

# Sistemas de Visión Artificial

## Práctica 4. Operaciones morfológicas y representación y descripción de imágenes

Nombre: Juan Casado Ballesteros

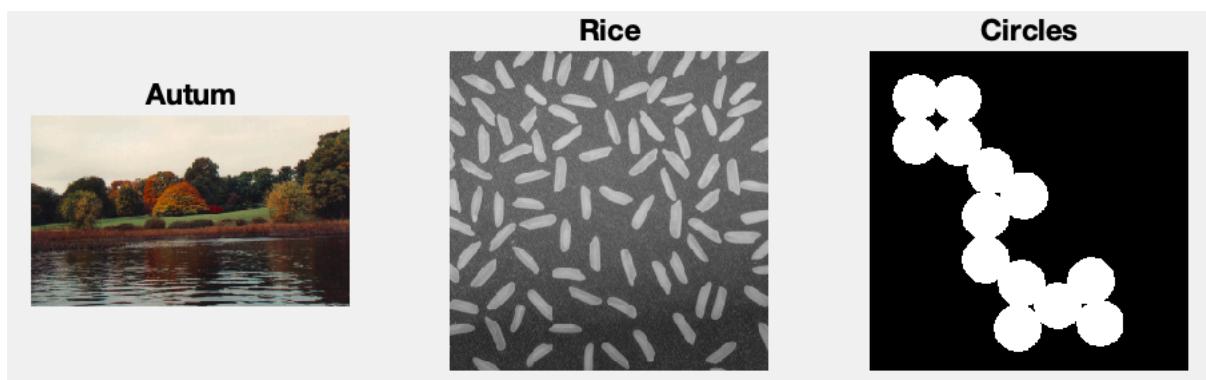
Fecha: 16 de noviembre 2019

### Operaciones morfológicas

#### Cargado de imágenes

Leemos las imágenes correspondientes de disco y las mostramos para poder comparar los resultados.

```
imgOriginalAutum = imread('autumn.tif');
imgOriginalRice = imread('rice.png');
imgOriginalCircles = imread('circles.png');
```



#### ¿Qué diferencias ve en el comportamiento de los operadores morfológicos para las distintas imágenes?

Creamos dos elementos estructurantes en forma de disco de radios 5 y 10, un radio moderado y otro más grande para poder apreciar mejor los efectos de su aplicación.

```
se = strel('disk', 5);
```

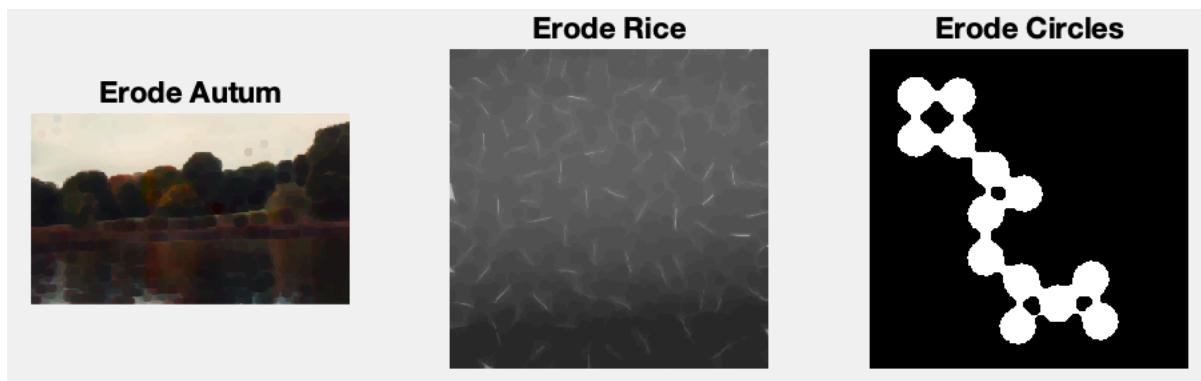
Aplicamos el elemento estructurante para erosionar las imágenes

```
erodeBWAutum = imerode(imgOriginalAutum, se);
erodeBWRice = imerode(imgOriginalRice, se);
erodeBWircles = imerode(imgOriginalCircles, se);
```

Con el elemento estructurante menor, a imagen Atum se oscurece, esto se debe a que aunque hay variedad de colores estos están mezclados con tonos oscuros que al ser erosionados aumentan la superficie que ocupan.

En el caso de Rice lo que logramos es hacer los granos de arroz mucho más finos concentrándolos en líneas a lo largo de su centro.

En la imagen Circles logramos algo similar, los círculos se separan unos de otros, quedando ahora solo unidos por delgadas líneas.



Con el elemento estructurante mayor los resultados se amplifican, la imagen Atum se ve mucho más oscura, en Rice los granos desaparecen completamente y en Circles los círculos se separan por completo.



Aplicamos el elemento estructurante para dilatar las imágenes

```
dilateBWAutum = imdilate(imgOriginalAutum,se);
dilateBWRice = imdilate(imgOriginalRice,se);
dilateBWircles = imdilate(imgOriginalCircles,se);
```

Con la dilatación logramos el efecto contrario, en Autum aumenta el brillo y los colores se difuminan como si la imagen no estuviera bien enfocada. En Rice el tamaño de los granos aumenta y estos se empiezan a juntar. Esta unión de elementos se aprecia también en Circles donde se ve más clara.



Con el elemento estructurante mayor los efectos se amplifican quedando colores y objetos de mayor brillo mucho más juntos.



La dilatación aumenta el tamaño de las superficies claras mientras que la erosión hace lo mismo con las oscuras.

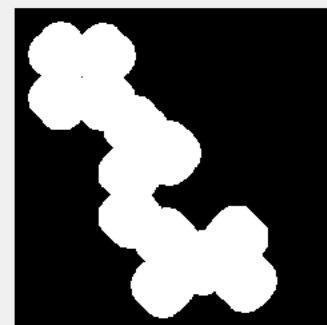
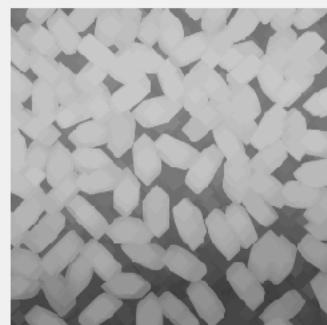
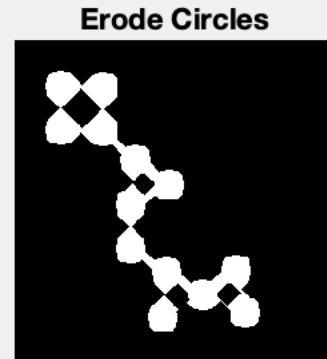
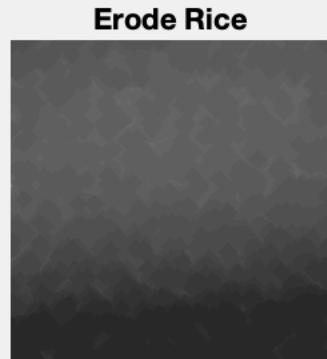
## Tipos de elementos estructurantes

Observaremos esta vez los efectos que producen los distintos elementos estructurantes que nos ofrece Matlab. Ya hemos comprobado su efecto al aumentar y reducir su tamaño por lo que nos centraremos ahora en lo que sucede cuando elegimos unos u otros.

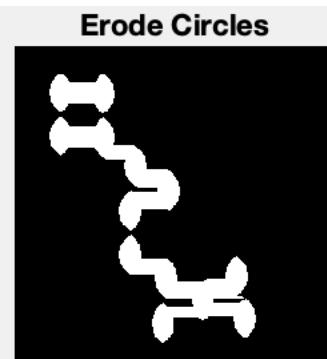
```
se = strel(5);
se = strel('arbitrary',30);
se = strel('diamond',5);
se = strel('octagon',5);
se = strel('line',20,0);
se = strel('rectangle',[7 3]);
se = strel('square',5);
se = strel('cube',5);
se = strel('cuboid',[2 3 3]);
se = strel('sphere',5);
se = strel('disk', 5);
```

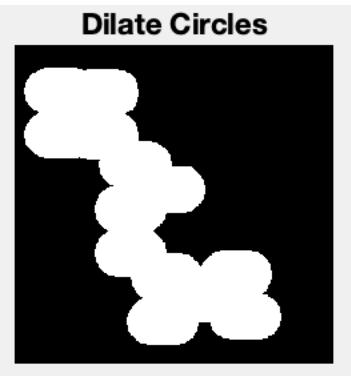
Algunos elementos como diamond o line se caracterizan por las formas ya que su forma se aleja del círculo esta puede verse reflejada en los resultados que producen.

```
se = strel(diamond,7);
```



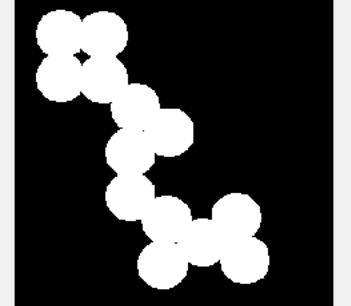
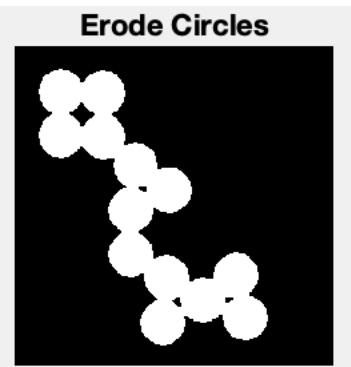
```
se = strel(line,7);
```





Otros por el contrario se caracterizan por aplicarse de forma irregular en cada color, utilizan un tamaño del elemento estructurante para cada componente rgb.

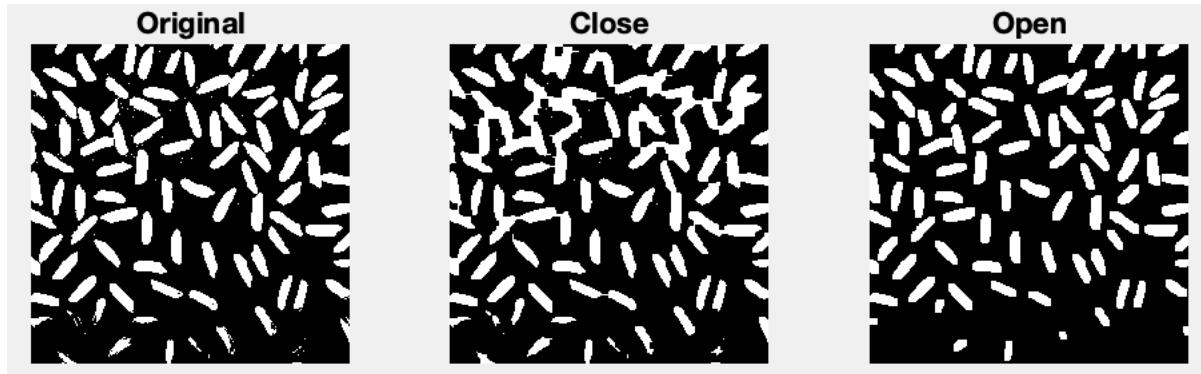
```
se = strel('cuboid',[2 3 3]);
```



## Eliminar pixels aislados

Cargamos la imagen, la binarizamos y aplicamos los operadores de apertura y cierre. La apertura consiste en una erosión seguida de una dilatación, el cierre por el contrario es una dilatación seguida de una erosión.

```
imgOriginalRice = imread('rice.png');
imgBWRice = imbinarize(imgOriginalRice);
se = strel('disk', 3);
imgRiceclosed = imclose(imgBWRice, se);
imgRiceOpened = imopen(imgBWRice, se);
```

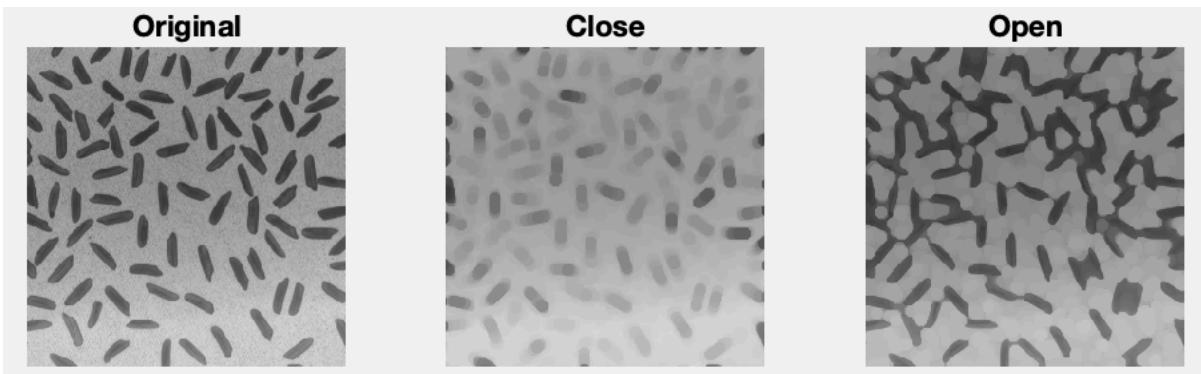


Podemos ver que al aplicar el operador cierre hemos eliminado los puntos negros que estaban dentro de los granos, sobre todo en la parte inferior de la imagen. Con la apertura logramos el efecto contrario eliminar los puntos blancos que estaban alrededor de los granos en el centro y la parte superior de la imagen.

## Cierre y apertura sobre imágenes complementarias

Aplicaremos ahora los mismos operadores, pero sobre la imagen complementaria de la original.

```
se = strel('disk', 5);
imgRiceclosed = imclose(~imgBWRice, se);
imgRiceOpened = imopen(~imgBWRice, se);
```

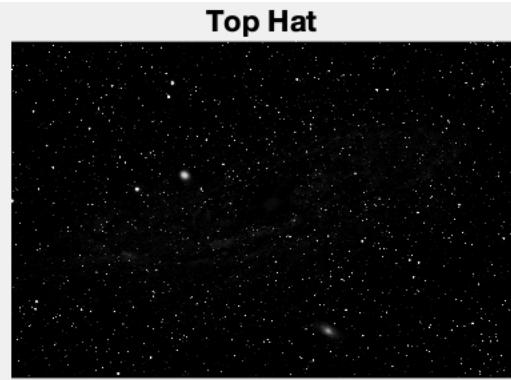
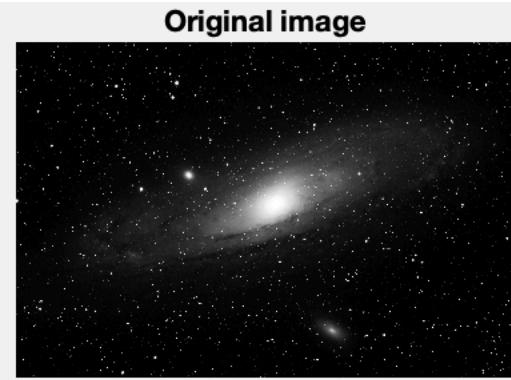


El cierre elimina los píxeles negros aislados de modo que los granos comienzan a desaparecer, por el contrario, la apertura elimina los blancos aislados por lo que los granos aumentan su superficie.

## White Top-Hat

Este operador elimina los objetos brillante más grandes que el elemento estructurante. Debemos encontrar uno lo suficientemente grande como para dejar pasar los elementos de interés y que elimine los que no deseamos.

```
se = strel('disk',10, 8);
tophat = imtophat(original,se);
```

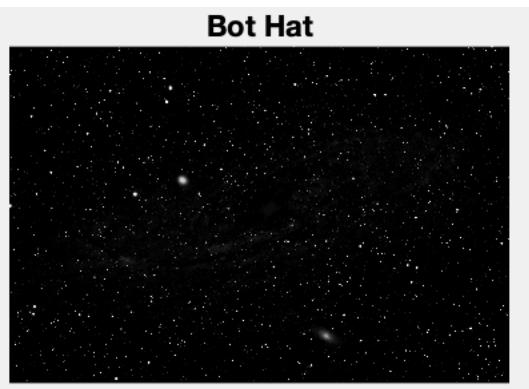
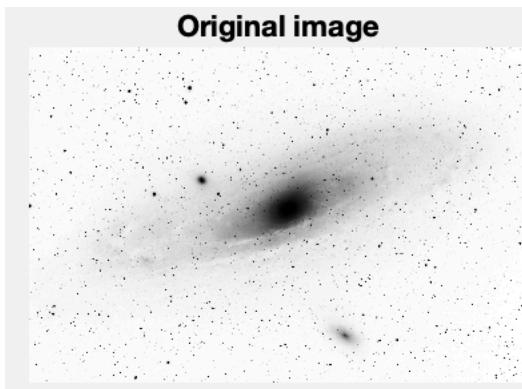


## Black Top-Hat

Este operador elimina los objetos oscuros más grandes que el elemento estructurante. Debemos encontrar uno lo suficientemente grande como para dejar pasar los elementos de interés y que elimine los que no deseamos. Ya que deseamos eliminar el mismo objeto podemos utilizar el tamaño usado anteriormente.

```
complement = imcomplement(original);
se = strel('disk',10, 8);
bothat = imbothat(complement,se);
```

Antes de aplicarlo debemos realizar el complementario de la imagen.



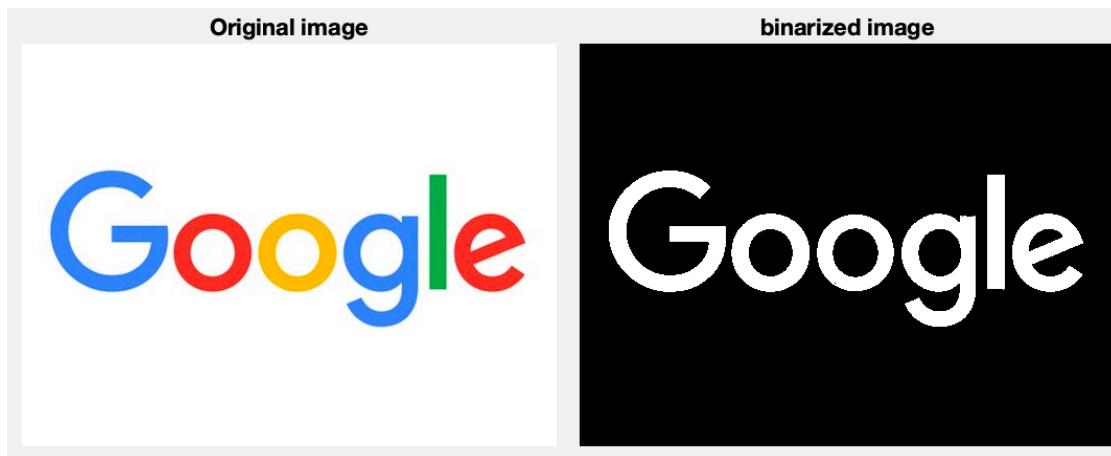
## Descriptoros de Fourier

Para realizar este apartado se utilizarán los .m proporcionados en los cuales ya está implementada la extracción de característica de Fourier, así como la operación inversa.

### Obtención de los descriptoros

Cargamos la imagen original, la binarizamos para tener una imagen con píxeles en blanco o negro, obtenemos su inverso para hacer que el fondo esté en negro y el contenido en blanco.

```
imgOriginal = imread('google.jpg');
title('Original image');
if size(imgOriginal,3) == 3
    imgOriginal = rgb2gray(imgOriginal);
end
imgBW = imbinarize(imgOriginal);
imgBW = imcomplement(imgBW);
```



A continuación, obtenemos los bordes la imagen para poder aplicar sobre ellos la extracción de descriptoros.

```
[B,L] = bwboundaries(imgBW,'noholes');
figure
imshow(label2rgb(L, @jet, [1.0 1.0 1.0]))
hold on
for k = 1:length(B)
    boundary = B{k};
    plot(boundary(:,2), boundary(:,1), 'g', 'LineWidth', 4);
end
hold off
```



Realizamos la extracción de descriptores para cada borde. Para extraer el descriptor de un único borde debemos utilizar.

```
descriptor = fourier_descriptors(boundary);
```

## Operación inversa

Extrayendo los descriptores bordes a borde y volviendo a extraer el borde de ellos con

```
boundary_recovered = fourier_inv_descriptors(descriptor, len_descriptor);
```

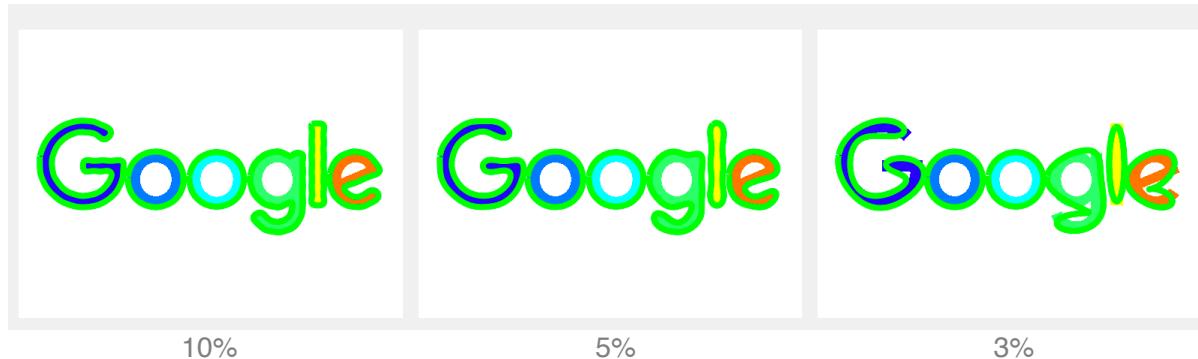
volvemos a componer la imagen de nuevo.

```
descriptor_percentage = 1;
imshow(label2rgb(L, @jet, [1.0 1.0 1.0]))
hold on
for k = 1:length(B)
    boundary = B{k};
    descriptor = fourier_descriptors(boundary);
    len_descriptor = size(descriptor)*descriptor_percentage;
    boundary_recovered = fourier_inv_descriptors(descriptor, len_descriptor);
    plot(boundary_recovered(:,2), boundary_recovered(:,1), 'g', 'LineWidth', 4);
end
hold off
```



## Operación inversa sin utilizar todos los descriptores

Si modificamos la cantidad de descriptores que utilizamos de nuevo para componer la imagen obtendremos esta ligeramente modificada pero aún así reconocible ahorrando grandes cantidades de espacio en su representación.



Podemos ver que con una reducción del 90% de los descriptores utilizado en la trasformada inversa las modificaciones obtenidas en la reconstrucción respecto de la imagen original son mínimas. Estas se acentúan cuando utilizamos todavía menos descriptores obteniendo una representación geometrizada de los bordes originales.

Podemos ahorrar mucho espacio sin perder apenas resolución reduciendo el número de descriptores utilizados en la transformada inversa.

## Características Locales

Cargamos una imagen en la que aparezca el objeto que deseamos buscar aislado. Posteriormente cargamos la imagen la que deseamos buscarlo.

```
Object1Image = imread('stapleRemover.jpg');
sceneImage = imread('clutteredDesk.jpg');
```

A continuación, extraemos las características de locales de ambos con el descriptor que se nos solicitó.

```
Object1Points = detectBRISKFeatures(Object1Image);
scenePoints = detectBRISKFeatures(sceneImage);
```

Podríamos haber utilizado otro, hay una gran variedad de ellos. Algunos son muy utilizados como SURF por su gran precisión mientras que otros como FAST son utilizados por su rapidez, de modo se que adaptan mejor a entornos donde es necesario buscar y emparejar características en tiempo real. Otros como ORB o HARRIS se centran en buscar características de líneas y esquinas.

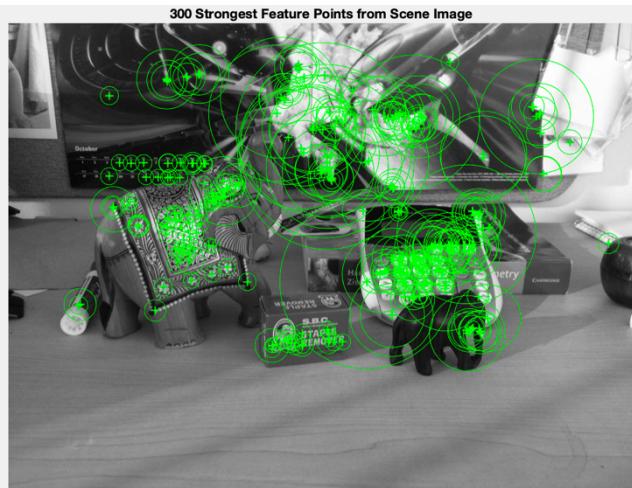
```
detectBRISKFeatures
detectFASTFeatures
detectHarrisFeatures
detectMinEigenFeatures
detectMSERFeatures
detectORBFeatures
detectSURFFeatures
detectKAZEFeatures
```

Mostramos a continuación los puntos más relevantes extraídos de cada una de las imágenes

100 Strongest Feature Points from Object1 Image



300 Strongest Feature Points from Scene Image



A partir de los puntos obtenemos sus descriptores.

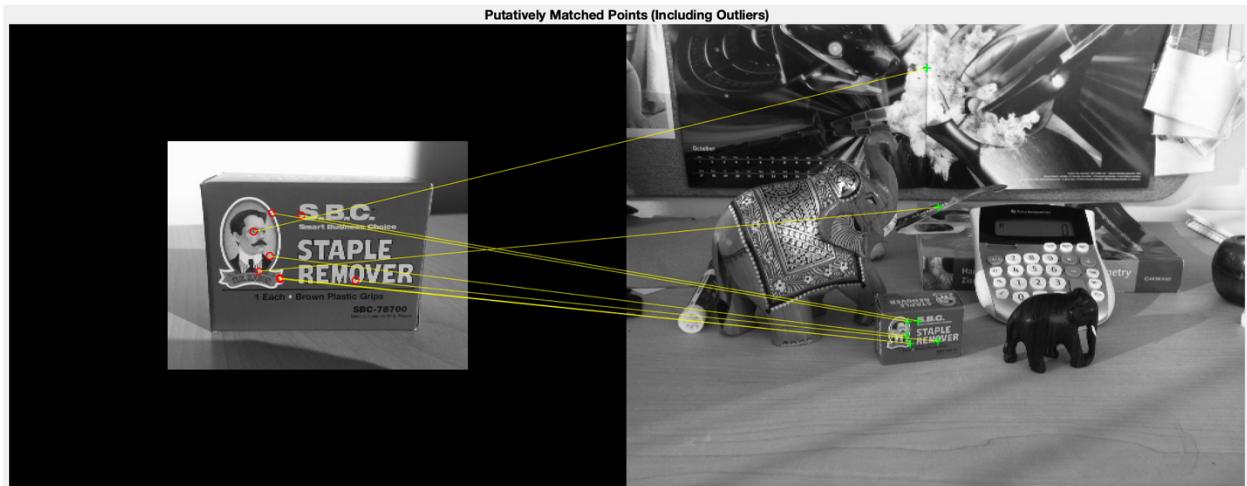
```
[Object1Features, Object1Points] = extractFeatures(Object1Image, Object1Points);
[sceneFeatures, scenePoints] = extractFeatures(sceneImage, scenePoints);
```

Utilizando los descriptores de ambas imágenes buscamos las características comunes entre ellas. Debemos emparejar los descriptores que sean comunes o tengan gran parecido en ambas imágenes.

```
Object1Pairs = matchFeatures(Object1Features, sceneFeatures, 'MatchThreshold',  
80, 'MaxRatio', 0.6);  
matchedObject1Points = Object1Points(Object1Pairs(:, 1), :);  
matchedScenePoints = scenePoints(Object1Pairs(:, 2), :);
```

Podemos en este punto modificar cómo de restrictivos somos a la hora de emparejar las características. Nosotros hemos sido relativamente permisivos pues hemos aumentado 'MatchThreshold' que por defecto está a 10, 'MatchThreshold' por defecto está a 0.6, de haberlo aumentado hubiéramos sido todavía más permisivos.

Podemos ser muy restrictivos utilizando 'Unique' para extraer solo coincidencias exactas o 'Approximate' en vez de 'Exhaustive' que es el usado por defecto si hubiéramos querido aumentar la velocidad del emparejamiento.



Como vemos son pocas las características emparejadas, no obstante, ya que la mayoría de estos emparejamientos son correctos podemos asumir que hemos logrado emparejar el objeto entre ambas fotografías.

Adicionalmente vemos que en el objeto tenemos características emparejadas en varios puntos separados de él lo cual nos indica que los parámetros están configurados correctamente y de forma robusta para estas imágenes.

```
[tform, inlierObject1Points, inlierScenePoints] =  
estimateGeometricTransform(matchedObject1Points, matchedScenePoints, 'affine');
```

A continuación, eliminamos los puntos que fueron emparejados pero que no formaban parte del objeto buscando relaciones geométricas entre la disposición de los puntos entre ambas imágenes. En este caso podemos modificar los parámetros de 'Confidence', por defecto a 99 y 'MaxDistance', por defecto a 1.5 pixeles de distancia.

```
Object1Polygon = [1, 1; size(Object1Image, 2), 1; size(Object1Image, 2),  
size(Object1Image, 1); 1, size(Object1Image, 1); 1, 1];  
newObject1Polygon = transformPointsForward(tform, Object1Polygon);
```

Por último buscamos un polígono tal que delimite al objeto a partir de los puntos emparejados.



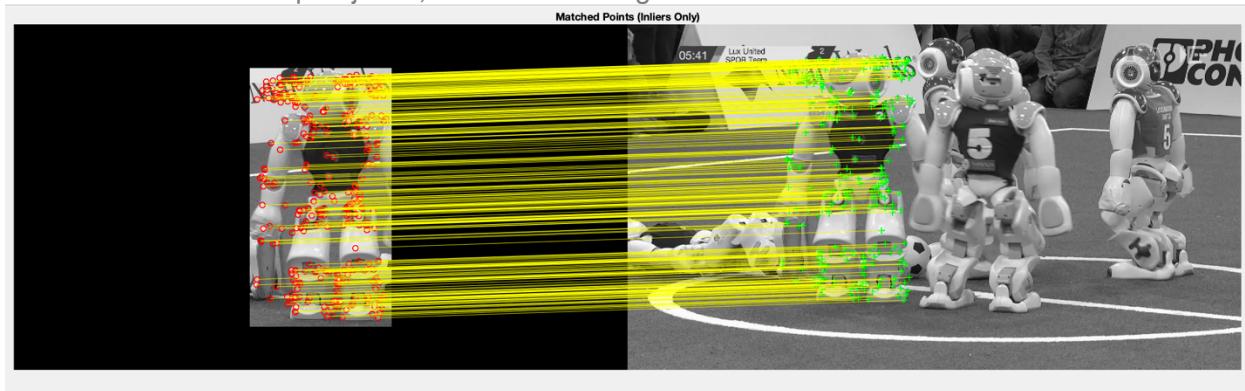
Podemos ver que nuestro objeto ha sido correctamente detectado en la imagen.

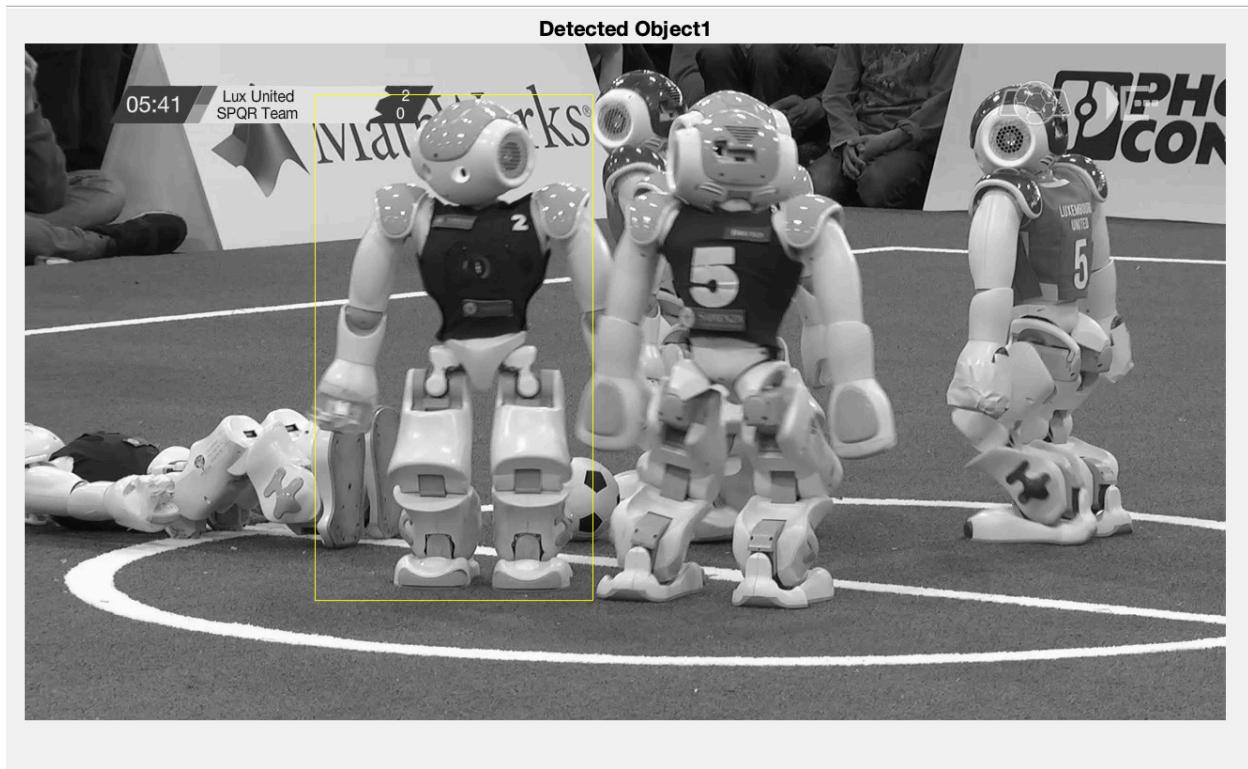
## Buscar otro objeto en otras imágenes

Repetimos el mismo proceso sobre otras imágenes y evaluamos los resultados obtenidos.

Para ello hemos buscado una imagen de la que hemos recortado un objeto.

Hemos establecido los valores a ratios muy permisivos con la intención de ver cuantas características eran emparejadas, obteniendo una gran cantidad de ellas





Adicionalmente se intentó hacer con objetos similares entre dos imágenes, pero no iguales. En este caso no se obtuvieron buenos resultados ya que tanto la forma como el brillo de las imágenes era suficientemente distinto como para no lograr emparejarlas.