los colores, la plantilla aún tiene píxeles que no corresponden a los "lacasitos", podemos filtrar estas agrupaciones de píxeles a través de su área. Si es demasiado pequeña, sabemos que no es de interés.

```
fPlantillaEtiq = bwlabel(fPlantillaAfterClose);
fPlantillaLacasitos = fPlantillaAfterClose;

for i=1:max(max(fPlantillaEtiq))
    fPlantillaObjeto = fPlantillaAfterClose & (fPlantillaEtiq == i);
    areaObjeto = regionprops(fPlantillaObjeto,'Area');
    if(areaObjeto.Area < areaLacasito)
        fPlantillaLacasitos = fPlantillaLacasitos & (fPlantillaEtiq ~= i);
    end
end


figure(5); imshow(fPlantillaLacasitos,[]);
```



BOUNDINGBOX Y CENTROS

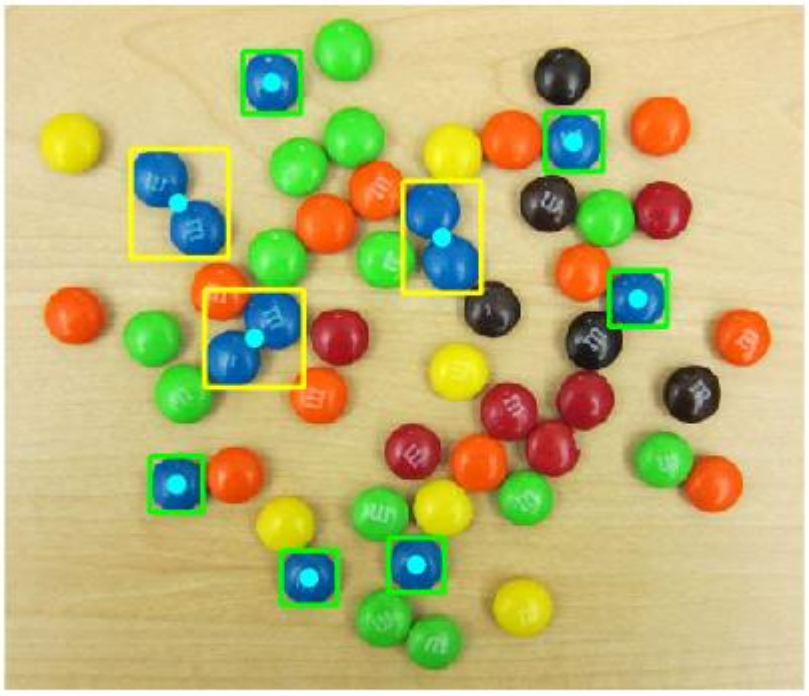
Recogiendo en la estructura *datos* la información devuelta por regionprops, aplicada sobre la plantilla final obtenida, obtenemos las dimensiones y un punto del bounding-box, la posición de los centros y el área.

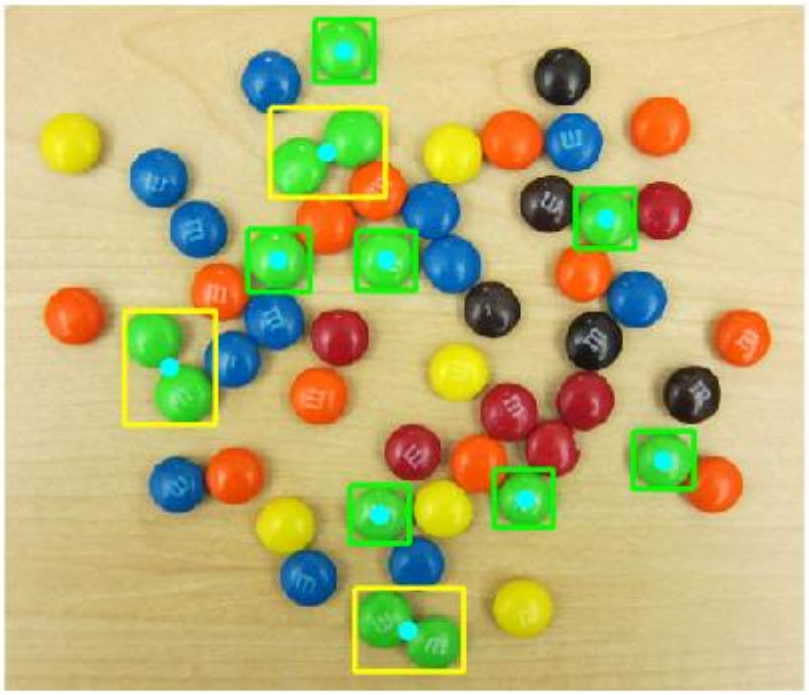
Con ello, podemos representar en el mismo plot la imagen original junto a los marcos de las boundingbox (de color verde en caso de "lacasitos" individuales, o amarillo en caso de no haber logrado separar los que se encuentran solapados, lo cual se distingue a través del área).

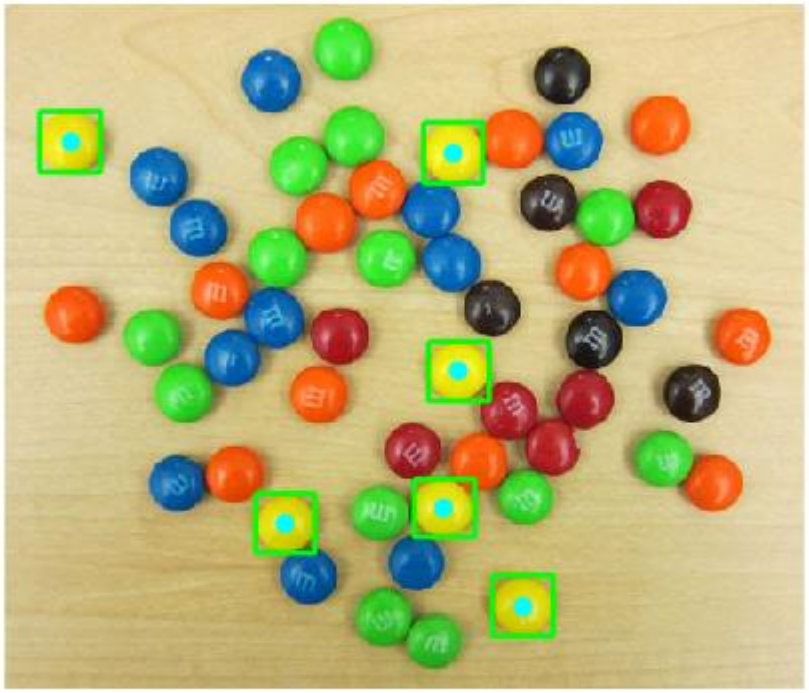
```
datos = regionprops(fPlantillaLacasitos,'BoundingBox','Centroid','Area');
bbox = cat(1,datos.BoundingBox);
sizebbox=size(bbox);
centroids = cat(1,datos.Centroid);
areaDetect = cat(1,datos.Area);
```

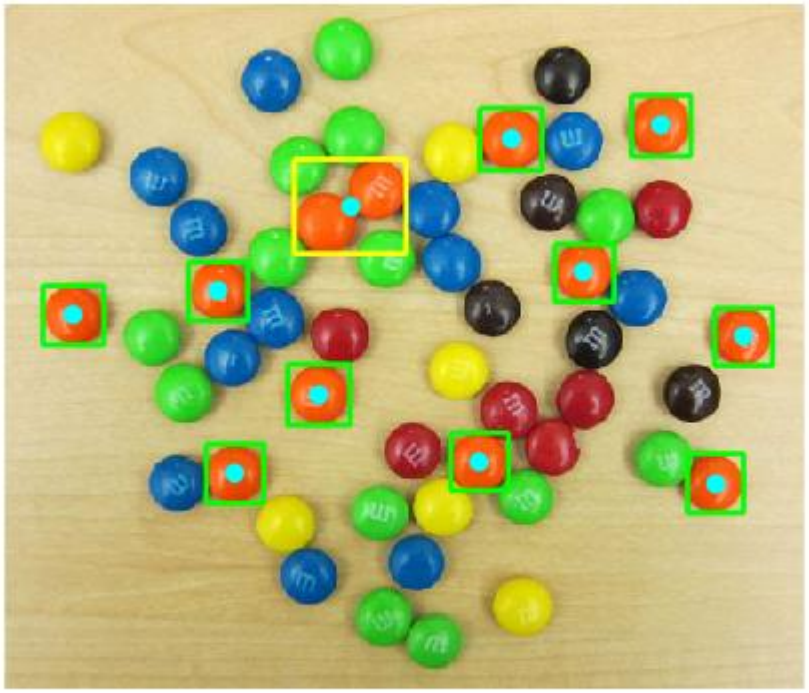
```
    figure();imshow(f);
    hold on
    %Centros
    plot(centroids(:,1),centroids(:,2),'c.','MarkerSize',25);
    %BoundingBox
    for i=1:sizebbox(1)
        pIniBbox = bbox(i,1:2)';
        anchoBbox = bbox(i,3);
        altoBbox = bbox(i,4);
        p1 = pIniBbox;
        p2 = pIniBbox+[anchoBbox,0]';
        p3 = pIniBbox+[anchoBbox,altoBbox]';
        p4 = pIniBbox+[0,altoBbox]';
        if(areaDetect(i) >= (areaLacasito*1.5))
            line ([p1(1),p2(1),p3(1),p4(1),p1(1)],
[p1(2),p2(2),p3(2),p4(2),p1(2)],'LineWidth',2 ,'Color', 'y');
        else
            line ([p1(1),p2(1),p3(1),p4(1),p1(1)],
[p1(2),p2(2),p3(2),p4(2),p1(2)],'LineWidth',2 ,'Color', 'g');
        end
    end
    hold off

end
```

Los resultados obtenidos para cada color son los siguientes:









A modo de conclusión, a través de la umbralización RGB no se ha conseguido separar en un primer paso los "lacasitos" próximos del mismo color. Por ello, como se muestra en el siguiente apartado, a veces es conveniente procesar las imágenes en otro modelo de color como es el HSV.

*HSV BÁSICO

Con el objetivo de mejorar el resultado, se recurre al modelo de color HSV, para umbralizar a través del canal H (hue) para poder distinguir puramente por el matiz de color, aún así filtrando también en los canales de S (saturación) y V (valor) para captar el color puro y el color marrón/negro.

Dado que el código funciona de manera idéntica salvo en la primera parte de umbralización, se resaltarán los cambios importantes de aquí en adelante.

CARACTERÍSTICA DE LA IMAGEN

```
Ncolor=6; %Número de colores en la imagen a clasificar
```

PARAMETROS

Organizamos en una matriz de 3 dimensiones los umbrales, de la siguiente manera:

-1ra componente: Mínimo, máximo.

-2da componente: Capa de HSV.

-3ra componente: Color seleccionado.

```
% Umbrales maximos y minimos HSV
umbralesHSV = zeros(2,3,Ncolor);

umbralesHSV(:,:,1) = [0.146, 0.587, 0.517; 0.251, 1.000, 1.000]; %Color amarillo
umbralesHSV(:,:,2) = [0.433, 0.309, 0.000; 0.800, 1.000, 1.000]; %Color azul
umbralesHSV(:,:,3) = [0.029, 0.566, 0.710; 0.080, 1.000, 1.000]; %Color naranja
umbralesHSV(:,:,4) = [0.000, 0.000, 0.000; 1.000, 1.000, 0.329]; %Color negro
umbralesHSV(:,:,5) = [0.933, 0.178, 0.378; 0.027, 1.000, 0.976]; %Color rojo
umbralesHSV(:,:,6) = [0.194, 0.294, 0.371; 0.467, 1.000, 1.000]; %Color verde

% Radios para aplicar erosionado y dilatado
radio_open = 1;
radio_close = 2;

% Área para eliminación de objetos no lacasitos
areaLacasito = 800;
```

DIVIDIR CANALES DE COLORES

```
h = imread('imagenDePartida.png');
figure(1); imshow(h);
```

```
[M,N,C] = size(h);

f = rgb2hsv(h);
fHue  = f(:,:,1);
fSat = f(:,:,2);
fVal  = f(:,:,3);

figure(2); imshow([fHue,fSat,fVal]);
```

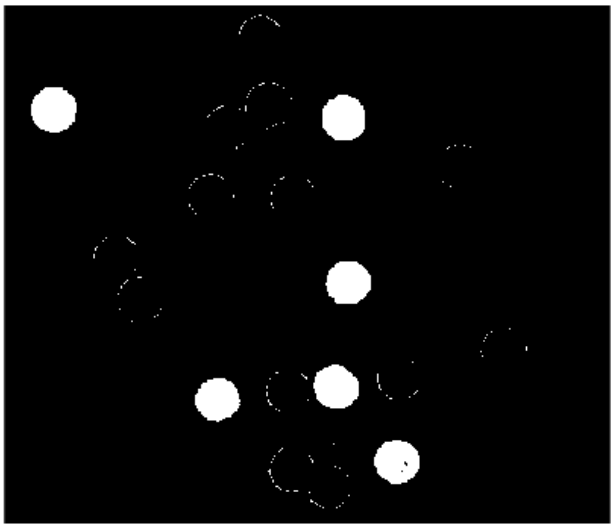
CLASIFICACIÓN E IDENTIFICACIÓN

```
Ncolor=6; %Número de colores en la imagen a clasificar
for color=1:Ncolor
```

BINARIZACIÓN POR UMBRALES

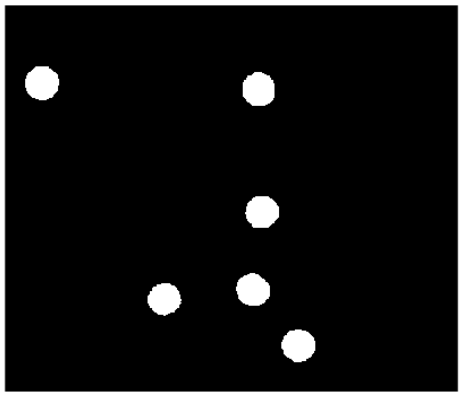
```
    if(umbralesHSV(1,1,color)<umbralesHSV(2,1,color))
        fHuePlantilla = (umbralesHSV(1,1,color)<=fHue & fHue<=umbralesHSV(2,1,color));
    else
        fHuePlantilla = (umbralesHSV(1,1,color)<=fHue | fHue<=umbralesHSV(2,1,color));
    end
    fSatPlantilla = (umbralesHSV(1,2,color)<=fSat & fSat<=umbralesHSV(2,2,color));
    fValPlantilla = (umbralesHSV(1,3,color)<=fVal & fVal<=umbralesHSV(2,3,color));
    fPlantilla = fHuePlantilla & fSatPlantilla & fValPlantilla;

    figure(3); imshow(fPlantilla);
```

Como se puede ver, la plantilla obtenida de primeras con sólo la umbralización es mucho mejor el resultado obtenido:



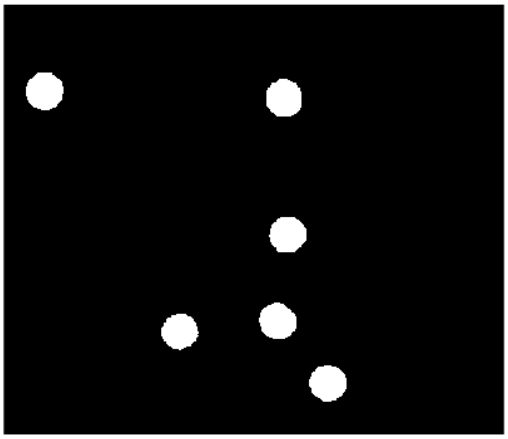
APLICAR DILATACIONES Y EROSIONES

```
se_open = strel('disk',radio_open);
se_close = strel('disk',radio_close);

fPlantillaAfterOpen = imopen(fPlantilla,se_open);
fPlantillaAfterClose = imclose(fPlantillaAfterOpen,se_close);

figure(4); imshow(fPlantillaAfterClose);
```



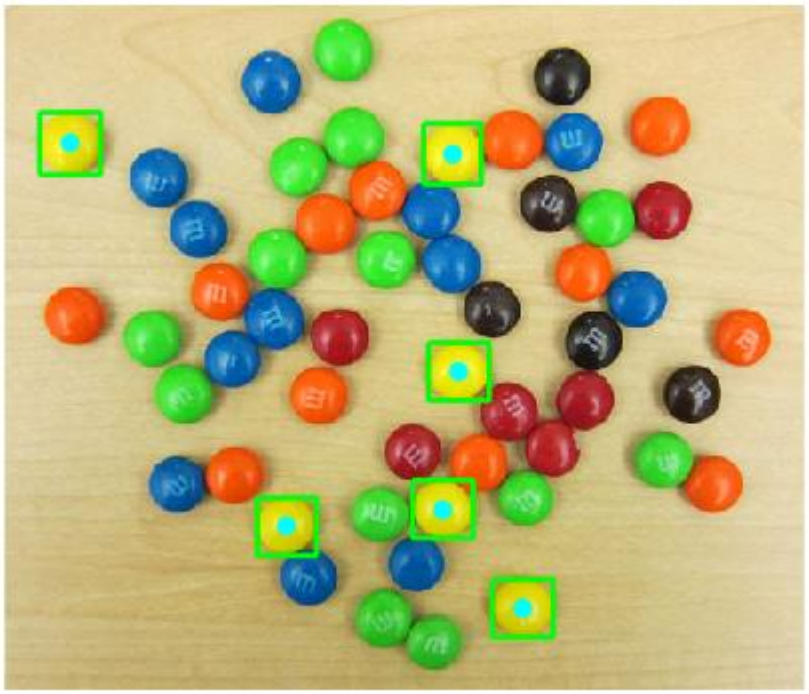
APLICAR FILTRO DE ÁREA

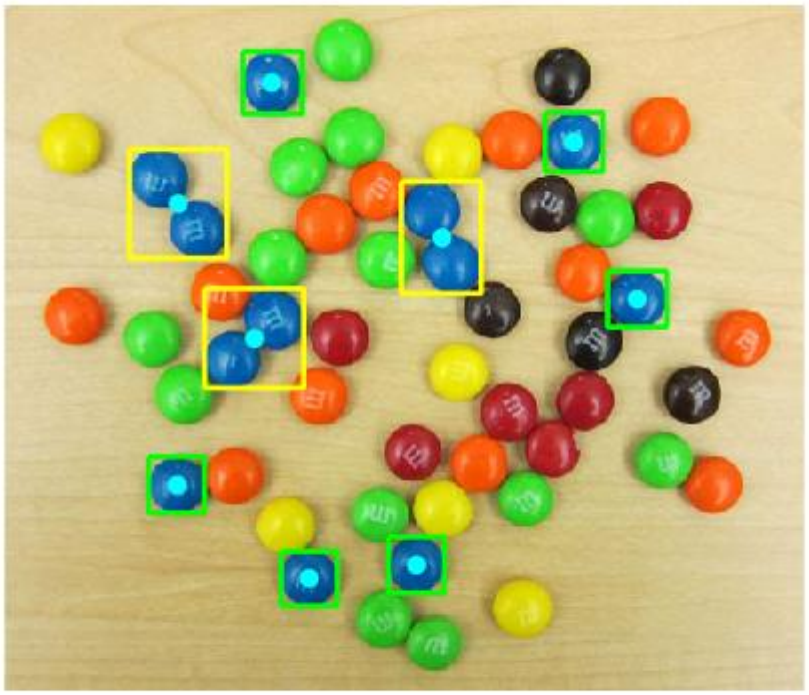
```
fPlantillaEtiq = bwlabel(fPlantillaAfterClose);
fPlantillaLacasitos = fPlantillaAfterClose;

for i=1:max(max(fPlantillaEtiq))
    fPlantillaObjeto = fPlantillaAfterClose & (fPlantillaEtiq == i);
    areaObjeto = regionprops(fPlantillaObjeto,'Area');
    if(areaObjeto.Area < areaLacasito)
        fPlantillaLacasitos = fPlantillaLacasitos & (fPlantillaEtiq ~= i);
    end
end

figure(5); imshow(fPlantillaLacasitos,[]);
```

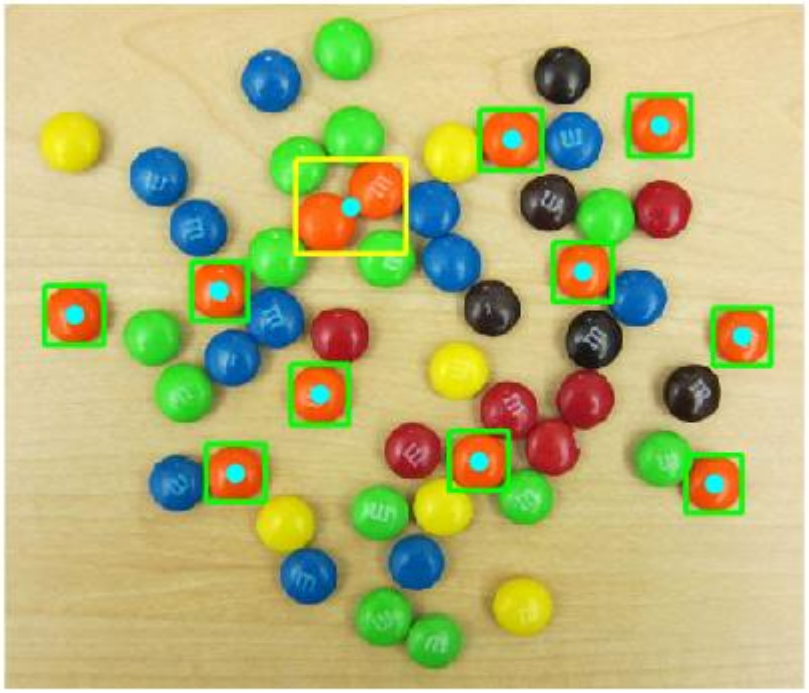


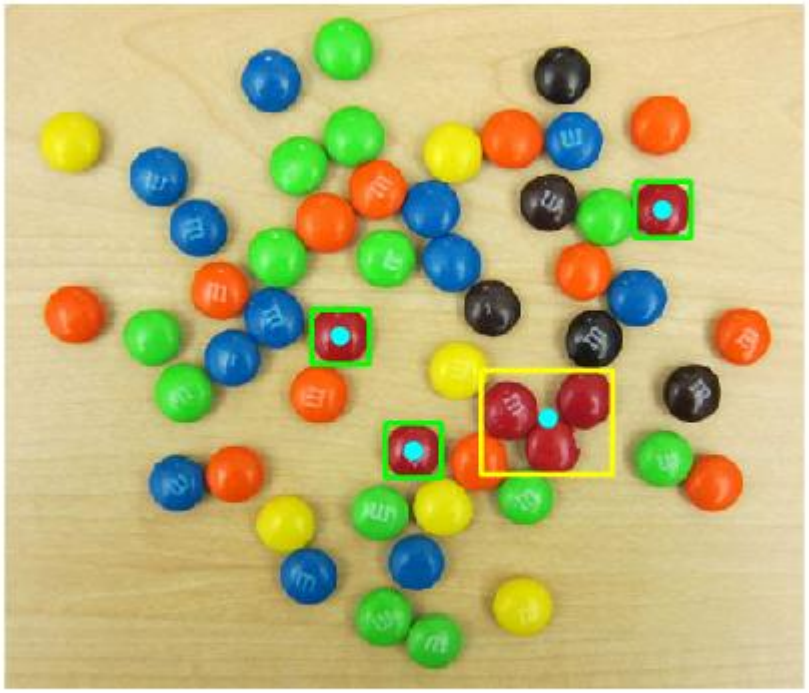
BOUNDINGBOX Y CENTROS

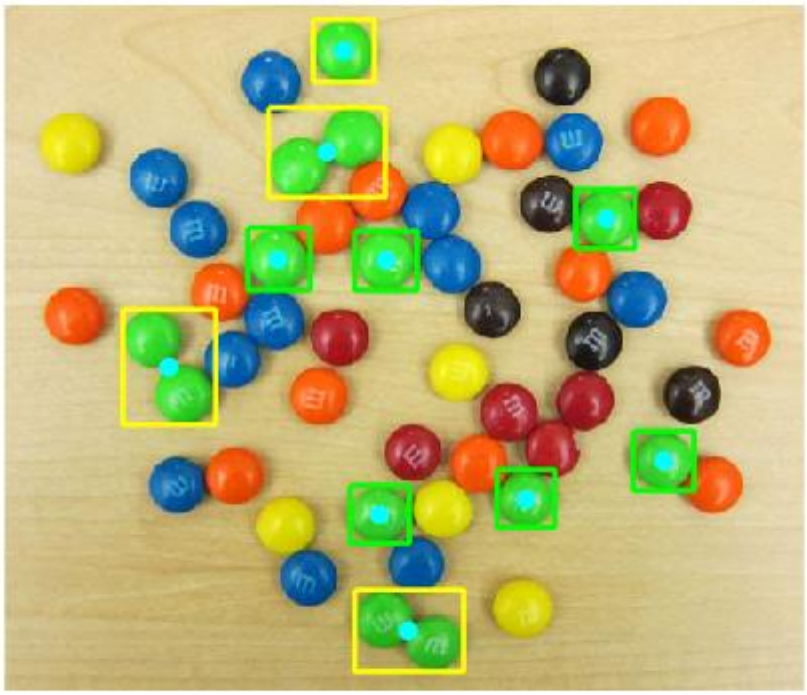
```
    datos = regionprops(fPlantillaLacasitos,'BoundingBox','Centroid','Area');
    bbox = cat(1,datos.BoundingBox);
    sizebbox=size(bbox);
    centroids = cat(1,datos.Centroid);
    areaDetect = cat(1,datos.Area);

    figure();imshow(h);
    hold on
    %Centros
    plot(centroids(:,1),centroids(:,2),'c.','MarkerSize',25);
    %BoundingBox
    for i=1:sizebbox(1)
        pIniBbox = bbox(i,1:2)';
        anchoBbox = bbox(i,3);
        altoBbox = bbox(i,4);
        p1 = pIniBbox;
        p2 = pIniBbox+[anchoBbox,0]';
        p3 = pIniBbox+[anchoBbox,altoBbox]';
        p4 = pIniBbox+[0,altoBbox]';
        if(areaDetect(i) >= (areaLacasito*1.5))
            line ([p1(1),p2(1),p3(1),p4(1),p1(1)],
[p1(2),p2(2),p3(2),p4(2),p1(2)],'LineWidth',2 ,'Color', 'y');
        else
            line ([p1(1),p2(1),p3(1),p4(1),p1(1)],
[p1(2),p2(2),p3(2),p4(2),p1(2)],'LineWidth',2 ,'Color', 'g');
        end
    end
    hold off

end
```











VERSIÓN AVANZADA

Para conseguir realizar la separación de los "lacasitos" solapados del mismo color, se realiza un segundo paso de procesado, a través del cual se consigue individualizar los casos problemáticos. Así mismo, también se sustituyen las funciones de alto nivel de Matlab por específicas desarrolladas por nosotros mismos, cuyo contenido se desarrolla en el último apartado de esta memoria.

De nuevo, distinguimos 2 códigos en función de si se umbraliza por modelo RGB o HSV.

*RGB AVANZADO

Respecto a la versión básica, se hace una segunda comprobación con el área, en donde si se identifica un área cuyo valor supera el correspondiente al doble a un "lacasito" podemos saber que nos encontramos ante un caso de solapes.

Para conseguir la separación y poder obtener las características de forma individual, se realiza primero una erosión con un radio muy grande que asegure tener ambos "lacasitos" separados, a pesar de verse reducidos sus tamaños.

Posteriormente, a través del etiquetado, generamos 2 imágenes donde cada una contiene a un solo lacasito por separado. Se prosigue aplicando un dilatado para recuperar el tamaño del lacasito original (lo cuál a priori debe mantener la forma del objeto). A través de estas nuevas imágenes, se obtienen las características de interés que restaban por obtener y se almacenan.

Se muestra únicamente las partes de código novedosas respecto a las versiones básicas.

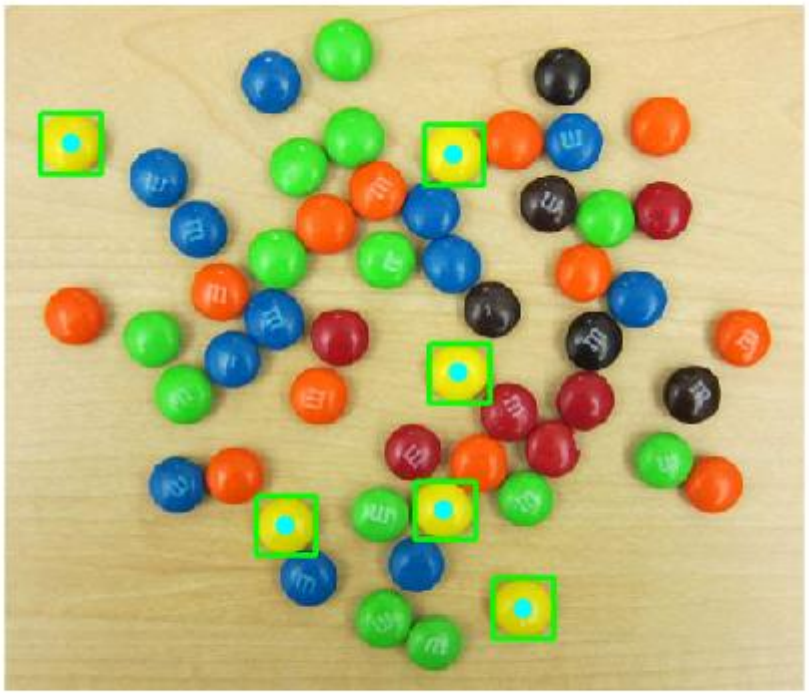
Dentro del apartado donde se aplicaba el filtrado por área se añade:

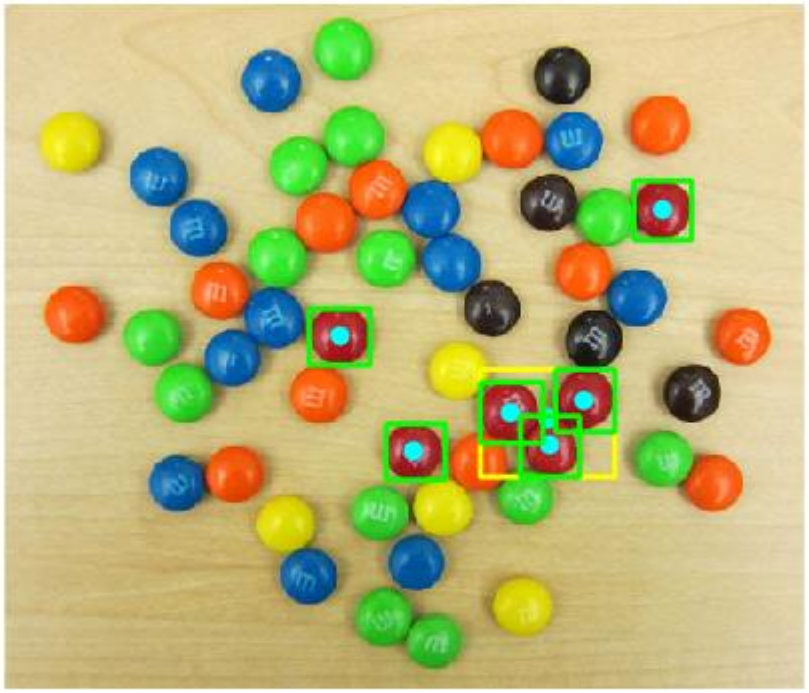
```
       elseif(areaObjeto.Area > 1.7*areaLacasito)
            fPlantillaIndiv = (fPlantillaEtiq == i); %selecciona conjunto de lacasitos
con sus Values asociados
            fPlantillaIndiv = procesadoErosion(fPlantillaIndiv,creaPlantillaDisco(12));
            fPlantillaIndiv=bwlabel(fPlantillaIndiv);
            fPlantillaSeparado=zeros(M,N,max(max(fPlantillaIndiv)));
            for j=1:max(max(fPlantillaIndiv))
                fPlantillaSeparado(:,:,j) = fPlantillaIndiv==j; %Lacasito separado
reducido
                fPlantillaSeparado(:,:,j) =
procesadoDilatado(fPlantillaSeparado(:,:,j),creaPlantillaDisco(12)); %Lacasito separado
con tamaño recuperado
                data = regionpropsManual(fPlantillaSeparado(:,:,j));
                bboxSeparated(x,:)= cat(1,data.BoundingBox);
                centroidsSeparated(x,:)= cat(1,data.Centroid);
                areaSeparated(x,:)= cat(1,data.Area);
                x=x+1;
            end
        end
```

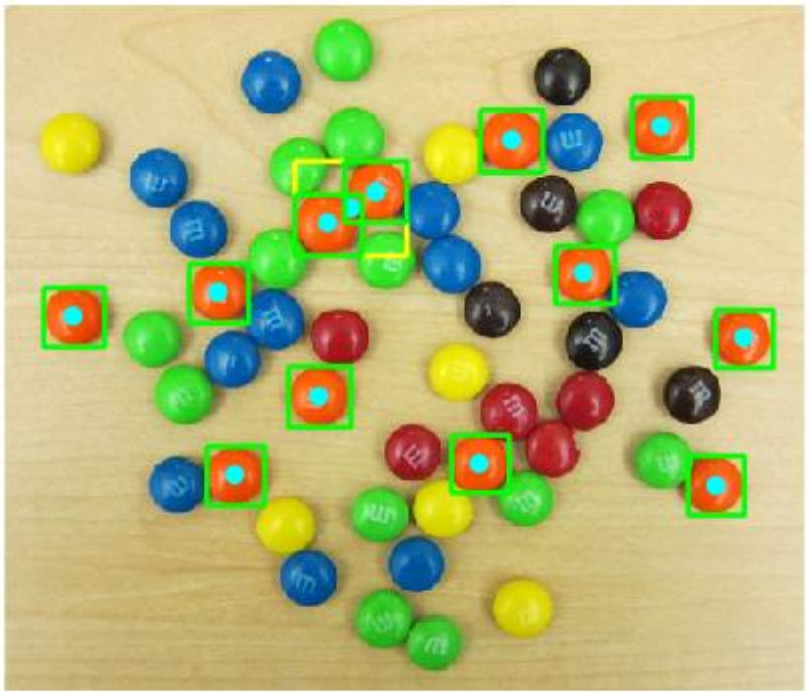
Por último, como novedad en esta parte de código, se concatenan los datos de los “lacasitos” individualizados con todos los demás.

```
datos = regionpropsManual(fPlantillaLacasitos);
bbox = cat(1,datos.BoundingBox,bboxSeparated);
sizebbox=size(bbox);
centroids = cat(1,datos.Centroid,centroidsSeparated);
areaDetect = cat(1,datos.Area,areaSeparated);
```

Finalmente, los resultados que se obtienen con este nuevo paso adicional son los deseados:







*HSV AVANZADO

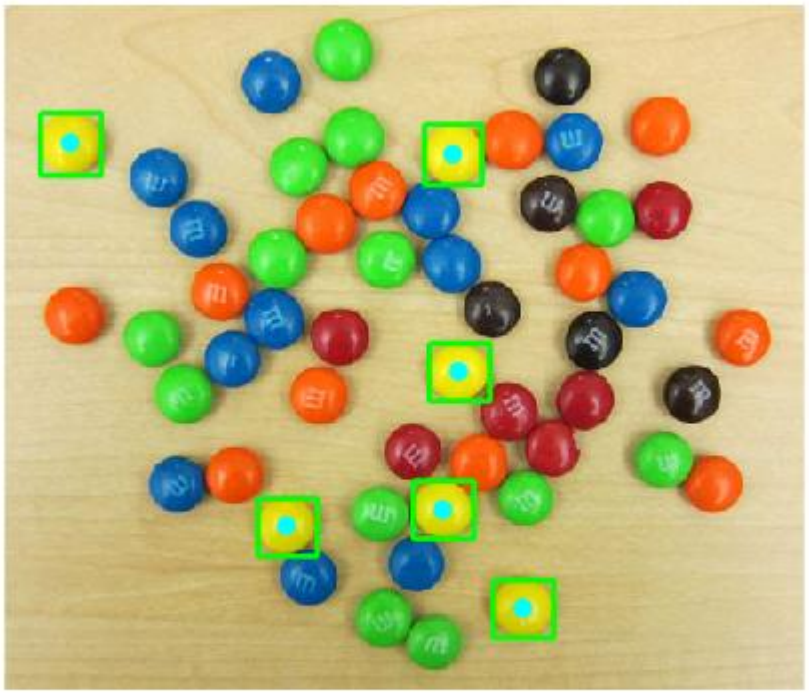
De la misma manera, pero con resultados más precisos, se procede con el modelo de color HSV. En el filtrado por área, al igual que en el caso anterior, incluimos:

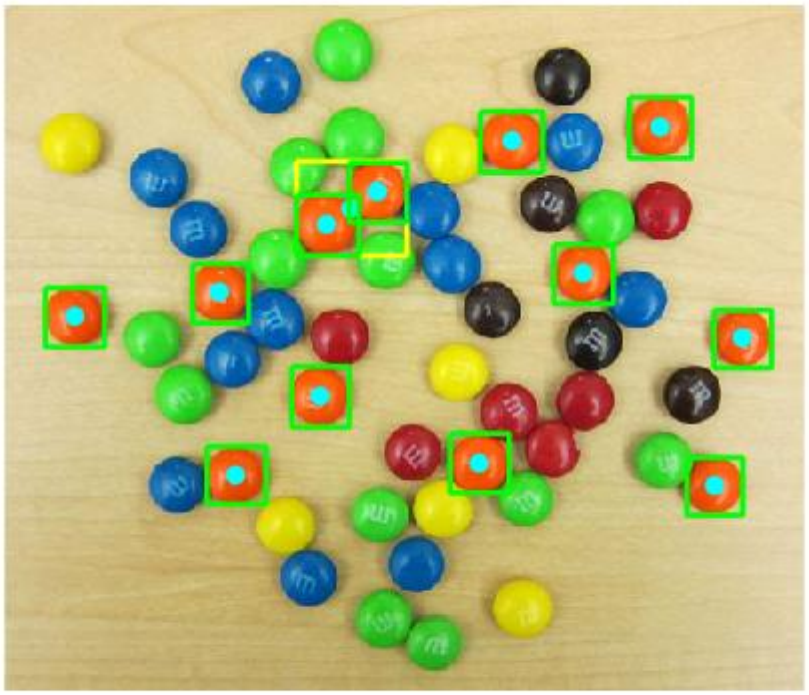
```
        elseif(areaObjeto.Area > 1.7*areaLacasito)
            fPlantillaIndiv = (fPlantillaEtiq == i); %selecciona conjunto de lacasitos
con sus Values asociados
            fPlantillaIndiv = procesadoErosion(fPlantillaIndiv,creaPlantillaDisco(10));
            fPlantillaIndiv=bwlabel(fPlantillaIndiv);
            fPlantillaSeparado=zeros(M,N,max(max(fPlantillaIndiv)));
            for j=1:max(max(fPlantillaIndiv))
                fPlantillaSeparado(:,:,j) = fPlantillaIndiv==j; %Lacasito separado
reducido
                fPlantillaSeparado(:,:,j) =
procesadoDilatado(fPlantillaSeparado(:,:,j),creaPlantillaDisco(10)); %Lacasito separado
con tamaño recuperado
                data = regionpropsManual(fPlantillaSeparado(:,:,j));
                bboxSeparated(x,:)= cat(1,data.BoundingBox);
                centroidsSeparated(x,:)= cat(1,data.Centroid);
                areaSeparated(x,:)= cat(1,data.Area);
                x=x+1;
            end
        end
```

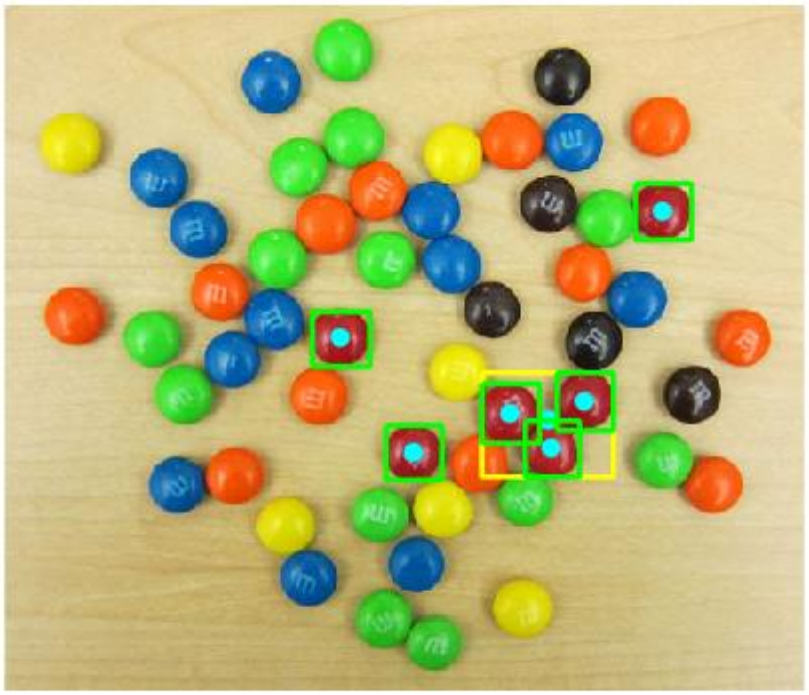
Nuevamente se almacenan los datos pertinentes para las representaciones siguiendo la misma filosofía vista.

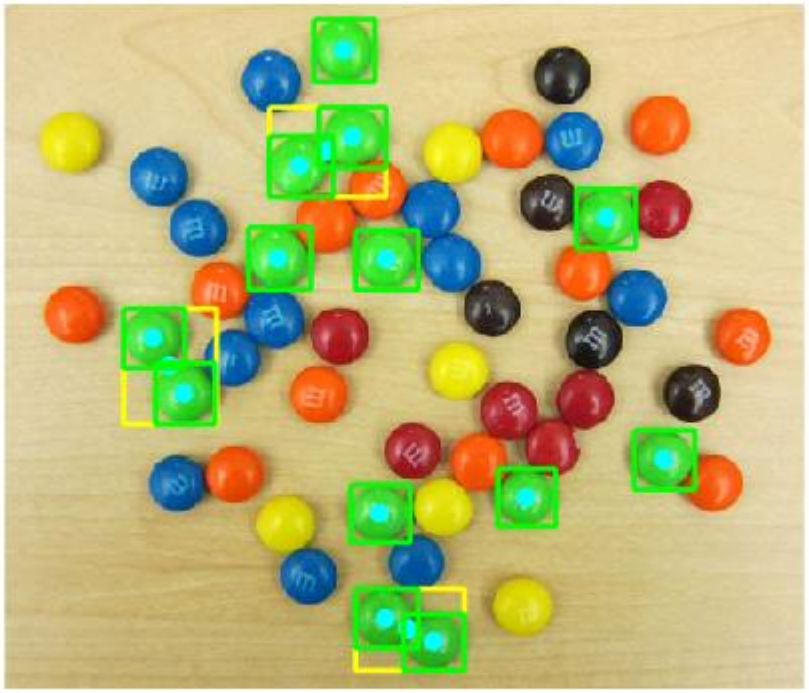
```
    datos = regionpropsManual(fPlantillaLacasitos);
    bbox = cat(1,datos.BoundingBox,bboxSeparated);
    sizebbox=size(bbox);
    centroids = cat(1,datos.Centroid,centroidsSeparated);
    areaDetect = cat(1,datos.Area,areaSeparated);
```

Los resultados son prácticamente idénticos a los obtenidos en RGB gracias al segundo procesado, aunque si bien al ampliar en detalle puede verse más precisión en estos últimos:









FUNCIONES MANUALES

Las funciones utilizadas en las versiones avanzadas previamente mostradas son:

strel() == creaPlantillaDisco()

Con 2 bucles anidados se recorre la plantilla vacía y se rellena con 1 las posiciones correspondientes al círculo central del radio que se indique en el parámetro de la función.

```
function plantilla = creaPlantillaDisco(radio)
tamPlantilla = 2*radio+1;
plantilla = zeros(tamPlantilla,tamPlantilla);
for i=1:tamPlantilla
    for j=1:tamPlantilla
        if(sqrt((i-(radio+1))^2+(j-(radio+1))^2)<=radio)
            plantilla(i,j)=1;
        else
            plantilla(i,j)=0;
        end
    end
end
end
```

imdilate() = procesadoDilatado()

Con 2 bucles anidados se recorren todos los píxeles de la imagen, y con otros 2 bucles anidados dentro se aplica la plantilla. En el caso del dilatado, se busca el píxel máximo del entorno de vecindad y dicho valor es el que se refleja en el píxel seleccionado.

```
function processedImage = procesadoDilatado(originalImage,plantilla)
M=size(originalImage,1);
N=size(originalImage,2);
tamM_Plantilla=size(plantilla,1);
tamN_Plantilla=size(plantilla,2);
processedImage=zeros(M,N);


for i=1:M
    for j=1:N
        maximo=originalImage(i,j);
        %bucle recorriendo entorno de vecindad con índices relativos
        for iPlantilla=1:tamM_Plantilla
            for jPlantilla=1:tamN_Plantilla
                if(plantilla(iPlantilla,jPlantilla)==1 & (i-
floor(tamM_Plantilla/2)+iPlantilla-1)>0 & (i-floor(tamM_Plantilla/2)+iPlantilla-1)<=M &
(j-floor(tamN_Plantilla/2)+jPlantilla-1)>0 & (j-floor(tamN_Plantilla/2)+jPlantilla-
1)<=N)
                    if(originalImage(i-floor(tamM_Plantilla/2)+iPlantilla-1,j-
floor(tamN_Plantilla/2)+jPlantilla-1)>maximo)
                        maximo=originalImage(i-floor(tamM_Plantilla/2)+iPlantilla-1,j-
floor(tamN_Plantilla/2)+jPlantilla-1);
                    end
                end
            end
        end
```

```
        processedImage(i,j)=maximo;
    end
end
end
```

imerode() = procesadoErosion()

Con 2 bucles anidados se recorren todos los píxeles de la imagen, y con otros 2 bucles anidados dentro se aplica la plantilla. En el caso del erosionado, se busca el píxel mínimo del entorno de vecindad y dicho valor es el que se refleja en el píxel seleccionado.

```
function processedImage = procesadoErosion(originalImage,plantilla)
M=size(originalImage,1);
N=size(originalImage,2);
tamM_Plantilla=size(plantilla,1);
tamN_Plantilla=size(plantilla,2);
processedImage=zeros(M,N);
for i=1:M
    for j=1:N
        minimo=originalImage(i,j);
        %bucle recorriendo entorno de vecindad con índices relativos
        for iPlantilla=1:tamM_Plantilla
            for jPlantilla=1:tamN_Plantilla
                if(plantilla(iPlantilla,jPlantilla)==1 & (i-
floor(tamM_Plantilla/2)+iPlantilla-1)>0 & (i-floor(tamM_Plantilla/2)+iPlantilla-1)<=M &
(j-floor(tamN_Plantilla/2)+jPlantilla-1)>0 & (j-floor(tamN_Plantilla/2)+jPlantilla-
1)<=N)
                    if(originalImage(i-floor(tamM_Plantilla/2)+iPlantilla-1,j-
floor(tamN_Plantilla/2)+jPlantilla-1)<minimo)
                        minimo=originalImage(i-floor(tamM_Plantilla/2)+iPlantilla-1,j-
floor(tamN_Plantilla/2)+jPlantilla-1);
                    end
                end
            end
        end
        processedImage(i,j)=minimo;
    end
end
end
```

imopen() == procesadoOpen()

```
function  processedImage = procesadoOpen(originalImage,plantilla)
   processedImage = procesadoErosion(originalImage,plantilla);
   processedImage = procesadoDilatado(processedImage,plantilla);
end
```

imclose() == procesadoClose()

```
function  processedImage = procesadoClose(originalImage,plantilla)
   processedImage = procesadoDilatado(originalImage,plantilla);
   processedImage = procesadoErosion(processedImage,plantilla);
end
```

regionprops() == regionpropsManual()

Con 2 bucles anidados, recorre todos los píxeles de la imagen (que previamente han sido etiquetados por la función bwlabel) para así seleccionar el objeto del que guardar características.

Para cada objeto identificado de la plantilla se calcula el área sumando todos los píxeles que forman parte del objeto en cuestión. Para el calculo de los centroides se hace una media de las coordenadas x e y de los pixeles del objeto. En otras palabras, se suman todas las x e y en cada pasada y posteriormente se divide entre el numero total de pixeles, lo que es equivalente en este caso a dividir entre el área.

A continuación, para conseguir los datos asociados a los boundingboxes lo que se propone es almacenar los valores máximos y mínimos de x e y. De esta forma tendremos disponibles las coordenadas del punto del rectángulo (esquina superior izquierda) y es trivial el cálculo del ancho y alto de las cajas.

Por último, devolvemos la información en forma de estructura tal y como lo hace la función original. De esta forma podemos usar exactamente el mismo código desarrollado simplemente cambiando la función a la que se llama.

```
function [out] = regionpropsManual(Plantilla)
%Función regionprops() desarrollada manualmente

[M,N] = size(Plantilla);
PlantillaEtiq = bwlabel(Plantilla);

%Areas y Centroids
Areas=zeros(max(max(PlantillaEtiq)),1);
Centroids=zeros(max(max(PlantillaEtiq)),2);
BoundingBox=zeros(max(max(PlantillaEtiq)),4);
BoundingBox(:,1)=ones(max(max(PlantillaEtiq)),1)*N; %Inicializo valor máximo
BoundingBox(:,2)=ones(max(max(PlantillaEtiq)),1)*M; %Inicializo valor máximo
```

```
for k=1:max(max(PlantillaEtiq))
    for y=1:M
        for x=1:N
            if(PlantillaEtiq(y,x)==k)
                Areas(k)=Areas(k)+1; % Areas
                Centroids(k,1)=Centroids(k,1)+x;
                Centroids(k,2)=Centroids(k,2)+y;
                %Xmin
                if(x<BoundingBox(k,1))
                    BoundingBox(k,1)=x; %Xmin Bbox
                end
                %Ymin
                if(y<BoundingBox(k,2))
                    BoundingBox(k,2)=y; %Ymin Bbox
                end
                %Xmax
                if(x>BoundingBox(k,3))
                    BoundingBox(k,3)=x;
                end
                %Ymax
                if(y>BoundingBox(k,4))
                    BoundingBox(k,4)=y;
                end
            end
        end
    end
    Centroids(k,1)=Centroids(k,1)/Areas(k); % Posicion X Centros
    Centroids(k,2)=Centroids(k,2)/Areas(k); % Posicion Y Centros
    BoundingBox(k,3)=BoundingBox(k,3)-BoundingBox(k,1)+1; %Ancho Bbox
    BoundingBox(k,4)=BoundingBox(k,4)-BoundingBox(k,2)+1; %Alto Bbox
end

out.BoundingBox=BoundingBox;
out.Centroid=Centroids;
out.Area=Areas;

end
```