Desafíos 2, 3 y 4

Mikel Dalmau

4 de Octubre de 2018

Contents

1 Enunciados

1.1 Desafío 2

El objetivo es crear una tabla de caraceristicas geometricas de las imagenes utilizando la herramienta de matlab regionprops.

- ¿es factible compararlos y distinguirlos?
- ¿que caracteristicas parecen ser más distitivas?

1.2 Desafío 3

El objetivo es crear una tabla de caraceristicas geometricas de las imagenes corrigiendo los problemas que plantean las ditintas orientaciones utilizando las herramientas de matlab regionprops, imrotate.

1.3 Desafío 4

El objetivo es asegurar que todas y cada una de las imágenes solo tienen un componente sin agujeros.

Año: 2018 - 2019
Profesor(a): Manuel Graña
Teléfono:
E-mail: mdalmau002@ikasle.ehu.es

GIPUZKOAKO CAMPUSA
CAMPUS DE GUIPUZCOA
Paseo Manuel de Lardizabal, 1
20018 Donostia (Gipuzkoa)

2 Procesado de las imágenes

Para procesar las imágenes y extraer las propiedades de estas existen una serie de pasos obligatorios requeridos para preparar la imagen y que esta se pueda evaluar por **regionprops**. Luego de cara a conseguir mejores resultados he introducido pasos intermedios adicionales.

El pipeline de las imágenes está definido de la siguiente manera.

1. Leer la imagen
2. Transformar a escala de grises
3. Operaciones sobre escala de grises (opcional)
4. Binarizar
5. Operaciones sobre imagen binarizada (opcional)
6. Calcular propiedades **regionprops**
7. Otras operaciones utilizando las propiedades (opcional)

Al final de cada paso, voy guardando las imágenes intermedias y termino el proceso entregando para cada imágen, todas sus intermedias y la estructura de propiedades calculada una vez rotada la imagen.

Este es el código de la función que realiza este proceso, recibe como parámetros el nombre de la imagen a procesar y algunos parámetros que influyen en otros pasos del proceso explicados más adelante.

```
%% Image Processing PIPE-LINE
%
%
function [propertiesR, step] = properties(image, sigma, threshold,
    connectedComponentMinPixels, rotate)

    % Read image
    original = imread(image);
    step(1)= {original};

    % Transform to greyscale
    greyScale = rgb2gray(original);
    step(2)= {greyScale};

    % Apply before binarization operations
    toBinarize = onBeforeBinarization(greyScale, sigma);
    step(3)= {toBinarize};

    % binarize image
    binary = binarize(toBinarize,threshold);
    step(4)= {binary};

    % Apply after binarization operations
    toComplement = onAfterBinarization(binary,connectedComponentMinPixels);
    step(5)= {toComplement};
```

```
    % Get the complement of the binary image to use in regionprops
    complement = imcomplement(toComplement);
    step(6)= {complement};

    % Apply functions to complement image
    afterComplement = onAfterComplement(complement);
    step(7) = {afterComplement};

    % RETO 3 Rotate images
    rotated = VRotate(afterComplement, rotate);
    step(8)= {rotated};

    % Calculate properties from the rotated image
    propertiesR = pickGreatestCC(regionprops(rotated,'centroid','BoundingBox','
        Area', 'Orientation','Perimeter','Extrema','ConvexHull','Eccentricity','
        MajorAxisLength','MinorAxisLength','Solidity'));

end
```

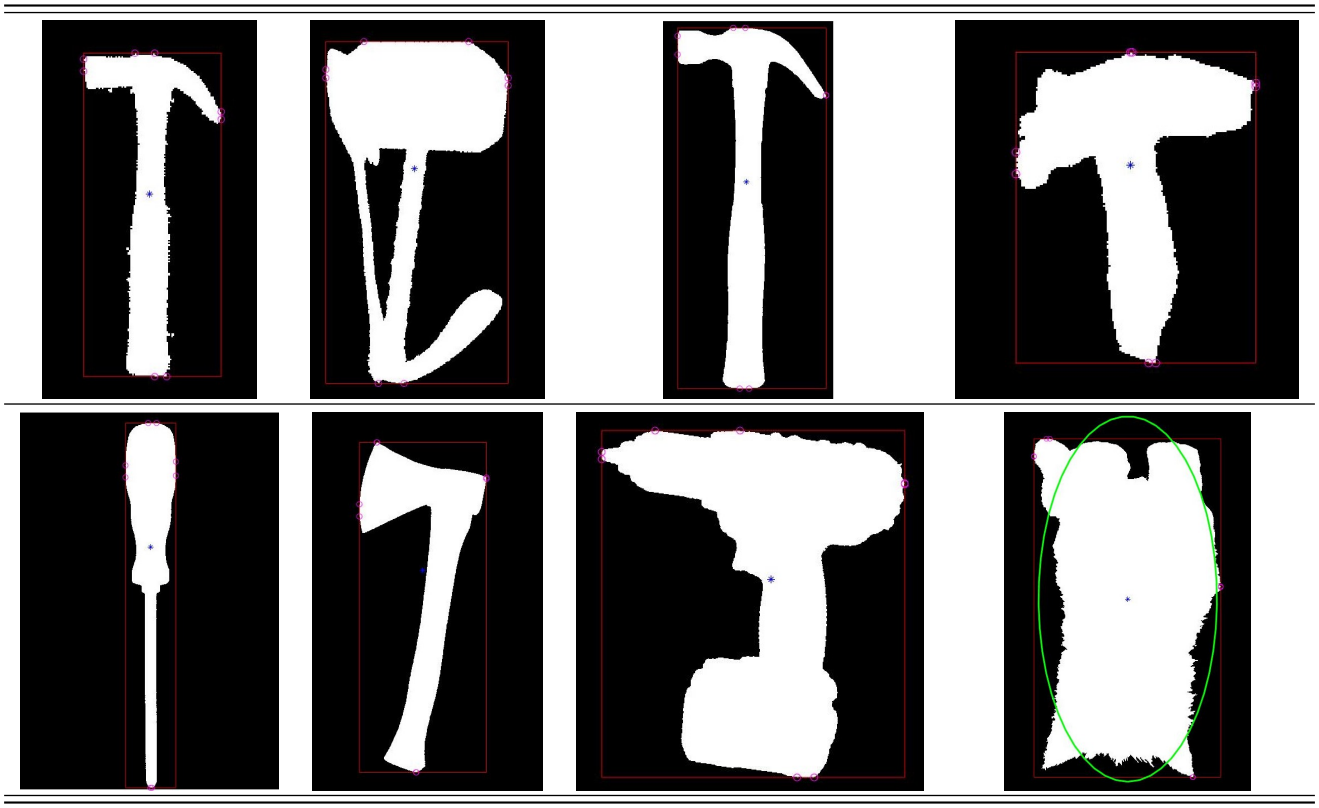
Un bucle lee los nombres de las imágenes en una carpeta y procesa todas guardando los resultados en una matriz de celdas compuesta por { Nombre, propiedades, imagenes intermedias, es martillo? } de la siguiente manera.

```
% Read Hammers
dirlist = dir(folder+'\martillo*');
hammers = length(dirlist);
for i = 1:hammers
    [propertiesR, step] = properties(dirlist(i).name, sigma, threshold,
        connectedComponentMinPixels, rotate);
    cells(i,1:4) = {dirlist(i).name,propertiesR, step, true};
end

% Read Others
dirlist = cat(1,dir(folder+'\gato*'),dir(folder+'\taladro*'),dir(folder+'\llave*
    '),dir(folder+'\hacha*'),dir(folder+'\destorni*'));
for i = 1:length(dirlist)
    [propertiesR, step] = properties(dirlist(i).name, sigma, threshold,
        connectedComponentMinPixels, rotate);
    cells(i+hammers,1:4) = {dirlist(i).name,propertiesR, step, false};
end

% Extract image properties
m = propertyExtractor(cells);
```

Algunas imágenes procesadas de martillos y de otras cosas



2.1 Binarizar la imagen

A la hora de binarizar la imagen existen diferentes métodos disponibles en Matlab, uno de ellos consiste en utilizar la función **im2bw (I, T)** siento **I** la imagen a escala de grises y **T** el umbral de binarización como un valor entre 0 y 1.

Por otro lado, existe también la función **imbinarize**, que dispone de más configuraciones, tales como elegir si la binarización es global donde utiliza el metodo de Otsu para determinar el umbral o es adaptativa (tiene en cuenta el color del fondo para varía el umbral a lo largo de la imagen).

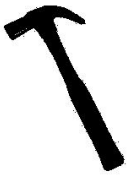
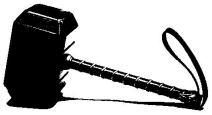
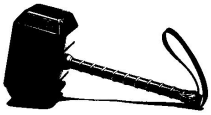
La siguiente función de matlab es la que he utilizado y computa una u otra binarización dependiendo del valor del parámetro T.

```
%  I - GreyScale image to binarize
%  T -Threshold or negative for Otsus
function binary = binarize(I,T)

    if T >= 0
        binary = im2bw(I,T);
    else
        binary = imbinarize(I);
    end
end
```

Finalmente me he decantado por la binarización simple con un umbral de 0.95 ya que los martillos están sobre fondo blanco y esto simplifica las cosas aunque algunas imágenes tienen mejores resultados con umbral

de Otsu.

Nombre	gay-scale	Otsus	$T_{0.7}$	$T_{0.95}$
martillo1				
martillo12				
martillo4				
martillo9				

2.2 Procesos Previos a la binarización

Como proceso previo a la binarización he considerado utilizar algún filtro de desenfoque como el gaussiano para así reducir los picos de diferencias en la luz y logar imágenes más homogéneas de cara la binarización, aunque es interesante para eliminar las sombras en un par de imágenes, en muchas otras empeoran los resultados dividiendo las imágenes en varios grandes conjuntos conexos. Actualmente la función realiza o no un filtrado gaussiano dependiendo del valor del parámetro sigma.

```
% Functions applied before the binarization step, on greyscale image
%
%  GSI - Grey Scale Image
%  sigma - Factor for gaussian filtering 2, 4, 8
%
function toBinarize = onBeforeBinarization(GSI, sigma)
    toBinarize = GSI;

    if sigma > 0
        toBinarize = imgaussfilt(toBinarize, sigma);
    end
end
```

2.3 Proceso posterior a la binarización

Más interesante que el proceso antes de binarizar, es el que realizo después, y consiste en dos pasos, primero, erosiono la imagen utilizando la función **imerode** y pasándole como parámetro una matriz con los unos en forma de pequeño diamante como se puede ver en el siguiente código. Con esto consigo que la zona negra de la imagen se dilate y se vuelva más homogénea tapando agujeros blanco. En segundo lugar, utilizando la función **bwareaopen** que elimina los conjuntos conexos de pixels más pequeños que un parámetro, por prueba y error he decidido que 6500 pixels es un número adecuado.

```
% Functions applied before the binarization step, on greyscale % Functions
    applied after the binarization step
%
function toComplement = onAfterBinarization(BW, connectedComponentMinPixels)

    toComplement = BW;

    % Erode image to make black area greater, thus, filling the white holes
    % in the hammers
    SE = strel('diamond',1);
    toComplement = imerode(toComplement,SE);

    % Remove small pixels areas leaving 1 connected component if possible
    toComplement = bwareaopen(toComplement,connectedComponentMinPixels);

end
```

2.4 Rotar las imágenes

De cara a extraer propiedades significativas, como puede ser la posición de los puntos extremos, es interesante que todos los martillo o objetos estén orientados de la misma manera.

Para rotar los pasos que he seguido han sido, en primer lugar usando **regionprops** he calculado el **Area** y **Orientación** de cada imagen (El área la calculo siempre por seguridad, ya que siempre elijo la propiedad de área más grande en caso de haber más de un objeto en la imagen). Dada la orientación utilizo la función **imrotate** para girar la imágen $\theta = -90 - Orientacion$ grados, ya que la quiero vertical.

En segundo lugar, para evitar que alguno martillos se puedan quedar con la cabeza hacia abajo, calculo sus **Centroides**, **Bounding Box** y puntos **Extremos** y realizo dos consideraciones. Si el centroide o todos los puntos extremos excepto los dos superiores están en la mitad inferior está en la mitad inferior del bounding box giro la imagen original $theta + 180$ evitando así este problema y evitando así realizar giros consecutivos sobre la misma imagen.

```
% Computes the rotation of images with the following parameters:
%
% Depending on the position of the extrema points and the centroid, the
%       rotation might vary.
%
% angle - The angle seeked is -90 degrees
%
% bilinear - Bilinear interpolation; the output pixel value is a weighted
```

```
%               average of pixels in the nearest 2-by-2 neighborhood
%
function rotated = VRotate(I, rotate)

    if rotate == false
        rotated = I;
        return;
    end

    % Extracting the properties and take those of the conected component
    % with greatest area
    props = pickGreatestCC(regionprops(I,'Area','Orientation'));

    % We want theta + orientation to be equal to -90
    theta = -90 - props.Orientation;

    rotated = imrotate(I,theta,'bilinear','loose');

    % Compute regionprops again to get the actual centroid and bounding box,
    propertiesR = pickGreatestCC(regionprops(rotated,'centroid','BoundingBox','
        Area','Extrema'));

    if centroidInLowerHalf(propertiesR) || hammerHeadPosition(propertiesR,true
        ,2)
         theta = theta + 180;
         rotated = imrotate(I,theta,'bilinear','loose');
    end


end
```

```
% Condition 1, centroid in the lower half.
% if the centroid is in the lower half of the BB, then probably the
% hammer is upside down and should be rotated again a 180 degrees.
%
function cond1 = centroidInLowerHalf(propertiesR)

    y = propertiesR.BoundingBox(2);
    yw = propertiesR.BoundingBox(4);
    yc =  propertiesR.Centroid(2);

    cond1 = yc > (y+(yw/2));
end
```

```
% Condition 2,
% We want extrema points top-right top-left to be in the upper
% half and the rest of points in the bottom half, meaning the head of
% the hammer is upside down.
%
% [top-left top-right right-top right-bottom bottom-right bottom-left left-
    bottom left-top]
```

```
%
% propertiesR - Regionprops struct
% slices - Numer of pieces to divide area into
%
function cond2 = hammerHeadPosition(propertiesR,upsideDown,slices)

    y = propertiesR.BoundingBox(2);
    yw = propertiesR.BoundingBox(4);
    thresholdY = y+(yw/slices);

    e = propertiesR.Extrema;
    topLeftY = e(1,2);      topRightY = e(2,2);
    righTopY = e(3,2);       rightBottomY = e(4,2);
    bottomRightY = e(5,2);   bottomLeftY = e(6,2);
    leftBottomY = e(7,2);    leftTopY = e(8,2);

    if upsideDown
        % Only this 2 points are in the upper section
        cond21 = topLeftY < thresholdY ;
        cond22 = topRightY < thresholdY;

        % Majority of points are in the lower section
        cond23 = righTopY > thresholdY;
        cond24 = rightBottomY > thresholdY;
        cond25 = leftBottomY > thresholdY;
        cond26 = leftTopY > thresholdY;
        cond27 = bottomRightY > thresholdY;
        cond28 = bottomLeftY > thresholdY;

    else
         % Majority of points are in the lower section
        cond21 = topLeftY < thresholdY ;
        cond22 = topRightY < thresholdY;
        cond23 = righTopY < thresholdY;
        cond24 = rightBottomY < thresholdY;
        cond25 = leftBottomY < thresholdY;
        cond26 = leftTopY < thresholdY;

        % Only this 2 points are in the bottom section
        cond27 = bottomRightY > thresholdY;
        cond28 = bottomLeftY > thresholdY;
    end

    cond2 = cond21 && cond22 && cond23 && cond24 && cond25 && cond26 && cond27
       && cond28;
end
```

3 Clasificación

Una vez terminadas todos los procesos y calculadas las propiedades de todas las imágenes extraigo la siguiente serie de características de cara a utilizarlas en un algoritmo de clasificación en el futuro.

Me he cuidado de que las características sean independientes de las dimensiones de la imagen y me han parecido significativas las siguientes:

- Ratio altura - anchura del Bounding Box.
- Ratio entre el perímetro de la imagen y el del Bounding Box.
- Verdadero o falso si los puntos Extremos (todos exceptos los inferiores) se encuentran en el tercio superior, cuarto superior o quinto superior del Bounding Box.
- Relación entre el área de la imagen y la del Bounding Box.
- Excentricidad (Cuanto se sále la imagen del polígono convexo)
- Solidez (Relación entre el área y el del polígono convexo)

Finalmente, he construido una matriz con todos estos valores y el valor de la clase, 0 o 1 si es un martillo o no. Luego esta matriz puede ser importada a la aplicación de Matlab para clasificación y jugar con los distintos algoritmos de clasificación.

```
%% Classification Variables
%
%
%
%
function m=propertyExtractor(cells)

    for i=1:length(cells(:,1))

        isHammer = cells{i,4};

        props = cells{i,2};

        % Bounding Box Height Width Rate
        bb = props.BoundingBox;
        bbHeightWidthRate = bb(4)/bb(3);

        % Rate Between BB perimeter and real perimeter
        bbPerimeter = 2*(bb(4) + bb(3));
        perimetersRate = bbPerimeter/props.Perimeter;

        % Extrema points area
        headInUpperThird = hammerHeadPosition(props, false,3);

        headInUpperFourth = hammerHeadPosition(props, false,4);

        headInUpperFifth = hammerHeadPosition(props, false,5);

        % Relation Between BB Area and real Area
        areasRate = bb(4)*bb(3)/props.Area;

        % Eccentricity
        eccentricity = props.Eccentricity;
```

```
        % Solidity
        solidity = props.Solidity;

        m(i,:)= [bbHeightWidthRate, perimetersRate, headInUpperThird,
            headInUpperFourth, headInUpperFifth, areasRate, eccentricity, solidity,
            isHammer];
    end
end
```

Bibliografía

[1] Matlab official documentation:*es.mathworks.com*

[2]