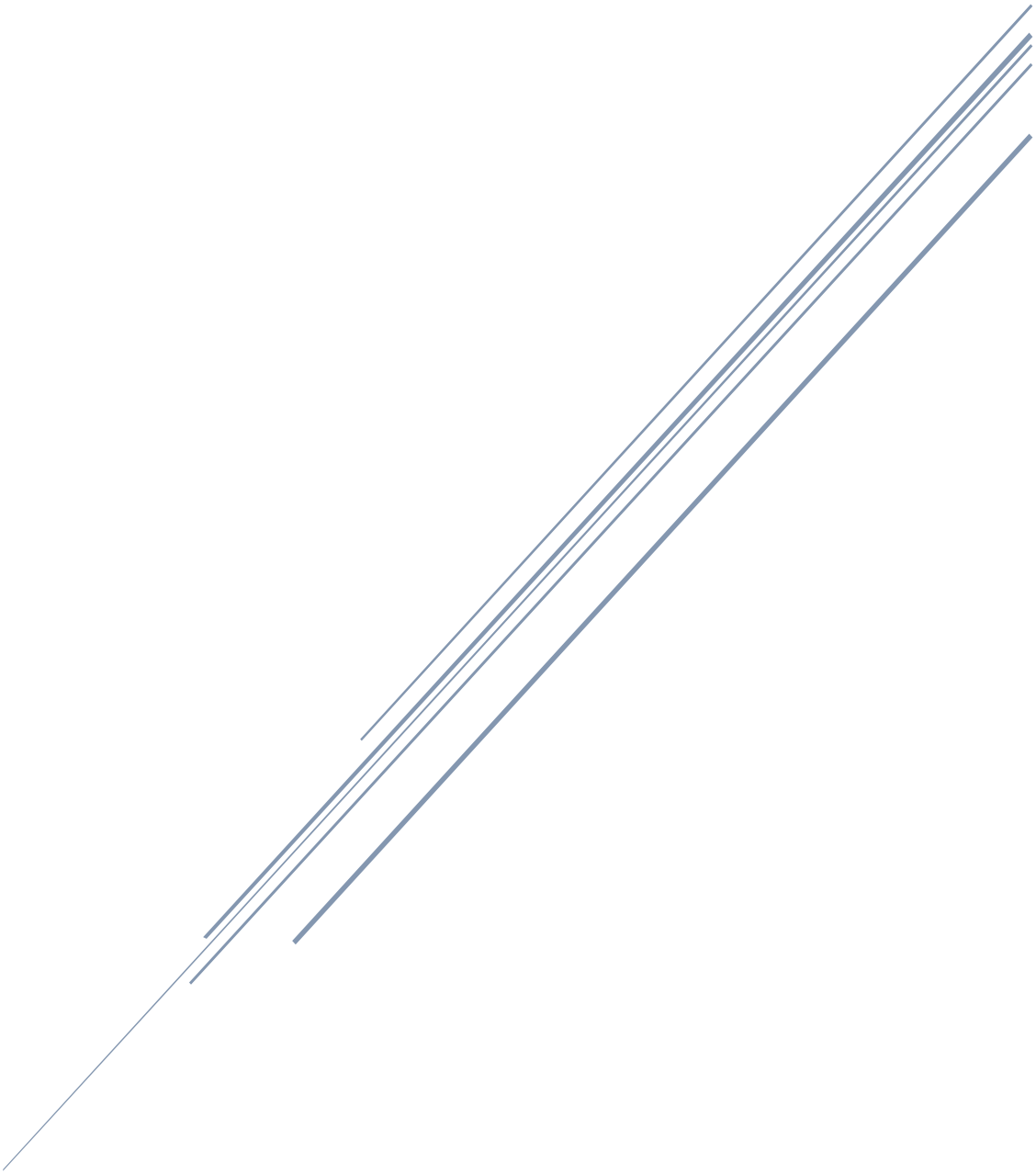
PRÁCTICA 2 – SEGMENTACIÓN POR COLOR

Sistemas de Percepción – 4 GIERM



Pablo León Barriga
2021/2022

Introducción

El programa realizado contiene una versión simple, en la que se hace uso de funciones nativas de MATLAB para el tratamiento de las imágenes, operaciones morfológicas y de obtención de las características de los objetos, y de una versión avanzada en la que se han programado dichas funciones "manualmente". El código principal del programa es idéntico para ambos casos, con la diferencia de que en la versión simple se llama a las funciones nativas y en la versión avanzada a las programadas manualmente. Por tanto, se explicará primero el funcionamiento del programa en general, y luego la forma en la que se han programado las funciones nativas de MATLAB. Las funciones programadas llevan el mismo nombre que las nativas + Propia.m.

Archivos MAIN_pr2.m y MAIN_pr2_avanzado.m

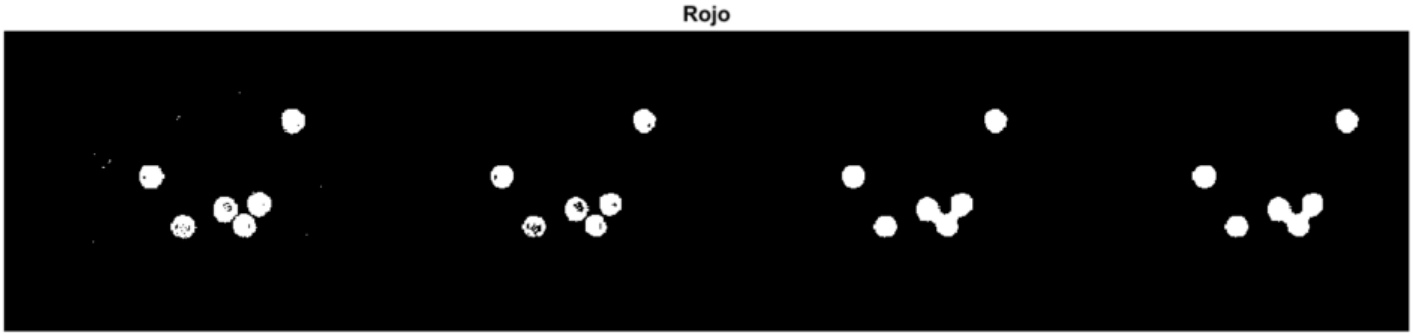
Se comienza leyendo la imagen a tratar, y convirtiéndola al espacio de colores HSV, sobre el que se trabajará durante la práctica. Se realiza la separación en canales, y se crea la matriz de umbrales de cada canal, obtenidos manualmente mediante la herramienta colorThresholder de MATLAB. Finalmente se crea un vector con los nombres de los colores que se van a analizar.

```
%% Sistemas de Percepcion - Practica 2 - Segmentacion por color
clear; clc; close all

% Leer la imagen
fOriginal = imread('imagenDePartida.png');
fHSV = rgb2hsv(fOriginal);

%Separar por canales
fH = fHSV( :,:,1); % canal hue
fS = fHSV( :,:,2); % canal saturation
fV = fHSV( :,:,3); % canal value



% Obtener umbrales HSV a traves de aplicacion colorThresholder
umbralesHSV = [ 0.0340, 0.9470, 1.0000, 0.4010, 1.0000, 0.5420;
                0.0820, 0.0200, 1.0000, 0.4140, 1.0000, 0.8340;
                0.3770, 0.2710, 1.0000, 0.1910, 1.0000, 0.6120;
                0.6110, 0.5090, 1.0000, 0.1910, 1.0000, 0.4700;
                0.1890, 0.1530, 1.0000, 0.4830, 1.0000, 0.6670;
                0.9850, 0,      1.0000, 0,      0.3050, 0.0980];

nombre_colores = ["Rojo", "Naranja", "Verde", "Azul", "Amarillo", "Negro"];
```

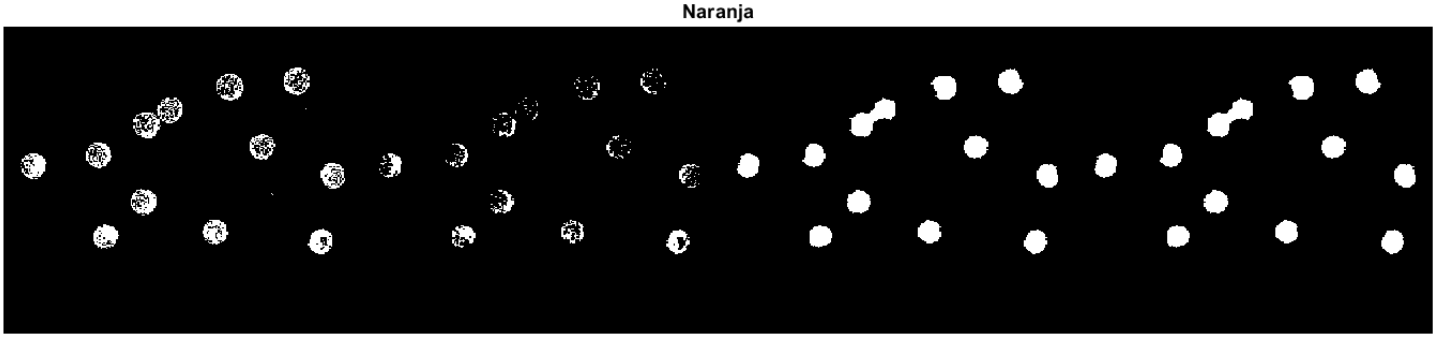
A continuación, se realiza un bucle for de 6 iteraciones, en el que se acota la imagen original para cada color que se quiere segmentar. Esta segmentación tiene en cuenta que puede darse el caso de que el umbral mínimo del canal "Hue" sea mayor que el umbral máximo del mismo. Una vez segmentada la imagen, se realizan una serie de operaciones morfológicas con el objetivo de eliminar el ruido creado al segmentar la imagen, y finalmente se llama a la función "f1_calculo" o "f1_calculo_avanzada" según se trate de una versión u otra. Las operaciones morfológicas con nombre *Propia son las programadas manualmente. En la versión simple se realiza el mismo orden de llamadas a las funciones nativas.

```
for i = 1:6
    % El siguiente if comprueba que no se trate del caso en que el umbral
    % minimo es mayor que el umbral maximo en el campo Hue.
    if (umbralesHSV(i,1) < umbralesHSV(i,2))
        fTratada =  (fH < umbralesHSV(i, 1) | fH > umbralesHSV(i, 2)) &...
                    (fS < umbralesHSV(i, 3) & fS > umbralesHSV(i, 4)) &...
                    (fV < umbralesHSV(i, 5) & fV > umbralesHSV(i, 6));
    else
        fTratada =  (fH < umbralesHSV(i, 1) & fH > umbralesHSV(i, 2)) &...
                    (fS < umbralesHSV(i, 3) & fS > umbralesHSV(i, 4)) &...
                    (fV < umbralesHSV(i, 5) & fV > umbralesHSV(i, 6));
    end

    % Una vez obtenida la mascara en blanco y negro, aplicar procesamiento
    % Realizar una erosión con plantilla circular de radio 1
    elem = strelPropia(1);
    fTratada = imerodePropia(fTratada, elem);

    % Realizar un cierre con plantilla circular de radio 8
    elem = strelPropia(8);
    fTratada = imclosePropia(fTratada, elem);

    % Realizar una apertura con plantilla circular de radio 1
    elem = strelPropia(1);
    fTratada = imopenPropia(fTratada, elem);

    % Una vez se tiene una imagen binaria con los objetos separados en
    % mayor o menor medida, se pasa a tratar la imagen para dibujar las
    % bounding boxes y los centros de masa de cada objeto.
    f1_calculo_avanzada(fTratada, fOriginal, i, nombre_colores);

%     pause();
end
```

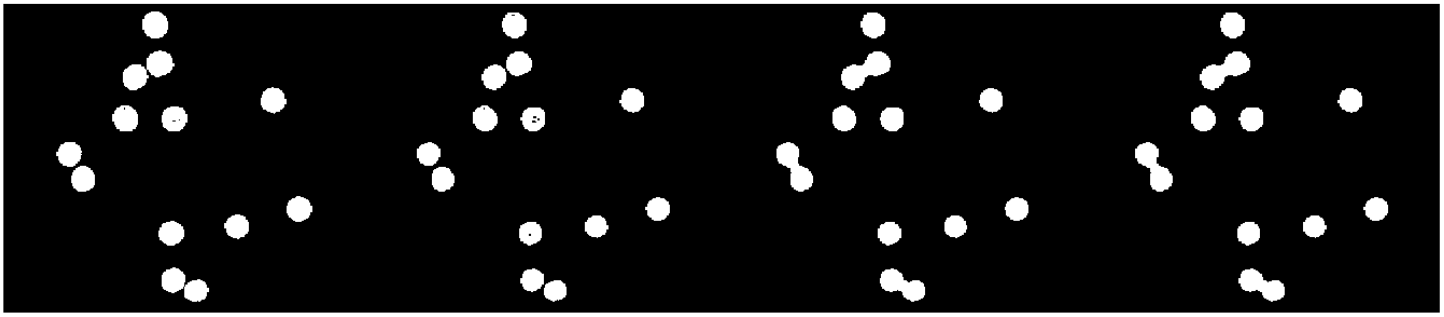
A continuación, se muestra la evolución de “fTratada” a lo largo del proceso de operaciones morfológicas, para cada color, utilizando la versión avanzada. Se muestra la imagen segmentada, la imagen después de realizar una erosión con una plantilla de radio 1, un cierre de radio 8 y una apertura de radio 1.



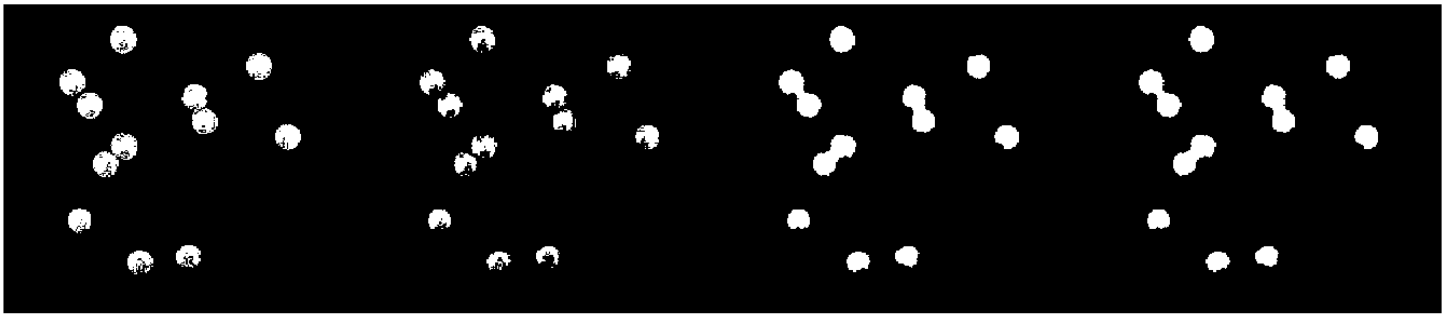
Naranja



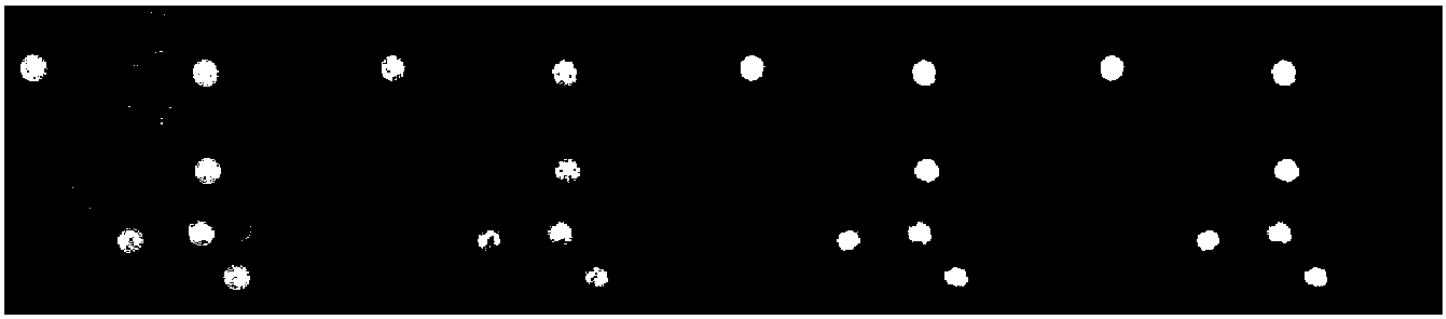
Verde



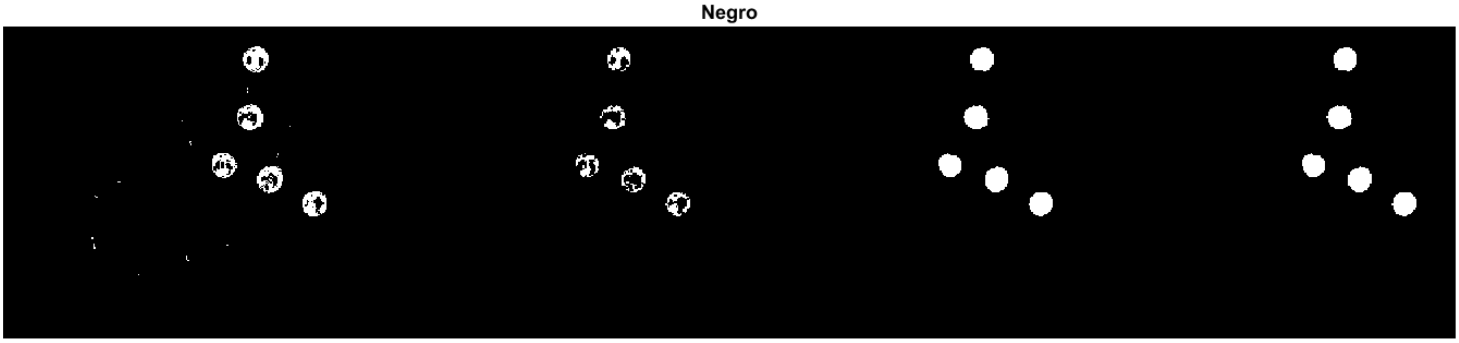
Azul



Amarillo



Negro



Archivos f1_calculo.m y f1_calculo_avanzada.m

Nuevamente ambos códigos son idénticos, con la excepción de llamadas a funciones nativas o programadas. Reciben la imagen ya tratada con las operaciones morfológicas, la imagen original, el índice del bucle for desde el que es llamada la función, y el array con los nombres de los colores, para el título de las imágenes. Se comienza etiquetando la imagen tratada, y obteniendo los datos de la imagen etiquetada con la función "regionprops" (o "regionpropsPropia" en la versión avanzada).

```
function f1_calculo_avanzada(fTratada, fOriginal, indice_plot, nombre_colores)
%f1_calculo: Funcion que realiza los calculos necesarios para obtener los
%centros de masa de los objetos a individualizar.

% Radio de los objetos, en pixeles
radioMM = 18;

% Convertir la imagen de entrada a imagen logica.
fEtiquetada = bwlabel(fTratada);

% Preparar subplot para mostrar cada color por separado
figure(2);                                  subplot(2,                                  3,
indice_plot);imshow(fOriginal,[]);title(nombre_colores(indice_plot));

% Obtener los datos de area, perimetro y bounding box de cada objeto.
datos = regionpropsPropia(fEtiquetada);
```

Una vez se tienen los datos, se realiza un bucle for que recorre cada uno esos objetos individualizados, comprueba si en efecto se individualizó correctamente (mediante el área devuelta por la función regionprops, y opera según el caso.

```
for i = 1:length(datos)
    areaObj = datos(i).Area;
    boundingboxObj = round(datos(i).BoundingBox);
    centroObj = datos(i).Centroid;
```

En caso de que se trate de más de un objeto marcado como uno único, se procede calcular cuántos objetos hay realmente, dividiendo el área en píxeles por 1000 (área aproximada de un único objeto), y redondeando. Se obtienen los valores de la bounding box "errónea", y se realiza una serie de erosiones consecutivas hasta que el número de objetos devuelta por la función regionprops se corresponde con el calculado a través del área.

```
    % Comprobar que se trata de un unico objeto, y no la union de varios
    % que no se han podido individualizar. Se utiliza el area, pero podria
    % utilizarse la circularidad por ejemplo.
    if (areaObj > 1500)
        % Si se trata de mas de un objeto, se procede a realizar
        % operaciones de erosion hasta conseguir individualizar los
        % objetos.

        % Se calcula el numero de objetos que hay en el segmento mal
        % individualizado
        numObjetos = round(areaObj / 1000);

        % Obtener valores de bounding boxes, pasados a filas/columnas de la
        % imagen tratada (filas se corresponde con la Y, columnas con la X)
        pFila_inicio = boundingboxObj(2);
        pFila_final = boundingboxObj(2) + boundingboxObj(4);
        pColumna_inicio = boundingboxObj(1);
        pColumna_final = boundingboxObj(1) + boundingboxObj(3);

        % Recortar la imagen para aplicar el procesamiento solo en la parte
        % fallida. A partir de ahora todos los valores seran "locales" a la
        % seccion fallida, por lo que se necesitara reconvertirlos a
        % coordenadas "globales".
        fRecortada    =    fTratada(pFila_inicio-2:pFila_final+2,    pColumna_inicio-
2:pColumna_final+2);% Se le da un pequeno margen para evitar que los bordes esten a 1.
        datosIndividualizados = regionpropsPropia(fRecortada);
        elem = strelPropia(1);

        % Se hace erosion con un disco de 1 de radio. Mientras que el numero
        % que devuelve regionprops es menor al calculado de objetos,
        % continuar iterando erosiones hasta cumplir la condicion.
        while(length(datosIndividualizados) < numObjetos)
            fRecortada = imerodePropia(fRecortada, elem);
            fEtiq = bwlabel(fRecortada);
            datosIndividualizados = regionpropsPropia(fEtiq);
        end
```

Una vez individualizados, se dibuja la bounding box de cada objeto ya correctamente individualizado, teniendo en cuenta el centro de gravedad devuelto por regionprops y el radio aproximado de un único objeto.

```
        % Una vez se tienen individualizados, se procede a dibujar su
        % bounding box y centroide.
        for j = 1:length(datosIndividualizados)
            centroX = round(pColumna_inicio + datosIndividualizados(j).Centroid(1));
            centroY = round(pFila_inicio + datosIndividualizados(j).Centroid(2));
            line([centroX           centroX],           [centroY           centroY],
'Marker','o','MarkerFaceColor','b','MarkerSize',5,'MarkerEdgeColor','w');

            % Obtener los valores de las bounding boxes
            pFila_inicioInd = datosIndividualizados(j).Centroid(2) - radioMM;
            pFila_finalInd = datosIndividualizados(j).Centroid(2) + radioMM;
            pColumna_inicioInd = datosIndividualizados(j).Centroid(1) - radioMM;
            pColumna_finalInd = datosIndividualizados(j).Centroid(1) + radioMM;

            % Devolver a coordenadas "globales" sumando el inicio de la
            % bounding box que se utilizo para recortar la imagen
            pFila_inicioInd = pFila_inicioInd + pFila_inicio;
            pFila_finalInd = pFila_finalInd + pFila_inicio;
            pColumna_inicioInd = pColumna_inicioInd + pColumna_inicio;
            pColumna_finalInd = pColumna_finalInd + pColumna_inicio;
```

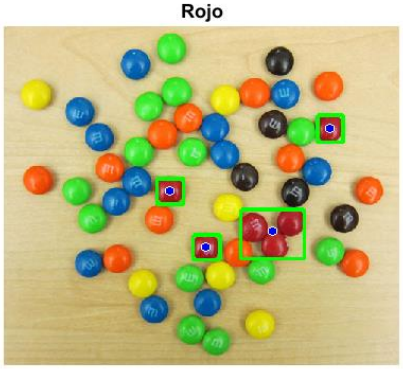
```
            % Dibujar cada bounding box
            line([pColumna_inicioInd        pColumna_finalInd        pColumna_finalInd
pColumna_inicioInd      pColumna_inicioInd],      [pFila_inicioInd      pFila_inicioInd
pFila_finalInd pFila_finalInd pFila_inicioInd],...
                'Color','cyan','LineWidth', 2);

            % Dibujar ademas el bounding box "erroneo"
            line([pColumna_inicio    pColumna_final    pColumna_final    pColumna_inicio
pColumna_inicio], [pFila_inicio pFila_inicio pFila_final pFila_final pFila_inicio],...
                'Color','yellow','LineWidth', 2);
        end
```

Por otro lado, en caso de que el área del objeto se corresponda con aquella de un único objeto, se dibuja directamente los centros de gravedad y las bounding box devueltas por regionprops.

```
    else
        % En caso de que se trate de un unico objeto bien individualizado,
        % dibujar el centro de masa y su bounding box.
        % Obtener valores de bounding boxes, pasados a filas/columnas de la
        % imagen tratada (filas se corresponde con la Y, columnas con la X)
        pFila_inicio = boundingboxObj(2);
        pFila_final = boundingboxObj(2) + boundingboxObj(4);
        pColumna_inicio = boundingboxObj(1);
        pColumna_final = boundingboxObj(1) + boundingboxObj(3);
        centroFila = centroObj(1);
        centroColum = centroObj(2);

        % Dibujar cada bounding box
        line([pColumna_inicio     pColumna_final     pColumna_final     pColumna_inicio
pColumna_inicio], [pFila_inicio pFila_inicio pFila_final pFila_final pFila_inicio],...
            'Color','green','LineWidth', 2);
        % Dibujar centro
        line([centroFila        centroFila],        [centroColum        centroColum],
'Marker','o','MarkerFaceColor','b','MarkerSize',5,'MarkerEdgeColor','w');
    end
end
end
```

La única diferencia entre los dos archivos es la llamada a regionprops o regionpropsPropia en distintos puntos, y a imerode o imerodePropia en el bucle while que comprueba el número de objetos individualizados. En ambas versiones se aplica el procesamiento que individualiza correctamente los objetos, a pesar de que para la versión simple no se pedía. Bastaría con aumentar el valor de la comprobación “if (areaObj > 1500)” a un valor muy alto para obtener la versión simple pedida (con areaObj > 15000 es suficiente), aunque pintando la bounding box de verde en lugar de amarillo.

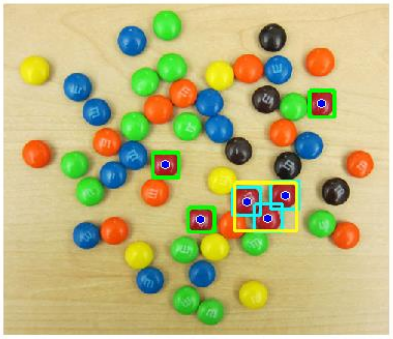
Rojo



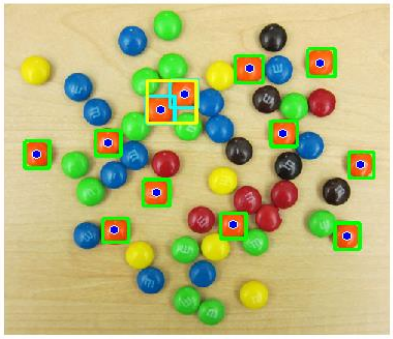
Resultados

Versión simple

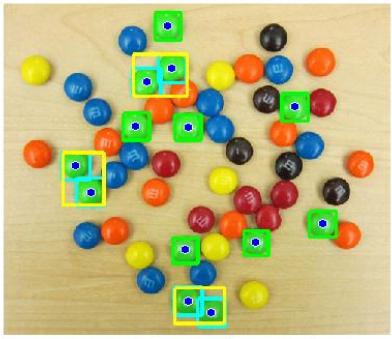
Rojo



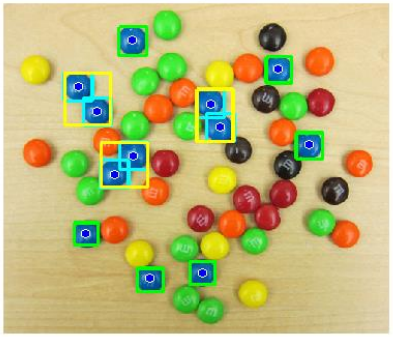
Naranja



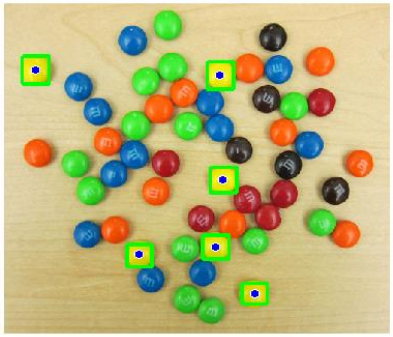
Verde



Azul



Amarillo

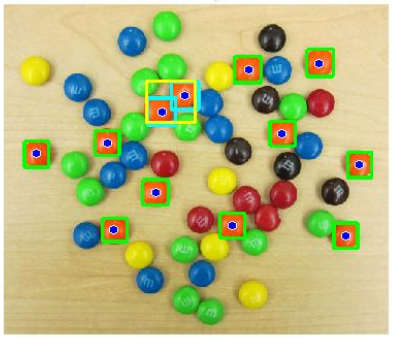


Negro

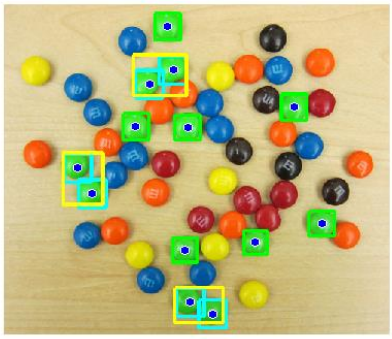
Versión avanzada

Rojo

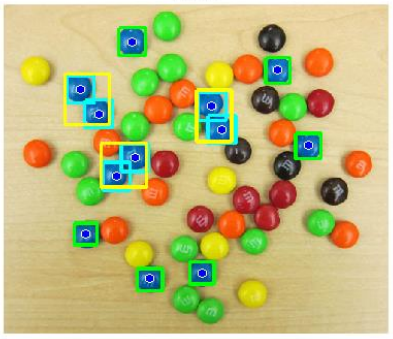
Naranja



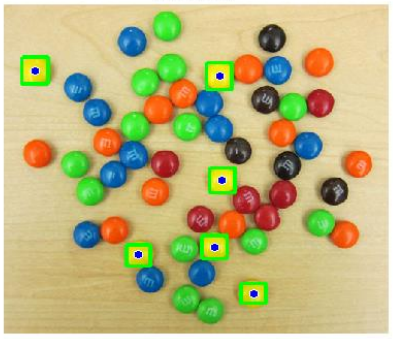
Verde



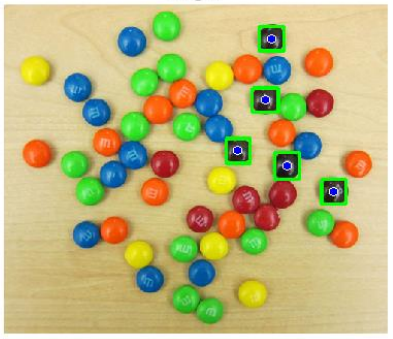
Azul



Amarillo



Negro



Anexo: funciones programadas manualmente

Función strelPropia.m

Función que sustituye a strel. En este caso solo se contempla la creación de plantillas de tipo "disk". Para ello, se crea una matriz del tamaño deseado, se comprueba si cada elemento de la matriz se encuentra dentro del radio pedido, y se sitúan a 0 o a 1 cada valor según corresponda.

```
function matrix = strelPropia(radio)
%STREL_PROP Crea una plantilla de tipo 'disk' para operaciones morfológicas
%   Se crea una matriz de tipo circular del radio indicado. El tamaño de la
%   matriz viene dado por el radio introducido, más la fila/columna
%   central. Por tanto un radio de 1 creará una matriz de 3x3.



% Variables de número de filas y columnas
nfila = 2*radio + 1;
ncolum = nfila;

% Variable que indica el centro de la matriz
centro = radio + 1;

% Crear matriz y rellenarla en forma de disco. Esto se realiza calculando
% el módulo de la distancia entre el elemento que se quiere definir y el
% centro. Si dicho módulo es menor o igual al radio, el elemento será igual
% a 1. En caso contrario será cero.
matrix = zeros(nfila,ncolum);

for i=1:nfila
    for j=1:ncolum
        dist_x = centro-j;  % Distancia en x, centro - ncolum_actual
        dist_y = centro-i;  % Distancia en y, centro - nfila_actual
        dist_total = sqrt(dist_x^2+dist_y^2);   % Módulo de la distancia
        if dist_total<=radio
            matrix(i,j) = 1;
        end
    end
end
end
```

Función imerodePropia.m

Función que sustituye a imerode. Recibe el elemento creado con strelPropia y la imagen binarizada sobre la que se quiere realizar la operación morfológica. Se comienza obteniendo el número de filas y columnas tanto de la imagen a tratar como de la plantilla que se utilizará.

```
function fTratar = imerodePropia(fTratar, plantilla)
%imerodePropia Funcion que realiza la misma funcionalidad que imerode.
%   Recibe la imagen a tratar y el elemento "strel" de tipo disco con el
%   que se realizara la operacion morfologica.

% Copiar la matriz original para no reescribirla
fAux = fTratar;

% Obtener tamano de la imagen a tratar
[nFilas, nColumnas, ~] = size(fTratar);

% Obtener radio de la plantilla.
nFilasPlantilla = length(plantilla(:,1));
nColumnasPlantilla = nFilasPlantilla;
radioPlantilla = (nFilasPlantilla - 1) / 2;
```

Se lee la plantilla entera y se van almacenando en una matriz las coordenadas para las que la plantilla es igual a 1, buscando disminuir la carga computacional más adelante, dejando de ser necesario recorrer la plantilla entera.

```
% Para evitar comprobar con aquellos valores iguales a 0, se hara un
% pre-procesado de la plantilla, y se almacenan los pares de valores XY
% para los que la plantilla es igual a 1. El vector "aComprobar" contendra
% aquellos valores de fila/columna en los que la plantilla es un 1.
aComprobar = [];
indice = 1;
for i = 1:nFilasPlantilla
    for j = 1:nColumnasPlantilla
        if plantilla(j, i) == 1
            aComprobar(indice, 1) = j;
            aComprobar(indice, 2) = i;
            indice = indice + 1;
        end
    end
end
tamComprobar = indice - 1;
```

El método que se utiliza coge un píxel de la imagen, y calcula el pixel superior izquierdo sobre el que se situaría el comienzo de la plantilla, y a partir de ahí se sumará el valor almacenado en cada fila de la matriz de píxeles a comprobar, suma las coordenadas al píxel correspondiente, y comprueba si tiene un valor igual a 1, en cuyo caso aumenta el contador de aciertos, comprobando al final si el valor de ese contador es igual al número de coordenadas que se deben comprobar, lo que indica que se cumple la condición para poder mantener el píxel a 1. Cabe destacar que si la suma de las coordenadas a comprobar con el píxel “origen” resulta caer fuera de los límites de la imagen se incrementa inmediatamente el contador, permitiendo así aplicar la plantilla parcialmente en los bordes de la imagen.

```
for filas = 1 : nFilas
    for columnas = 1 : nColumnas
        % Se coge la esquina superior izquierda de la plantilla superpuesta
        % al pixel que se comprueba.
        filaOrigen = filas - radioPlantilla - 1;
        columOrigen = columnas - radioPlantilla - 1;

        % Inicializar contador de aciertos
        contadorAciertos = 0;

        % Tratar unicamente aquellos pixeles que estan a 1
        if fTratar(filas, columnas) ~= 0

            % Se recorre el vector "aComprobar"
            for indComprobar = 1:tamComprobar
                % Obtener el valor que habra que sumar al pixel superior
                % izquierdo obtenido en "filaOrigen" y "columOrigen".
                sumFila = aComprobar(indComprobar, 1);
                sumColum = aComprobar(indComprobar, 2);

                % Comprobar si el valor del pixel es 1, y aumentar el contador
                % en caso afirmativo. Si no es igual a 1, salir del bucle de
                % comprobacion.
                if (filaOrigen + sumFila > 0) && (filaOrigen + sumFila <= nFilas) ...
                       && (columOrigen + sumColum > 0) && (columOrigen + sumColum <=
nColumnas)
                    if fTratar(filaOrigen + sumFila, columOrigen + sumColum) == 1
```

```
                        contadorAciertos = contadorAciertos + 1;
                    else
                        break
                    end
                else
                    contadorAciertos = contadorAciertos + 1;
                end
            end

            % Dar valor al pixel "padre", en funcion del valor del contador.
            if contadorAciertos == tamComprobar
                fAux(filas, columnas) = 1;
            else
                fAux(filas, columnas) = 0;
            end
        end
    end
end
fTratar = fAux;

end
```

Función imdilatePropia.m

En este caso se sigue un procedimiento parecido a imerodePropia. Se crea la matriz tras recorrer la plantilla en su totalidad, y se recorre cada píxel de la imagen realizando la misma operación. En este caso no es necesario llevar un contador de los aciertos, puesto que se ponen a 1 aquellos píxeles que en la plantilla estén a 1.

```
function fTratar = imdilatePropia(fTratar, plantilla)
%imdilatePropia Funcion que realiza la misma funcionalidad que imdilate.
%   Recibe la imagen a tratar y el elemento "strel" de tipo disco con el
%   que se realizara la operacion morfologica.

% Copiar la imagen a tratar, para no sobreescribirla.
fAux = fTratar;

% Obtener tamano de la imagen a tratar
[nFilas, nColumnas, ~] = size(fTratar);

% Obtener radio de la plantilla.
nFilasPlantilla = length(plantilla(1,:));
nColumnasPlantilla = nFilasPlantilla;
radioPlantilla =  (nFilasPlantilla - 1) / 2;

% Para evitar comprobar con aquellos valores iguales a 0, se hara un
% pre-procesado de la plantilla, y se almacenan los pares de valores XY
% para los que la plantilla es igual a 1. El vector "aComprobar" contendra
% aquellos valores de fila/columna en los que la plantilla es un 1.
aComprobar = [];
indice = 1;
for i = 1:nFilasPlantilla
    for j = 1:nColumnasPlantilla
        if plantilla(j, i) == 1
            aComprobar(indice, 1) = j;
            aComprobar(indice, 2) = i;
            indice = indice + 1;
        end
    end
end
tamComprobar = indice - 1;
```

```
% Se comprueba en aquellos pixeles que esten a mas de un radio de distancia
% del borde
for filas = 1 : nFilas
    for columnas = 1 : nColumnas
        % Se coge la esquina superior izquierda de la plantilla superpuesta
        % al pixel que se comprueba.
        filaOrigen = filas - radioPlantilla - 1;
        columOrigen = columnas - radioPlantilla - 1;
        if fTratar(filas, columnas) == 1
            % Se recorre el vector "aComprobar"
            for indComprobar = 1:tamComprobar
                % Obtener el valor que habrá que sumar al pixel superior
                % izquierdo obtenido en "filaOrigen" y "columOrigen".
                sumFila = aComprobar(indComprobar, 1);
                sumColum = aComprobar(indComprobar, 2);

                if (filaOrigen + sumFila > 0) && (filaOrigen + sumFila <= nFilas) ...
                       && (columOrigen + sumColum > 0) && (columOrigen + sumColum <=
nColumnas)



                    fAux(filaOrigen + sumFila, columOrigen + sumColum) = 1;
                end
            end
        end

    end
end
fTratar = fAux;

end
```

Funciones imopenPropia.m e imclosePropia.m

En estas funciones lo único que se hace es llamar a las funciones imerodePropia e imdilatePropia en el orden requerido para realizar cada operación morfológica.

```
function fTratar = imclosePropia(fTratar, plantilla)
%imclosePropia Funcion que realiza la misma funcionalidad que imclose.
%   Recibe la imagen a tratar y el elemento "strel" de tipo disco con el
%   que se realizara la operacion morfologica.

% Primero se realiza un dilatado y posteriormente una erosion
fTratar = imdilatePropia(fTratar, plantilla);

fTratar = imerodePropia(fTratar, plantilla);

end
```

```
function fTratar = imopenPropia(fTratar, plantilla)
%imopenPropia Funcion que realiza la misma funcionalidad que imopen.
%   Recibe la imagen a tratar y el elemento "strel" de tipo disco con el
%   que se realizara la operacion morfologica.

% Primero se realiza una erosion y posteriormente un dilatado
fTratar = imerodePropia(fTratar, plantilla);

fTratar = imdilatePropia(fTratar, plantilla);

end
```

Función regionpropsPropia.m

Función que sustituye a regionprops. Devuelve un vector de estructuras que contienen el área, centroide y bounding box de cada uno de los elementos disponibles en la imagen etiquetada. Se realiza un bucle for que recorre cada una de las etiquetas y calcula el área, los momentos m01 y m10, el centroide, y los límites de la bounding box, y los devuelve con el mismo formato que la función nativa de MATLAB.

```
function estructuraRP = regionpropsPropia(fTratada)
%regionpropsPropia Funcion que realiza parte de la funcionalidad de la
%funcion nativa de MATLAB "regionprops".

% Obtener tamano de la imagen a tratar
[nFilas, nColumnas, ~] = size(fTratada);

% Recibe la imagen etiquetada con cada objeto individualizado.
nEtiquetas = double(max(max(fTratada)));

for i = 1:nEtiquetas
    fAux = (fTratada == i);     % Se escoge cada etiqueta por separado
    m00 = sum(sum(fAux));       % Momento de orden 0, area

    % Variables para los maximos y minimos de la bounding box
    bbXmin = nColumnas; bbXmax = 0;
    bbYmin = nFilas;    bbYmax = 0;

    % Calculo de m01 y m10
    m01 = 0; m10 = 0;
    for x = 1 : nColumnas
        for y = 1 : nFilas

            % Si el pixel esta a 1, comprobar maximos y minimos
            if (fAux(y, x) == 1)
                if y < bbYmin
                    bbYmin = y;
                elseif y > bbYmax
                    bbYmax = y;
                end
                if x < bbXmin
                    bbXmin = x;
                elseif x > bbXmax
                    bbXmax = x;
                end
            end

            m10 = m10 + x * fAux(y, x); % Momento en que r = 1, s = 0
            m01 = m01 + y * fAux(y, x); % Momento en que r = 0, s = 1
        end
    end

    % Una vez se tienen los momentos m10 y m01 se puede calcular el centro
    % de gravedad.
    centroGravX = round(m10 / m00);
    centroGravY = round(m01 / m00);

    % Para calcular los limites de la bounding box basta con comprobar el
    % valor minimo y maximo en cada imagen.
    estructuraRP(i).BoundingBox = [bbXmin bbYmin bbXmax-bbXmin bbYmax-bbYmin];
    estructuraRP(i).Area = m00;
    estructuraRP(i).Centroid = [centroGravX centroGravY];
end
end
```