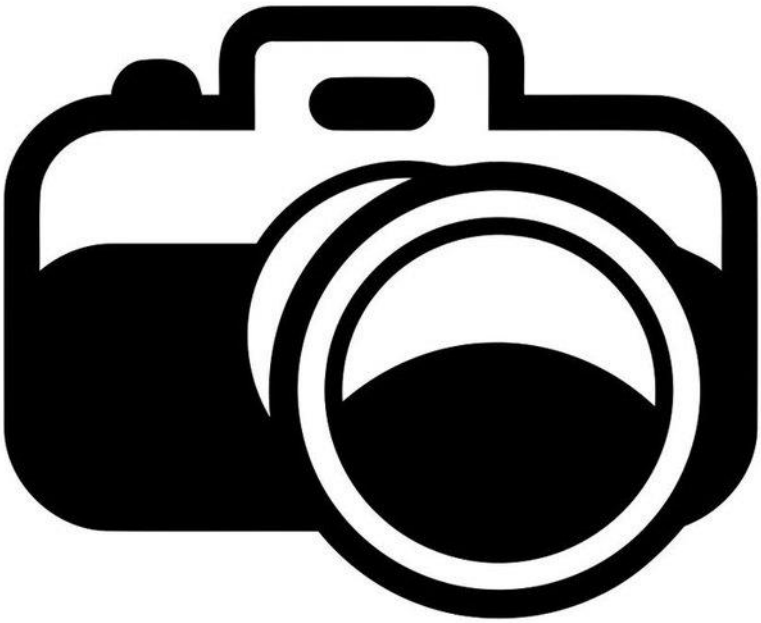

SISTEMAS DE PERCEPCIÓN

Ejercicio práctico 3: Segmentación por color

Álvaro Calvo Matos
Damián Jesús Pérez Morales

Índice

1. INTRODUCCIÓN

En el tercer ejercicio práctico propuesto, se pretende trabajar con segmentación por color a partir de la imagen de partida siguiente:

Ilustración 1. Imagen de partida

Con ella, se quiere identificar cada una de las figuras según el color que, en este caso, se encuentran los siguientes colores: rojo, verde, azul, amarillo, naranja y negro. Para ello, será necesario hacer unas plantillas con unos umbrales definidos para cada intensidad RGB para obtener la plantilla de cada uno de los colores para, así, poder tratar con cada color por separado. En este caso, en vez de trabajar con RGB, se trabajará con HSV por ser más intuitivo la separación de colores. En el script "*main.c*", está explicado el procedimiento paso por paso.

2. PROCEDIMIENTO

2.1. FILTRO DE LA MEDIANA A LA IMAGEN DE PARTIDA

Teniendo en cuenta que no se tiene el mejor fondo de imagen para segmentar los colores de cada objeto a identificar y que, además, en cada objeto de interés se ve un reflejo luminoso (brillo, alta intensidad en RGB), se recurrirá a filtrar la imagen con la mediana para tener así un rango de intensidades más uniforme.

Ilustración 2. Imagen procesada con el filtro de la mediana

2.2. INTENSIDADES DE INTERÉS EN HSV

Se ha empleado la función de *colorThresholder(imagen)* de MATLAB para poder identificar las intensidades en HSV color a color para, así, poder obtener los umbrales de interés con el fin de sacar las plantillas de cada color. En la función *umbralesHSV.m* se muestran los umbrales de interés y, las plantillas resultantes, quedan:

- Plantilla de color rojo

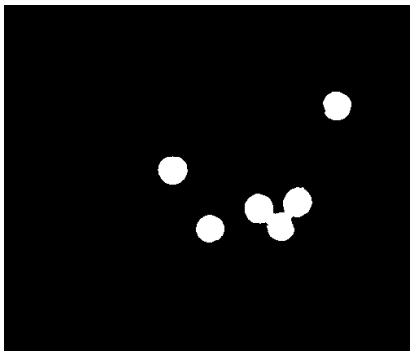

Ilustración 3. Plantilla roja sin tratar

- Plantilla de color verde

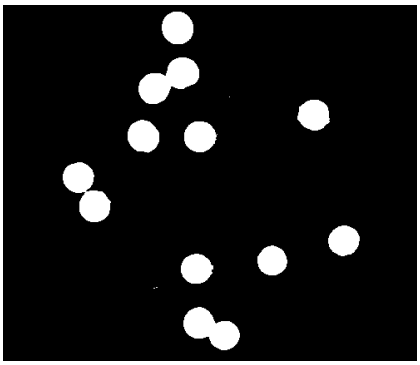

Ilustración 4. Plantilla verde sin tratar

- Plantilla de color azul

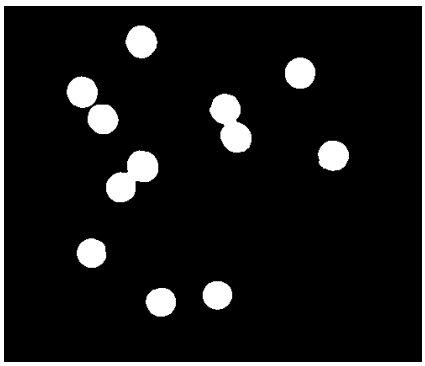

Ilustración 5. Plantilla azul sin tratar

- Plantilla de color amarillo

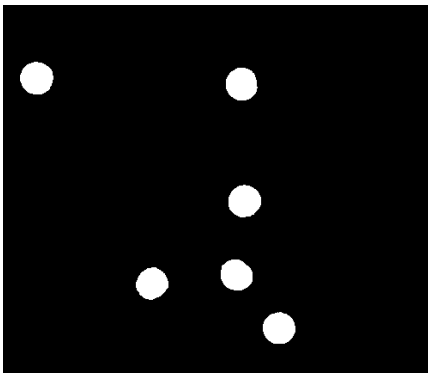

Ilustración 6. Plantilla amarilla sin tratar

- Plantilla de color naranja

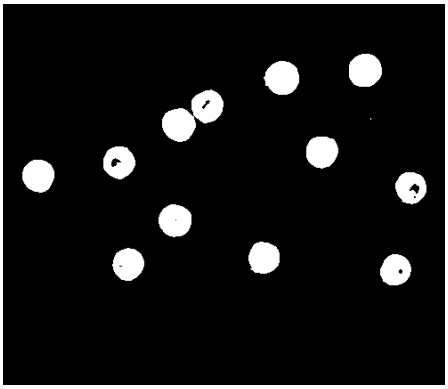

Ilustración 7. Plantilla naranja sin tratar

- Plantilla de color negro

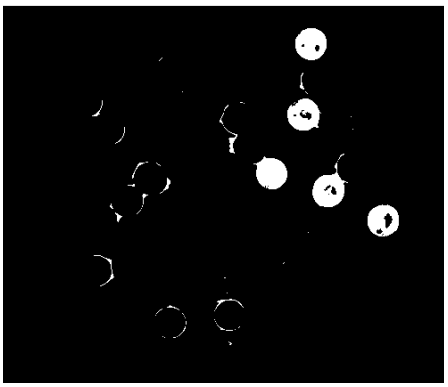

Ilustración 8. Plantilla negra sin tratar

Código de *umbralesHSV.m*

```
% Script donde se guardan los umbrales para identificar cada figura
function [uR,uG,uB,uY,uO,uK] = umbralesHSV(gMediana)
% colorThresholder(gMediana);
uR = [0.949 0.020; 0.317 1; 0.295 1];
uG = [0.205 0.372; 0.361 1; 0.251 1];
uB = [0.511 0.630; 0.235 1; 0 1];
uY = [0.136 0.203; 0.612 1; 0.612 1];
uO = [0.964 0.122; 0.639 1; 0.781 1];
uK = [0 1; 0 0.536; 0 0.464];


% Aspecto del vector de umbrales:
% [Hmin Hmax; Smin Smax; Vmin Vmax]
```

2.3. APLICAR MÉTODOS MORFOLÓGICOS

Se puede notar en las plantillas anteriores que aparecen zonas espurias que pueden dar problemas futuros a la hora de etiquetas a los objetos de interés. Por ello, se aplican métodos morfológicos (erosión y/o dilatación) a cada una de las plantillas empleando elementos estructurantes circulares de diferentes radios según el caso, con el fin de obtener una plantilla solo con los objetos a identificar. Para realizar la erosión y la dilatación, se han realizado los scripts *erosionaImagen.m, dilataImagen.m, abreImagen.m, cierraImagen.,* Los resultados son los siguientes:

- Plantilla de color rojo

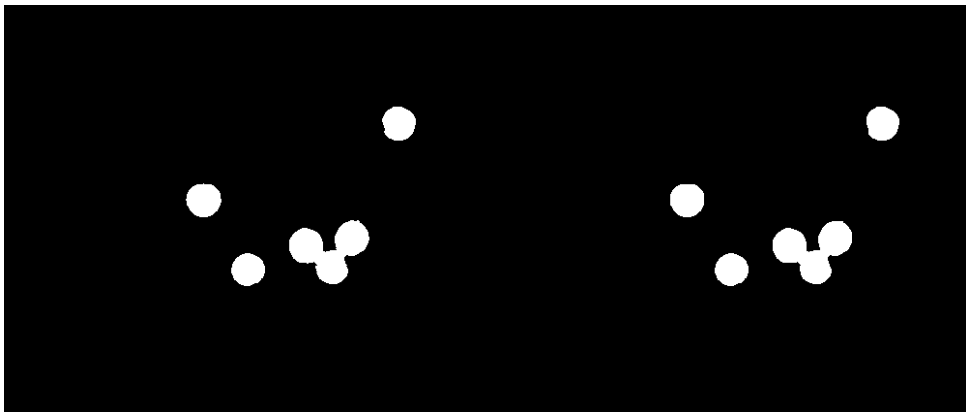

Ilustración 9. Plantilla roja corregida

- Plantilla de color verde

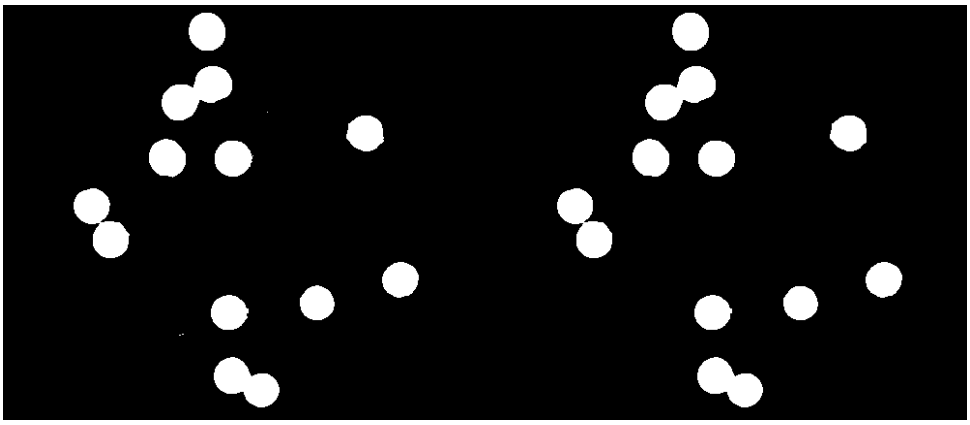

Ilustración 10. Plantilla verde corregida

- Plantilla de color azul

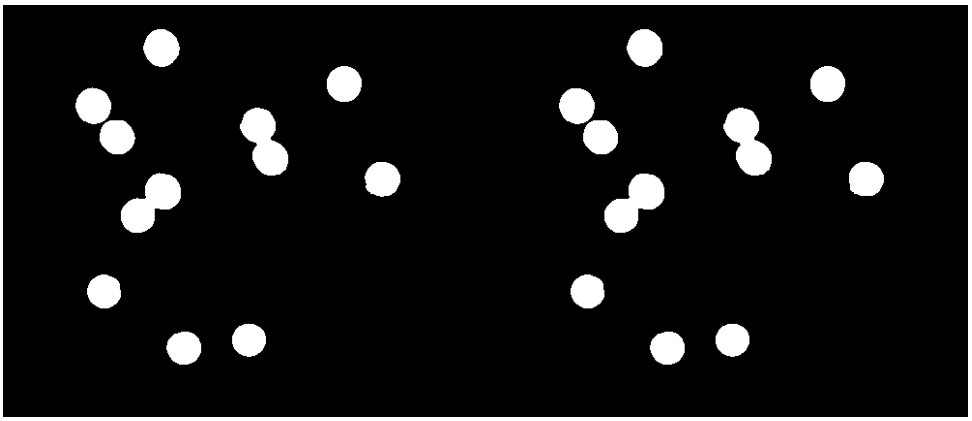

Ilustración 11. Plantilla azul corregida

- Plantilla de color amarillo

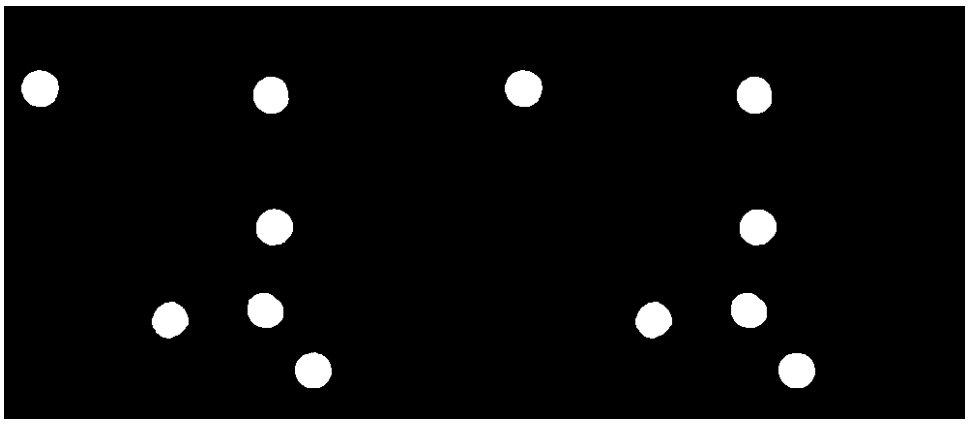

Ilustración 12. Plantilla amarilla corregida

- Plantilla de color naranja

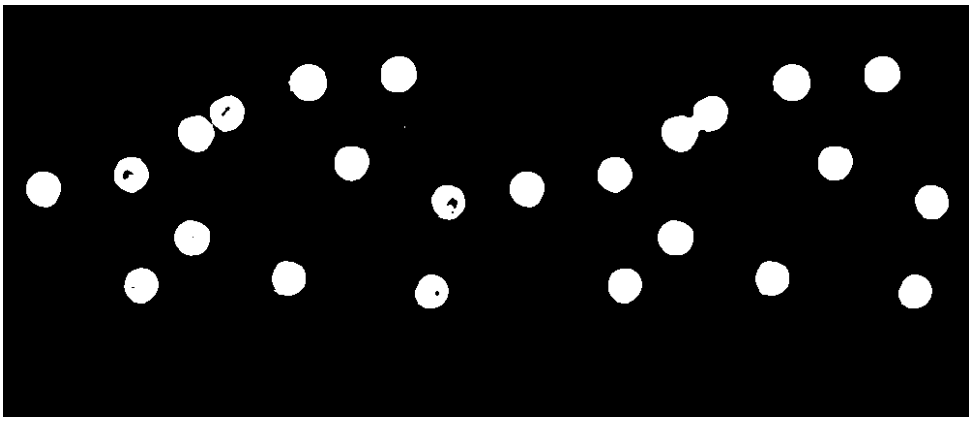

Ilustración 13. Plantilla naranja corregida

- Plantilla de color negro

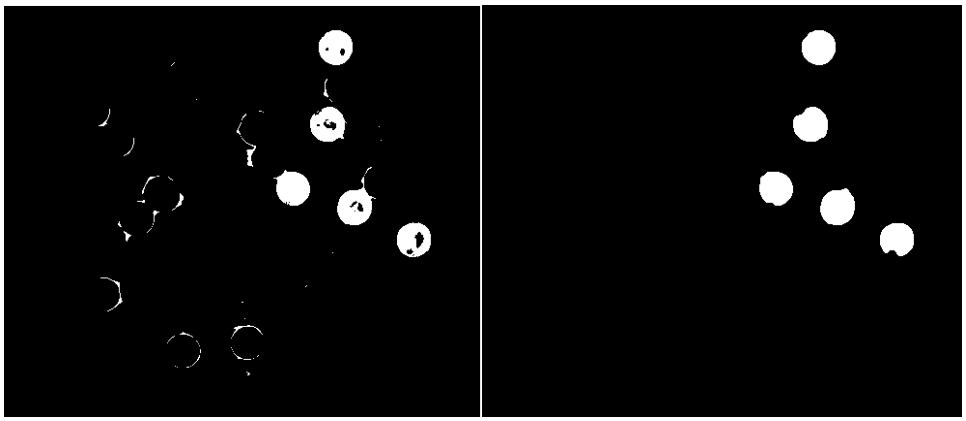

Ilustración 14. Plantilla negra corregida

Código de *erosionaImagen.m*:

```
function plantillaErosionada = erosionaImagen(plantilla,mascara)

% Función que hace la erosión de una plantilla dado la máscara que se
desee

% Dimensiones de la plantilla
[M,N] = size(plantilla);

% Obtencion del radio de la plantilla
radioMask = floor(size(mascara.Neighborhood)/2);

% Inicializamos a cero la imagen erosionada resultante
plantillaErosionada = zeros(M,N);

% Procedemos a operar con la mascara sobre la plantilla
% Recorremos pixel a pixel la imagen a erosionar
for y = 1+radioMask:M-radioMask
    for x = 1+radioMask:N-radioMask
        % Si el resultado de multiplicar la mascara por el area de la
        % imagen alrededor del pixel actual es exactamente igual que la
        % propia mascara, el pixel actual debe ponerse a 1
        if(mascara.Neighborhood.*plantilla(y-radioMask:y+radioMask,x-
radioMask:x+radioMask) == mascara.Neighborhood)
           plantillaErosionada(y,x) = 1;
        end
    end
end
```

Código de *dilataImagen.m*:

```
function plantillaDilatada = dilataImagen(plantilla,mascara)

% Función que hace la erosión de una plantilla dado la máscara que se
desee
% Dimensiones de la plantilla
[M,N] = size(plantilla);

% Obtenemos el radio de la mascara
radioMask = floor(size(mascara.Neighborhood)/2);

% Inicializamos a cero la imagen dilatada resultante
plantillaDilatada = zeros(M,N);

% Procedemos a operar con la mascara sobre la plantilla
% Recorremos pixel a pixel la imagen a dilatar
for y = 1+radioMask:M-radioMask
    for x = 1+radioMask:N-radioMask
        % Si al multiplicar la mascara por el area alrededor del pixel
        % actual resulta algun 1, el pixel actual debe ponerse a 1
          if(sum(sum(mascara.Neighborhood.*plantilla(y-
radioMask:y+radioMask,x-radioMask:x+radioMask))))
                plantillaDilatada(y,x) = 1;
        end
    end
end
```

Código de *abreImagen.m*:

```
function plantillaAbierta = abreImagen(plantilla, mascara)

    % El cierre consiste en concatenar las operaciones de erosionado y
    % dilatado en ese mismo orden
    plantillaAbierta = erosionaImagen(plantilla, mascara);
    plantillaAbierta = dilataImagen(plantillaAbierta, mascara);
```

Código de *cierraImagen.m*:

```
function plantillaCerrada = cierraImagen(plantilla, mascara)

    % El cierre consiste en concatenar las operaciones de dilatado y
    % erosionado en ese mismo orden
    plantillaCerrada = dilataImagen(plantilla, mascara);
    plantillaCerrada = erosionaImagen(plantillaCerrada, mascara);
```

2.4. ETIQUETADO DE PLANTILLAS

Para el etiquetado de las diferentes plantillas, se ha recurrido a la función *bwlabel()* de MATLAB y no se ha programado por complicidad. Esta función etiqueta los objetos de una imagen en blanco y negro (en este caso, las plantillas) con el fin de poder ser identificable cada uno de ellos. La plantilla etiquetada de cada color resulta:

- Plantilla de color rojo

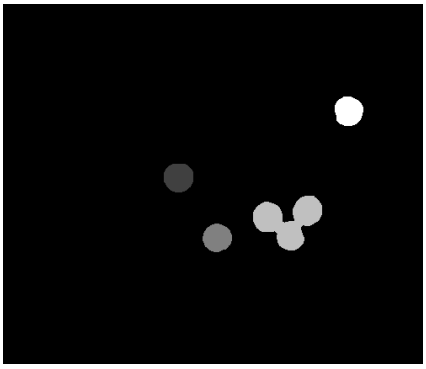

Ilustración 15. Etiquetado rojo

- Plantilla de color verde

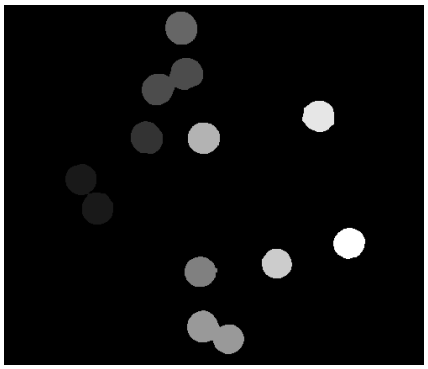

Ilustración 16. Etiquetado verde

- Plantilla de color azul

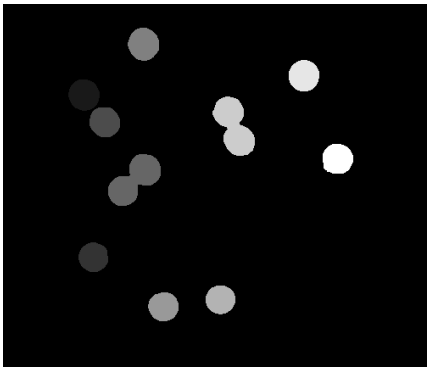

Ilustración 17. Etiquetado azul

- Plantilla de color amarillo

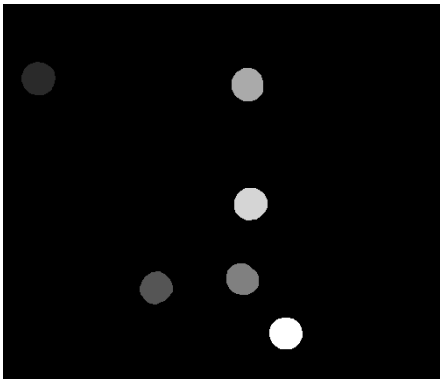

Ilustración 18. Etiquetado amarillo

- Plantilla de color naranja

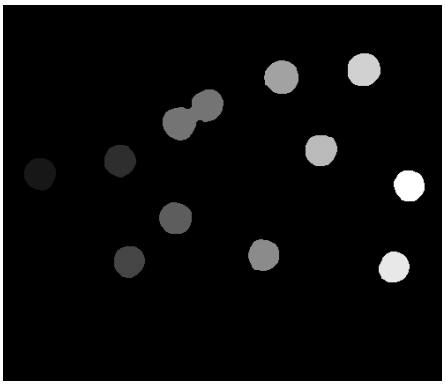

Ilustración 19.Etiquetado naranja

- Plantilla de color negro

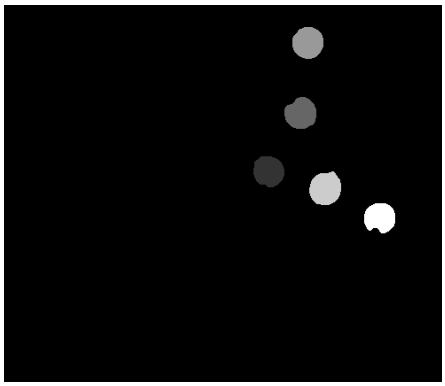

Ilustración 20. Etiquetado negro

2.5. Obtención de los momentos de inercia y Bounding Box

Con el fin de poder identificar a cada objeto de una plantilla, se hará un cálculo de los momentos de orden 0 (masa), de orden 1 (centro de masas) y de orden 2 (inercias) de cada uno de ellos; y también se hará el Bounding Box de cada uno de los objetos de interés para poder determinar su posición en la imagen. Se han empleado las funciones *calculaMomentos.m* y *BoundingBox*.m para obtener la información requerida. Los resultados obtenidos son los siguientes:

- Identificación de color rojo

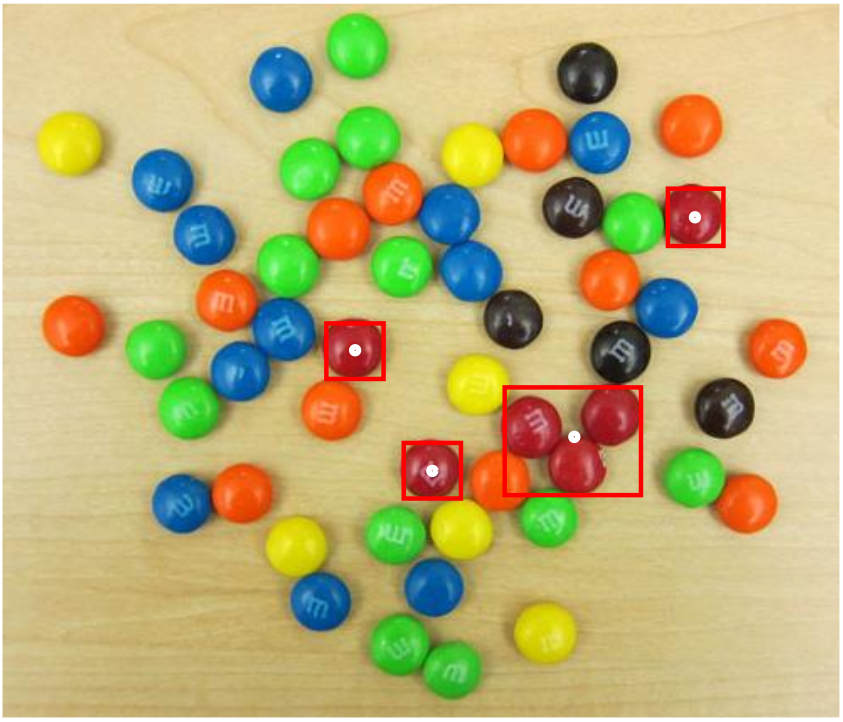

Ilustración 21. Identificación de rojos

- Identificación de color verde

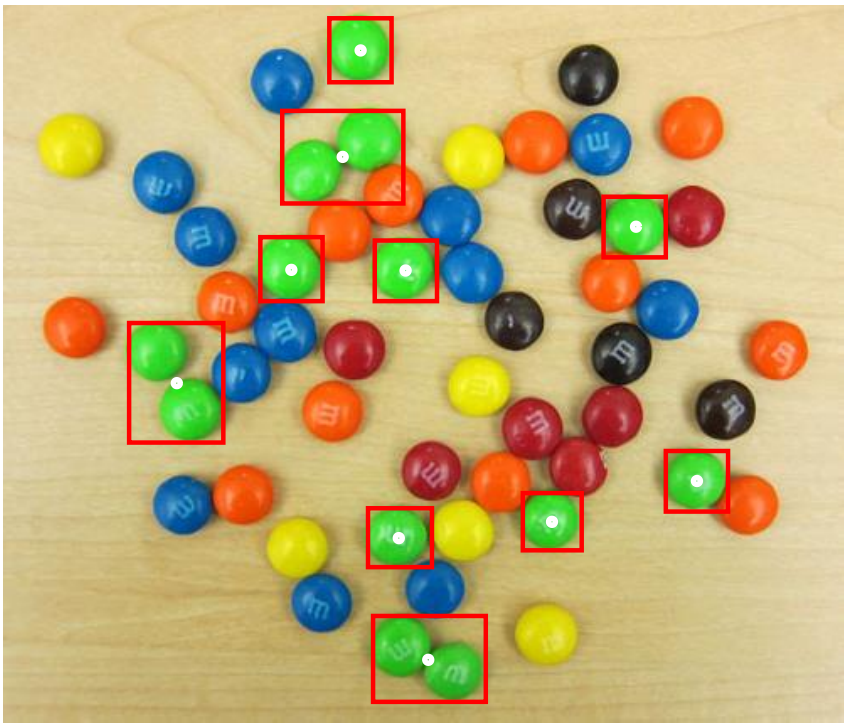

Ilustración 22. Identificación de verdes

- Identificación de color azul

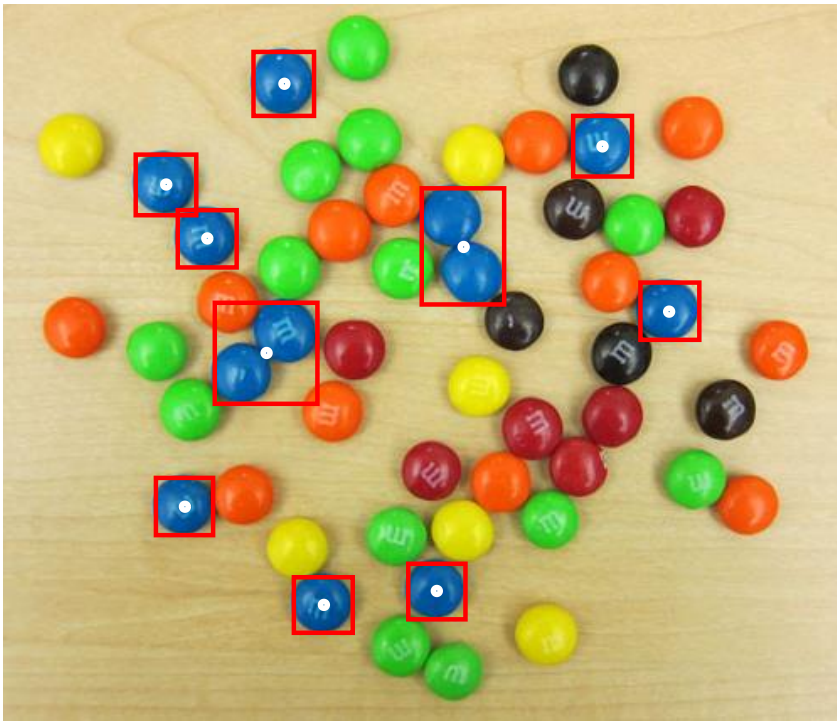

Ilustración 23. Identificación de azules

- Identificación de color amarillo

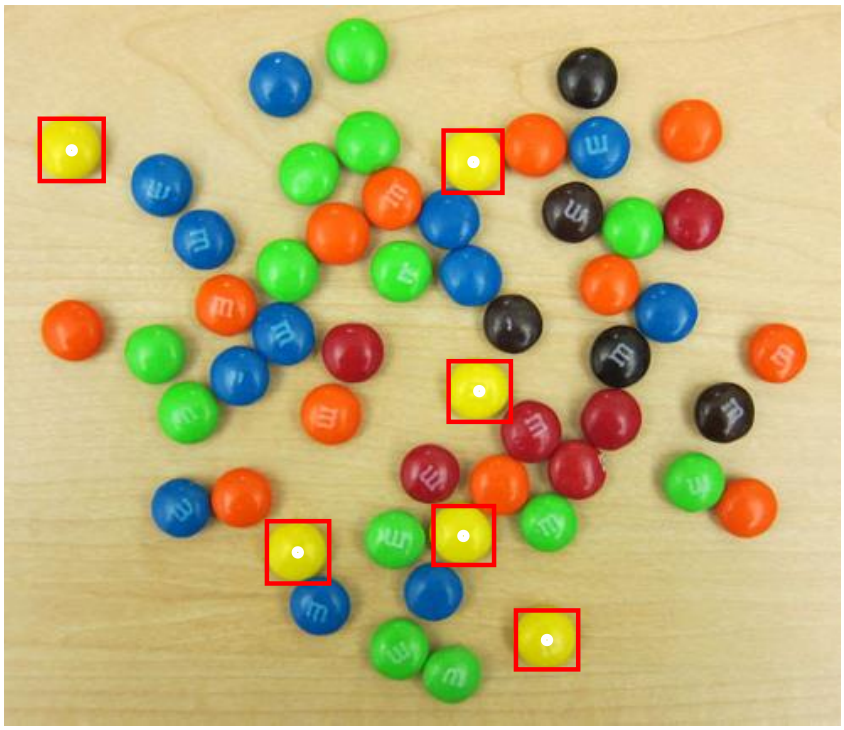

Ilustración 24. Identificación de amarillos

- Identificación de color naranja

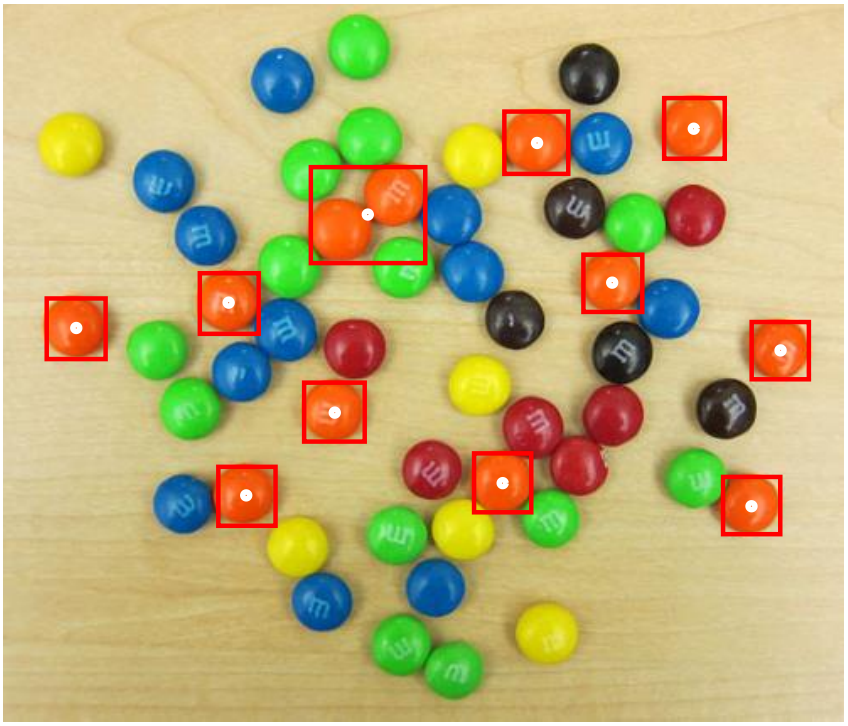

Ilustración 25. Identificación de naranjas

- Identificación de color negro

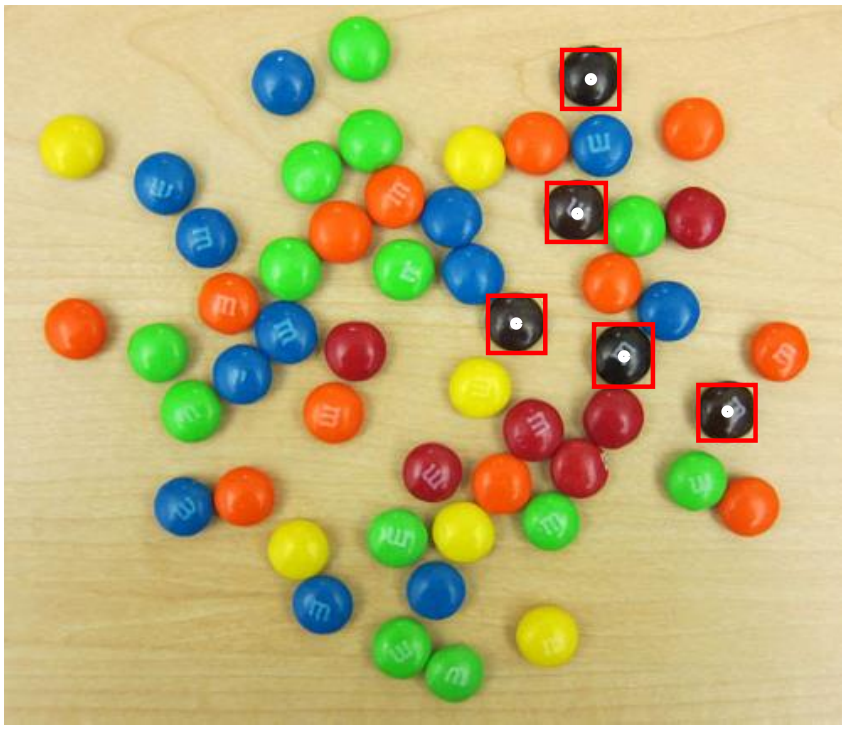

Ilustración 26. Identificación de negros

Código de *calculaMomentos.m*:

```
function [masa, cdmasa, inercia] = calculaMomentos(plantilla)

% Función que calcula los momentos de orden 0, 1 y 2 a partir de una
% plantilla
[M,N] = size(plantilla);
% Suponiendo que está etiquetada, vamos a calcular el máximo de
etiquetas:
etiqs = max(max(plantilla));

% Inicialización de los momentos
masa = zeros(1,etiqs);
cdmasa = zeros(2,etiqs);
inercia = zeros(2,2,etiqs);
m10 = zeros(1,etiqs);
m01 = zeros(1,etiqs);
mu11 = zeros(1,etiqs);
mu20 = zeros(1,etiqs);
mu02 = zeros(1,etiqs);
```

```
% Para calcular la masa, r y s deben ser 0
r = 0; s = 0;
for i = 1:etiqs
    for x = 1:N
        for y = 1:M
            if (plantilla(y,x)==i)
                masa(i) = masa(i) + (x^r)*(y^s);
            end
        end
    end
end


% Para calcular el centro de masas, es necesario conocer el momento de
% orden r=1;s=0; y viceversa
r = 1; s = 0;
for i = 1:etiqs
    for x = 1:N
        for y = 1:M
            if (plantilla(y,x)==i)
                m10(i) = m10(i) + (x^r)*(y^s);
            end
        end
    end
end


r = 0; s = 1;
for i = 1:etiqs
    for x = 1:N
        for y = 1:M
            if (plantilla(y,x)==i)
                m01(i) = m01(i) + (x^r)*(y^s);
            end
        end
    end
end

% Por tanto, el centro de masas se encuentra en:
xcm = m10./masa;
ycm = m01./masa;
cdmasa = [xcm; ycm];

% Se calcula ahora el momento central
r = 1; s = 1;
for i = 1:etiqs
    for x = 1:N
        for y = 1:M
            if (plantilla(y,x)==i)
                mu11(i) = mu11(i) + (x-xcm(i))^r*(y-ycm(i))^s;
            end
        end
    end
end
```

```
% Para calcular el centro de inercias, es necesario calcular el momento
% central de orden 2, 0 y el de 0, 2:
% Se calcula ahora el momento central
r = 2; s = 0;
for i = 1:etiqs
    for x = 1:N
        for y = 1:M
            if (plantilla(y,x)==i)
                mu20(i) = mu20(i) + (x-xcm(i))^r*(y-ycm(i))^s;
            end
        end
    end
end


r = 0; s = 2;
for i = 1:etiqs
    for x = 1:N
        for y = 1:M
            if (plantilla(y,x)==i)
                mu02(i) = mu02(i) + (x-xcm(i))^r*(y-ycm(i))^s;
            end
        end
    end
end

% Por tanto, el centro de inercias resulta:
for i = 1:etiqs
    inercia(:,:,i) = [mu02(i) -mu11(i); -mu11(i) mu20(i)];
end
```

Código de *BoundingBox.m*:

```
function BoundingBox (pEtiq,colorRectangulo)

%Se averiguan las dimensiones de la plantilla
[M,N]=size(pEtiq);

%El número máximo de etiquetas:
etiq = max(max(pEtiq));

%Se inicializan las variables de xMin, xMax, yMin, yMax
for i=1:etiq
    xMax(i) = 0;
    xMin(i) = N;
    yMax(i) = 0;
    yMin(i) = M;
```

```
     % Bucle componente a componente que, según el valor de la etiqueta,
    % establece un valor mínimo y un máximo para cada coordenada
    for y=1:M
        for x=1:N
            if (pEtiq(y,x)==(i))
                if (x<xMin(i))
                    xMin(i)=x;
                end
                if (x>xMax(i))
                    xMax(i)=x;
                end
                if (y<yMin(i))
                    yMin(i)=y;
                end
                if (y>yMax(i))
                    yMax(i)=y;
                end
            end
        end
    end
end

% BB es el vector de coordenadas mínimas y máximas
BB=[xMin;xMax;yMin;yMax];

% Se hace el cuadro que rodea a cada etiqueta
for i = 1:etiq
    rectangle('Position',[xMin(i) yMin(i) (xMax(i)-xMin(i)) (yMax(i)-
yMin(i))],'LineWidth',2,'EdgeColor',colorRectangulo);
end
```

2.6. SEPARAR OBJETOS SUPERPUESTOS

Se puede notar en el apartado anterior que en ciertos casos hay objetos que aparecen superpuestos y se identifican como uno solo a la hora de hacer el Bounding Box. Para corregirlo:

1. Se ha empleado la masa (momento de orden 0) promedio de los colores en los que se superponían objetos del mismo color y se averiguaba cuáles de ellas excedían la media de la masa, con el fin de poder tratar por separado dichos casos y se obtiene la plantilla.
2. Se ha hecho una erosión de la plantilla obtenida hasta que queden separados en la ella, con el objetivo de poder ser etiquetadas por separado e ir haciendo una plantilla para cada una de ellas para, posteriormente, ser de nuevo dilatadas (volver al tamaño de partida)
3. Se ha calculado los momentos de inercia para cada uno de los objetos por separado y se ha aplicado el Bounding Box sobre la imagen de partida.

El resultado obtenido para cada una de las imágenes que lo necesitaban es el siguiente:

- Rojo

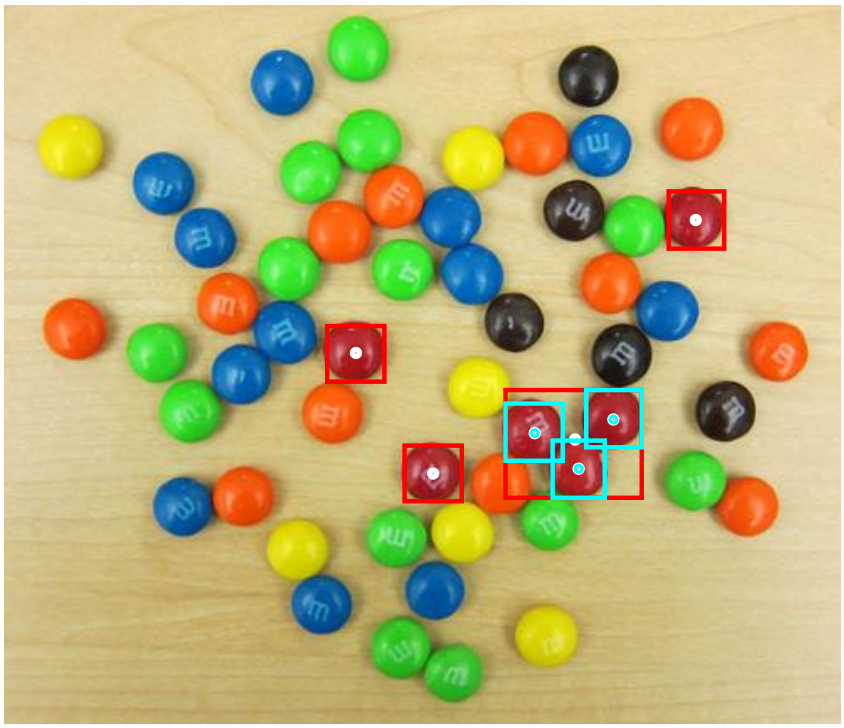

Ilustración 27. Identificación corregida de rojos

- Verde

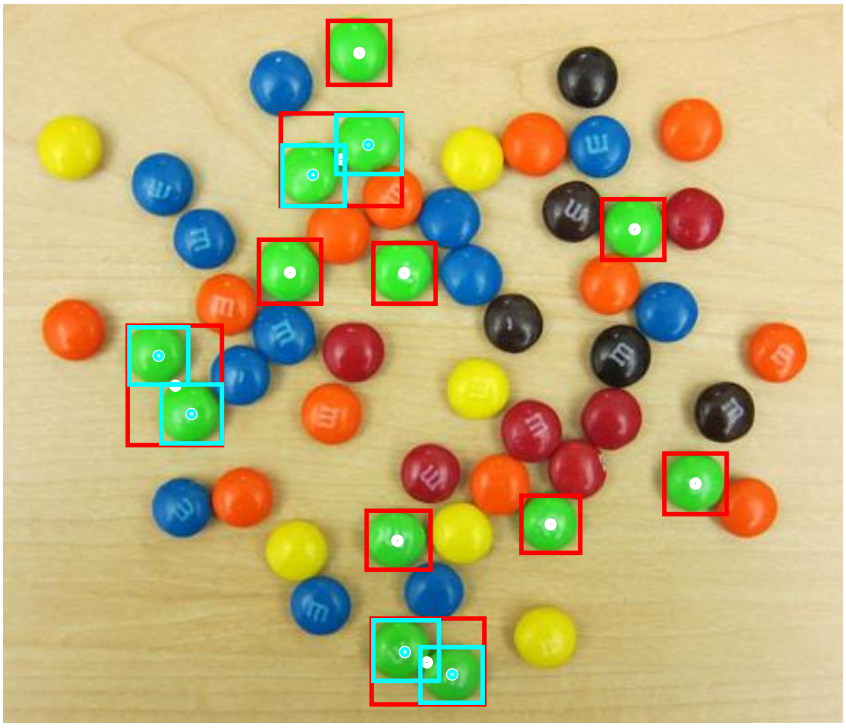

Ilustración 28. Identificación corregida de verdes

- Azul

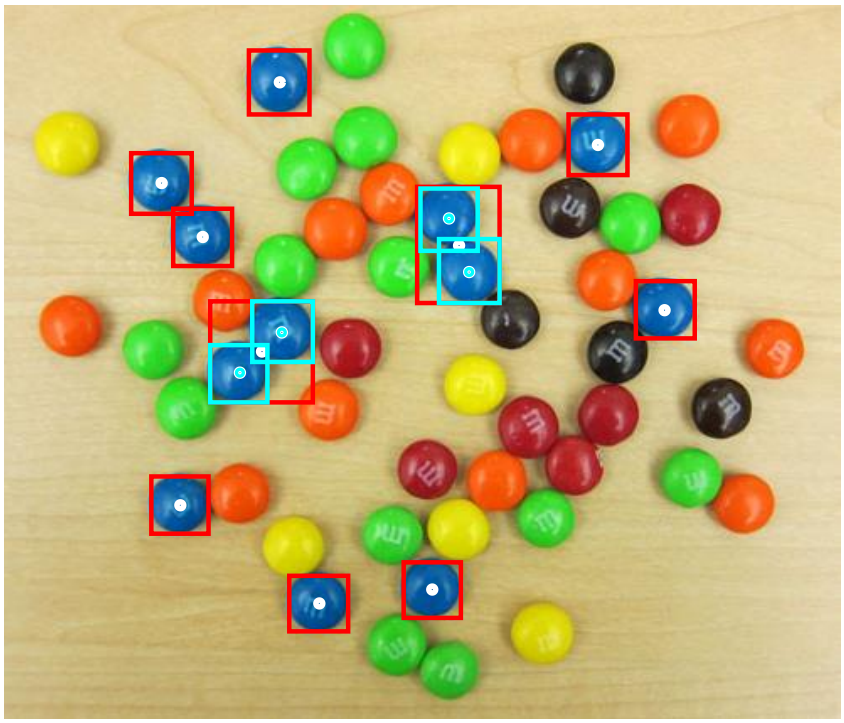

Ilustración 29. Identificación corregida de azules

- Naranja

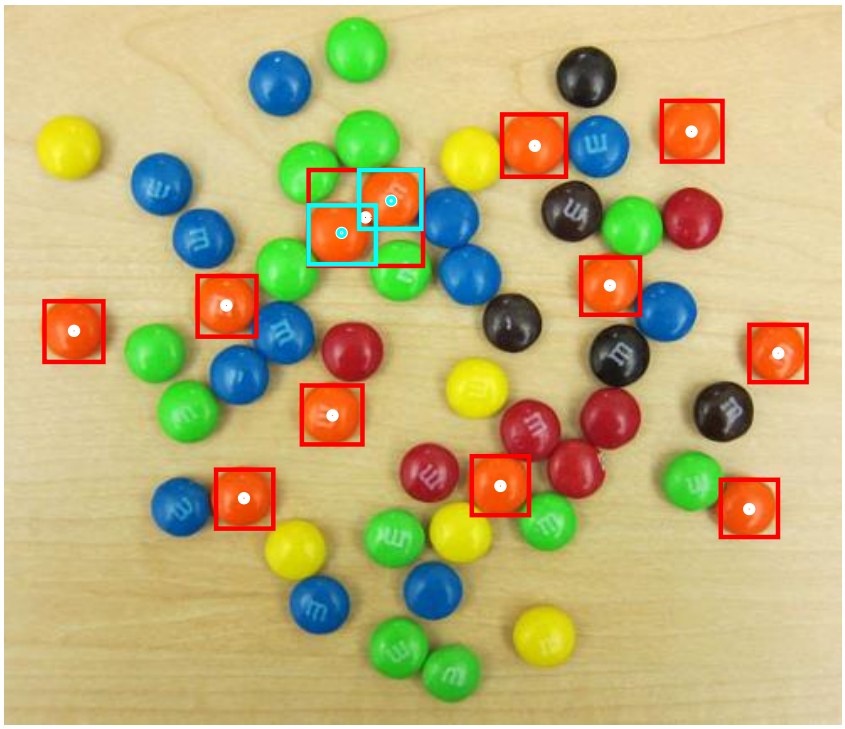

Ilustración 30. Identificación corregida de naranjas

El script implementado para la separación de objetos es *separaObjetos.m* y el código es el siguiente:

```
function separaObjetos(plantilla,etiqueta,tamanoMask)
% Función que separa los objetos que están juntos a la hora de hacer el
BB

    mask=strel('disk',tamanoMask);

    % Plantilla con el objeto de interés
    plantillaObjeto = (plantilla==etiqueta);
    plantillaErosionada = erosionaImagen(plantillaObjeto, mask);

    %--------------- Para pruebas comentar y descomentar ---------------%
%     imshow (plantillaErosionada);
%     pause();
%
%     return;
    %-------------------------------------------------------------------%

    % Una vez separados los objetos, se etiqueta de nuevo
    etiquetado2 = bwlabel(plantillaErosionada);

    % Se declara una matriz de matrices para separar en plantillas
    % distintas cada uno de los objetos
    [M,N] = size (plantillaErosionada);
    etiqSeparada = max(max(etiquetado2));
    plantillaDilatada = zeros(M,N,etiqSeparada);

    % Ahora, se estudia cada objeto por separado según cada plantilla y
se
    % le devuelve el tamaño original
    for i=1:etiqSeparada
        plantillaObjetoSeparado = (etiquetado2==i);
        plantillaDilatada(:, :, i) =
dilataImagen(plantillaObjetoSeparado, mask);
%         imshow(plantillaDilatada(:,:,i));
%         pause();
    end
```

```
    % Una vez obtenidas las plantillas, se hace el Bounding Box de los
    % objetos por separado. Para ello, es necesario obtener la masa y el
    % centro de masas de cada uno de ellos. Se va a recurrir a la función
    % calculaMomentos

    % Se inicializa la matrices de los momentos
    masaSeparada = zeros(1,etiqSeparada);
    cdmSeparada = zeros(2,etiqSeparada);
    inerciaSeparada = zeros(2,2,etiqSeparada);

    % Se calculan los momentos y se hace el Bounding Box de cada objeto
    for i = 1: etiqSeparada
        BoundingBox(plantillaDilatada(:,:,i),'c');
        [masaSeparada,cdmSeparada,inerciaSeparada] =
calculaMomentos(plantillaDilatada(:,:,i));
        viscircles(cdmSeparada',2,'LineWidth',1.5,'EdgeColor','c');
    end
```

3. CÓDIGO MAIN.C

El código principal a ejecutar es *main.c*, donde se recurre a todas las funciones anteriormente mencionadas y sigue el procedimiento anteriormente expuesto.

```
    % Ejercicio Practico 3
clc
clear all
% En primer lugar, añadimos la imagen
f = imread ('imagenDePartida.png');

figure();
imshow(f);
title('Imagen de partida');

% Antes de ver los límites de cada color, se va a aplicar un filtro de la
% mediana para tener una intensidad de las figuras a identificar no tan
% dispar

% Hay que elegir el radio de la máscara con la que se quiere operar
% rMask = 1;
% gMediana1 = filtroMediana(f,rMask);
% figure();
% imshow(gMediana1);
% title('Filtro de la media con máscara de radio 1');

rMask = 2;
gMediana = filtroMediana(f,rMask);
figure();
imshow(gMediana);
title('Filtro de la media con máscara de radio 2');

% Ahora, hacemos el Threshold para saber los umbrales.
% Se trabajará con HSV
[uR,uG,uB,uY,uO,uK] = umbralesHSV(gMediana);

% Se hará ahora una imagen por separado de cada color y, si es necesario,
% se aplicarán métodos morfológicos para identificar cada una de las
% figuras de interés
imHSV = rgb2hsv(gMediana);

% Se recorrerá pixel a pixel la imagen una vez por cada color e
% identificando los umbrales de H, S, V de cada uno de ellos para obtener
% la plantilla de cada color. Se emplea la función plantillaBinaria
[pR,pG,pB,pY,pO,pK] = plantillaBinaria(imHSV,uR,uG,uB,uY,uO,uK);

% figure()
% imshow(pR);
% pause();
% figure()
% imshow(pG);
% pause();
% figure()
% imshow(pB);
% pause();
```

```
% figure()
% imshow(pY);
% pause();
% figure()
% imshow(pO);
% pause();
% figure()
% imshow(pK);
% return;

% El elemento estructurante con el que se trabajará para aplicar métodos
% morfológicos según los umbrales HSV será de un disco de radio
arbitrario:
maskDisco3 = strel('disk',3);
maskDisco4 = strel('disk',4);
maskDisco5 = strel('disk',5);

% Viendo los resultados de cada plantilla, se operará sobre cada una de
% ellas de forma distinta

pRadj = abreImagen(pR,maskDisco3);
imshow([pR,pRadj]);
% pause();
pGadj = abreImagen(pG,maskDisco3);
imshow([pG,pGadj]);
% pause();
pBadj = abreImagen(pB,maskDisco3);
imshow([pB,pBadj]);
% pause();
pYadj = abreImagen(pY,maskDisco5);
imshow([pY,pYadj]);
% pause();
pOadj = abreImagen(pO,maskDisco3);
pOadj = cierraImagen(pOadj,maskDisco5);
imshow([pO,pOadj]);
% pause();
pKadj = abreImagen(pK,maskDisco4);
pKadj = cierraImagen(pKadj,maskDisco5);
imshow([pK,pKadj]);
% pause();
% return;


% Se hace el etiquetado de las figuras en cada caso:
pRetiq = bwlabel(pRadj);
pGetiq = bwlabel(pGadj);
pBetiq = bwlabel(pBadj);


pYetiq = bwlabel(pYadj);
pOetiq = bwlabel(pOadj);
pKetiq = bwlabel(pKadj);

imshow(pRetiq,[]);
% pause();
imshow(pGetiq,[]);
% pause();
imshow(pBetiq,[]);
```

```
% pause();
imshow(pYetiq,[]);
% pause();
imshow(pOetiq,[]);
% pause();
imshow(pKetiq,[]);
% return;

% Ahora, se hará el bounding box de cada color
figure();
imshow(f);
BoundingBox(pRetiq,'r');
% pause();
imshow(f);
BoundingBox(pGetiq,'r');
% pause();
imshow(f);
BoundingBox(pBetiq,'r');
% pause();
imshow(f);
BoundingBox(pYetiq,'r');
% pause();
imshow(f);
BoundingBox(pOetiq,'r');
% pause();
imshow(f);
BoundingBox(pKetiq,'r');
% return;

% ------------------------------------------------------------------------
--

% Asignamos los momentos de orden 0, 1 y 2 a cada uno de los colores
usando
% la función calculaMomentos
[masaR,cdmR,inerciaR] = calculaMomentos(pRetiq);
[masaG,cdmG,inerciaG] = calculaMomentos(pGetiq);
[masaB,cdmB,inerciaB] = calculaMomentos(pBetiq);
[masaY,cdmY,inerciaY] = calculaMomentos(pYetiq);
[masaO,cdmO,inerciaO] = calculaMomentos(pOetiq);
[masaK,cdmK,inerciaK] = calculaMomentos(pKetiq);

% Se representa ahora el centro de masas de cada color en cada caso
figure();
imshow(f);
BoundingBox(pRetiq,'r');
viscircles(cdmR',(2*ones(max(max(pRetiq)),1)),'LineWidth',1.5,'EdgeColor'
,'w');
pause();

imshow(f);
BoundingBox(pGetiq,'r');
viscircles(cdmG',(2*ones(max(max(pGetiq)),1)),'LineWidth',1.5,'EdgeColor'
,'w');
pause();
```

```
imshow(f);
BoundingBox(pBetiq,'r');
viscircles(cdmB',(2*ones(max(max(pBetiq)),1)),'LineWidth',1.5,'EdgeColor'
,'w');
pause();

imshow(f);
BoundingBox(pYetiq,'r');
viscircles(cdmY',(2*ones(max(max(pYetiq)),1)),'LineWidth',1.5,'EdgeColor'
,'w');
pause();

imshow(f);
BoundingBox(pOetiq,'r');
viscircles(cdmO',(2*ones(max(max(pOetiq)),1)),'LineWidth',1.5,'EdgeColor'
,'w');
pause();

imshow(f);
BoundingBox(pKetiq,'r');
viscircles(cdmK',(2*ones(max(max(pKetiq)),1)),'LineWidth',1.5,'EdgeColor'
,'w');

% return;

% Es necesario hacer un promedio de las masas de cada plantilla, ya que
se
% puede apreciar en las figuras que se da a veces la ocasión de que se
% juntan más de un objeto y, con este promedio, se podrá estudiar el caso
% en el que más de un objeto se junta, con el fin de separarlos

% Promediado de la masa de cada color
masaRprom = sum(masaR)/(max(size(masaR)));
masaGprom = sum(masaG)/(max(size(masaG)));
masaBprom = sum(masaB)/(max(size(masaB)));
masaYprom = sum(masaY)/(max(size(masaY)));
masaOprom = sum(masaO)/(max(size(masaO)));
masaKprom = sum(masaK)/(max(size(masaK)));



% Ahora, se identifican las masas que están juntas y se aplican
% transformaciones morfológicas para separarlas

% Se cierran todas las ventanas anteriormente creadas para pintar
% exclusivamente las seis imagenes finales
close all;

% Para color rojo
figure();
imshow(f); hold on;
BoundingBox(pRetiq,'r');
viscircles(cdmR',(2*ones(max(max(pRetiq)),1)),'LineWidth',1.5,'EdgeColor'
,'w');

for i = 1:(max(max(pRetiq)))
    if (floor(masaR(i)/masaRprom))
        separaObjetos(pRetiq, i, 9);
     end
end
```

```
% Para color verde
figure();
imshow(f); hold on;
BoundingBox(pGetiq,'r');
viscircles(cdmG',(2*ones(max(max(pGetiq)),1)),'LineWidth',1.5,'EdgeColor'
,'w');

for i = 1:(max(max(pGetiq)))
    if (floor(masaG(i)/masaGprom))
        separaObjetos(pGetiq, i, 10);
    end
end

hold off;


% Para color azul
figure();
imshow(f); hold on;
BoundingBox(pBetiq,'r');
viscircles(cdmB',(2*ones(max(max(pBetiq)),1)),'LineWidth',1.5,'EdgeColor'
,'w');

for i = 1:(max(max(pBetiq)))
    if (floor(masaB(i)/masaBprom))
        separaObjetos(pBetiq, i, 9);
    end
end

hold off;

% Volvemos a pintar el amarillo, ya que lo hemos cerrado
figure();
imshow(f);
BoundingBox(pYetiq,'r');
viscircles(cdmY',(2*ones(max(max(pYetiq)),1)),'LineWidth',1.5,'EdgeColor'
,'w');

% Para color naranja
figure();
imshow(f); hold on;
BoundingBox(pOetiq,'r');
viscircles(cdmO',(2*ones(max(max(pOetiq)),1)),'LineWidth',1.5,'EdgeColor'
,'w');

for i = 1:(max(max(pOetiq)))
    if (floor((masaO(i)/masaOprom)-0.1))        % El -0.1 es para evitar
que salga el objeto más grande
        separaObjetos(pOetiq, i, 9);
    end
end

hold off;

% Volvemos a pintar el negro, ya que lo hemos cerrado
figure();
imshow(f);
BoundingBox(pKetiq,'r');
viscircles(cdmK',(2*ones(max(max(pKetiq)),1)),'LineWidth',1.5,'EdgeColor'
,'w');
```

4. CONCLUSIONES

A la hora de realizar el trabajo no se han tenido muchos percances, salvo que en las funciones de *erosionaImagen.m* y *dilataImagen.m* de la forma que se realizó en primera instancia nos dio bastantes problemas porque tardaba mucho tiempo en ejecutarse el código. Finalmente, se consiguió llegar a una forma más óptima de realizar el algoritmo y, así, aumentando su eficiencia computacional.

Por otra parte, se ha apreciado que, si no se trabaja con un fondo de un color que contraste mucho con los objetos de trabajo, puede dar problemas hasta el punto de tener que aplicar métodos morfológicos antes de poder trabajar con las identificaciones de objetos.

También cabe destacar que se ha trabajado con objetos del mismo tamaño, por lo que era bastante fácil de identificar a priori qué objetos estaban superpuestos y se han podido separar con facilidad empleando el promedio de la masa. Sin embargo, si no hubiéramos trabajado con objetos de tamaño uniforme, podría haber resultado bastante más complicado hacer la separación de objetos.