PROCESAMIENTO DE IMÁGENES EN EL AJEDREZ

Ana Robledano Abasolo

4º Ingeniería Matemática UFV

Asignatura: Procesamiento Multimedia

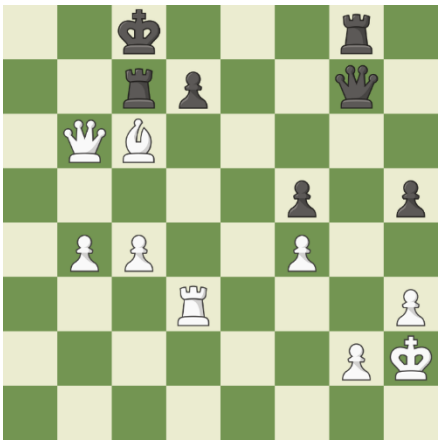
Profesor: Eusebio Daniel Rodrigues Parente

ÍNDICE

Selección del dataset

Se utilizan imágenes de tableros generadas por la famosa aplicación de ajedrez chess.com de la siguiente imagen obtenemos su **metadata** (capa lógica que se crea sobre el dato y le aporta información).

1FINAL DE PARTIDA MAGNUS CONTRA SIPKE



```
board = imread("magnus_sipke.png");

imshow(board);

imfinfo("magnus_sipke.png")
```

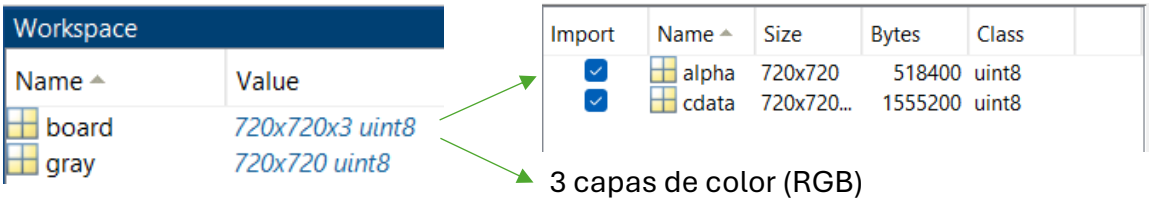
```
ans = struct with fields:
                Filename: 'C:\Users\anruk\One
             FileModDate: '05-Nov-2024 00:15:
                FileSize: 82870
                  Format: 'png'
           FormatVersion: []
                   Width: 720
                  Height: 720
                BitDepth: 24
               ColorType: 'truecolor'
         FormatSignature: [137 80 78 71 13 10
                Colormap: []
               Histogram: []
           InterlaceType: 'none'
            Transparency: 'alpha'
    SimpleTransparencyData: []
```

Destacamos algunas características de la imagen como:

Metadata	Valor
Formato	png
Width	720
Height	720
BitDepth	24
Transparency	alpha

La imagen tiene 3 capas de 8 bits cada una

Alpha tiene valores 0 o 1, no debe ser pintado o sí



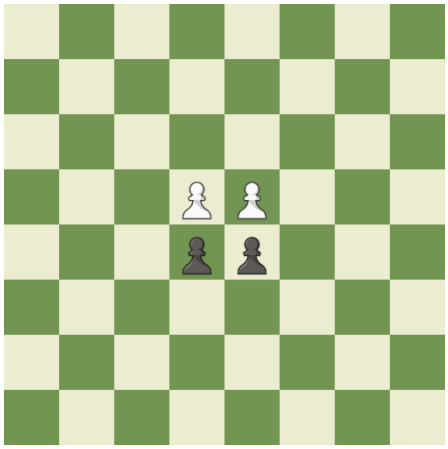
3 capas de color (RGB)

Si pasamos la imagen a escala de grises solo tendrá una capa.

Exploración visual de peones

Puesto que hay piezas blancas y negras, y casillas blancas y verdes no está claro si utilizar la imagen en blanco y negro facilitará o dificultará la tarea de reconocimiento de piezas. Por ejemplo, una pieza blanca sobre una casilla blanca podría ser más difícil de reconocer.

1º Cargamos la imagen

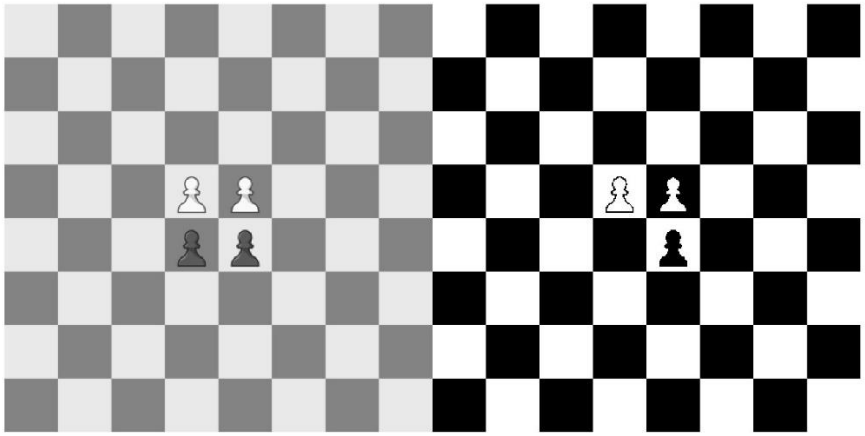


```
pawns = imread("pawns.png");
imshow(pawns);
```

2º Paso a escala de grises y binarización

```
gray_pawns = im2gray(pawns);
binary_pawns = imbinarize(gray_pawns);
montage({gray_pawns,binary_pawns});
```

Una pieza blanca sobre casilla blanca no desaparece gracias a su silueta negra. No obstante, se reduce su tamaño cuando está sobre una casilla negra ya que la silueta se mezcla con la casilla.



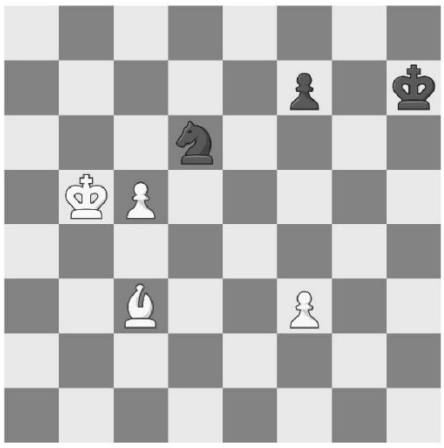
Problema: ¡Una pieza negra sobre una casilla negra desaparece en la binarización!

Posibles soluciones:

1. Binarizar con un threshold haciendo que las casillas negras (que en realidad son verdes) se binaricen a blanco y no negro.
2. Usando sensibilidad en la binarización

Exploración visual de varias piezas

Teniendo en cuenta lo anterior, se utiliza una imagen con distintas piezas.



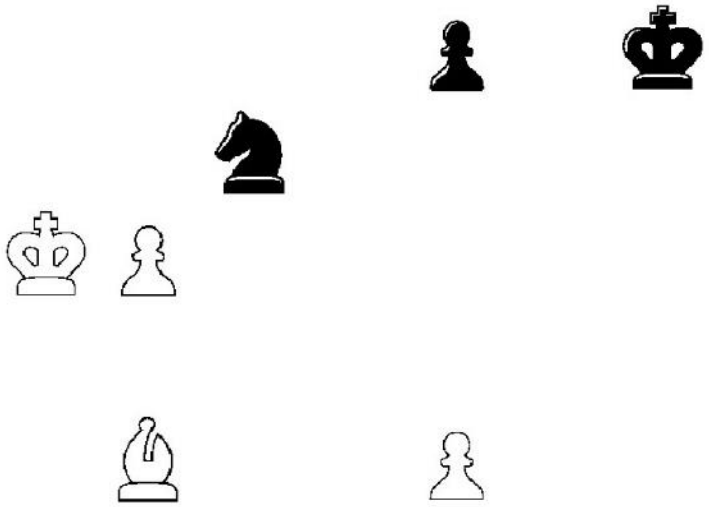
```
board_0 = imread("manyPieces.png");
gray_board = im2gray(board_0);
imshow(gray_board);
```

Y se binariza de 2 maneras:

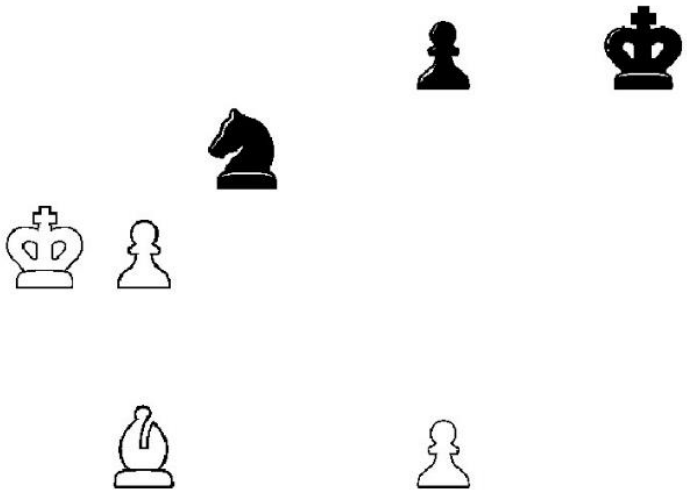
```
% Binariar con sensibilidad
binary_board = imbinarize(gray_board,0.4);
imshow(binary_board);
title('Binarización con sensibilidad')
```

```
% Binarizar con threshold
binary_board = gray_board > 128;
% si es mayor que 128 -> 255 (blanco)
% si es menor o igual -> 0 (negro)
imshow(binary_board);
title('Binarización con threshold')
```

Binarización con sensibilidad



Binarización con threshold

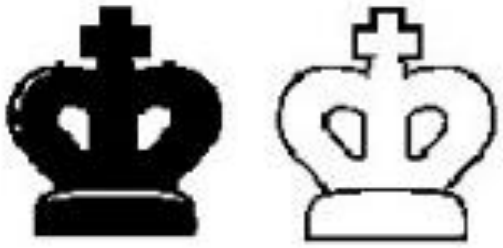


De ambas maneras, el proceso es aplicable a cualquier color de pieza sobre cualquier color de casilla.

Operaciones morfológicas

Con el siguiente pincel:

```
brush = strel("square",2);
```

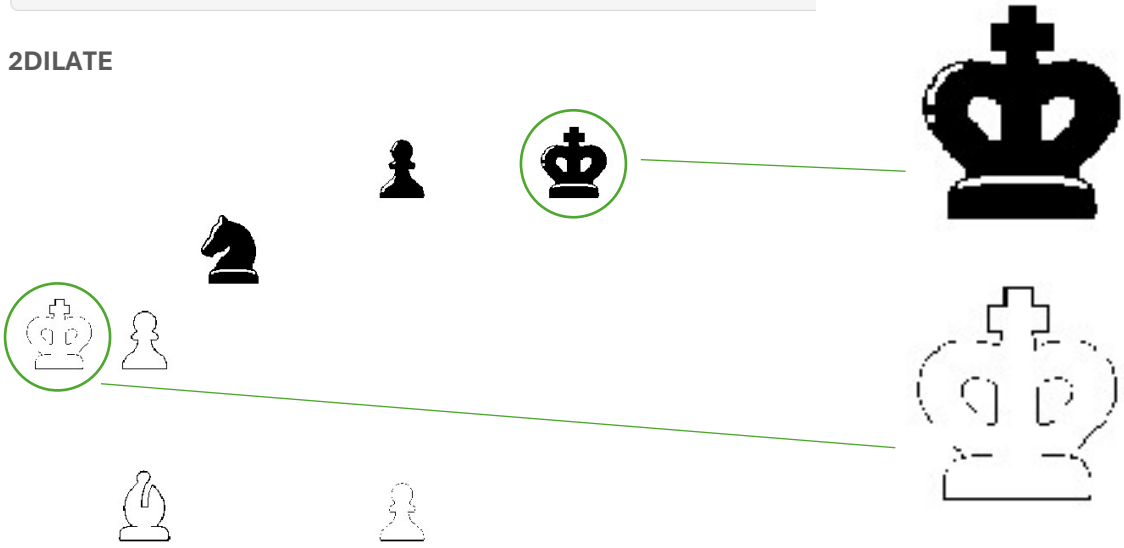


Piezas originales

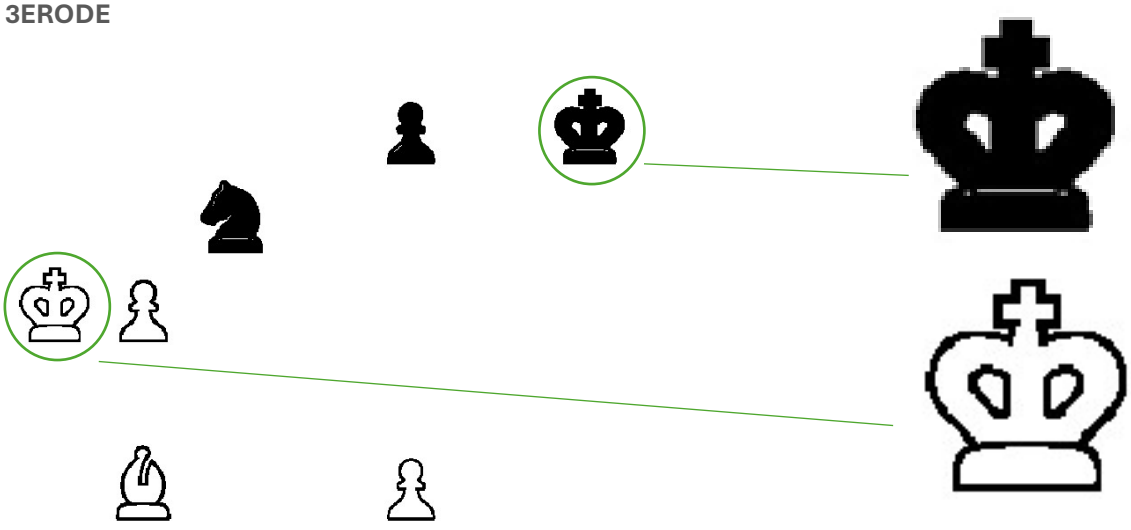
DILATACIÓN vs EROSIÓN

```
board_DILATE = imdilate(binary_board,brush);
imshow(board_DILATE);
title('Dilate')
board_ERODE = imerode(binary_board,brush);
imshow(board_ERODE);
title('Erode')
```

2DILATE



3ERODE



Dilate (Dilatación):

Aumenta las regiones blancas en la imagen.

Es útil para conectar componentes blancos que están cerca o para hacer los detalles internos de las figuras menos prominentes.

Por ejemplo, las áreas blancas de la corona del rey se expanden. Hasta tal punto que la silueta negra del rey blanco casi desaparece.

Erode (Erosión):

La erosión reduce las regiones blancas en la imagen.

Es útil para separar objetos que están conectados o para eliminar detalles más pequeños que sobresalen de los bordes.

Las áreas internas de la figura se vuelven menos visibles, como los 2 agujeros del rey. Y los detalles negros de la corona más gruesos.

Open (Apertura):

Combinación de erosión seguida de dilatación. Es útil para eliminar pequeños detalles o ruido, mientras que preserva la forma general de los objetos en la imagen.

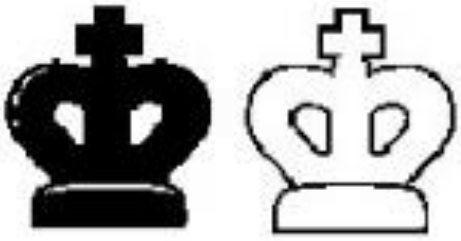
La apertura suaviza los bordes de las piezas y elimina posibles detalles internos o ruido pequeño. Como las zonas blancas dentro del rey negro.

Close (Cierre):

Dilatación seguida de una erosión. Es útil para cerrar pequeños huecos o discontinuidades en el interior de un objeto.

El cierre une o cierra pequeños espacios dentro de las figuras, como en el contorno del rey.

En la pieza del rey, los bordes se ven más gruesos y los detalles de los huecos internos se conservan, pero los contornos se ven más sólidos y conectados en comparación con la apertura.

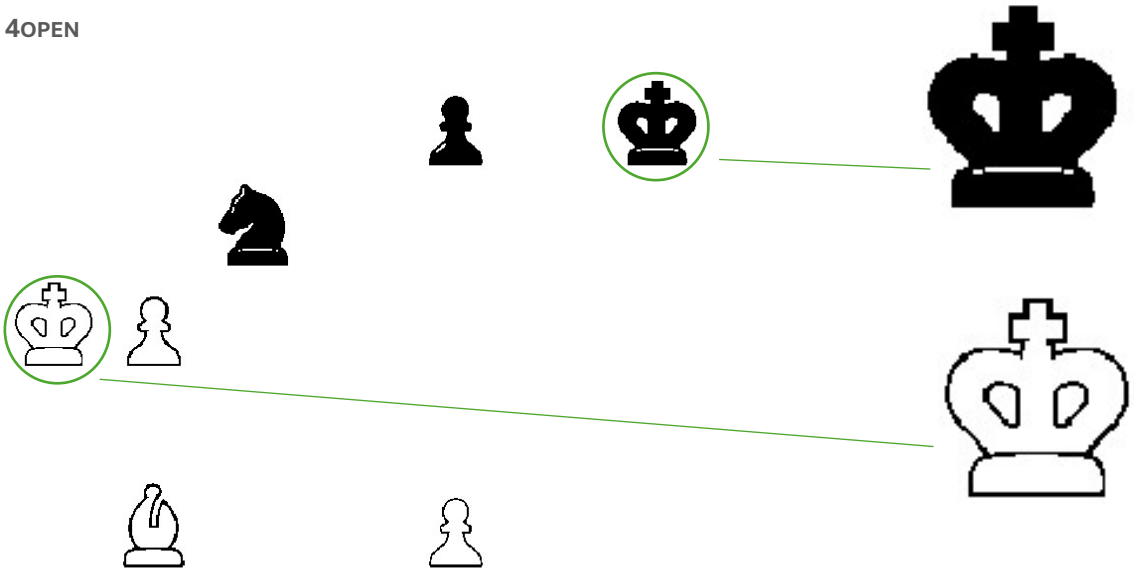


Piezas originales

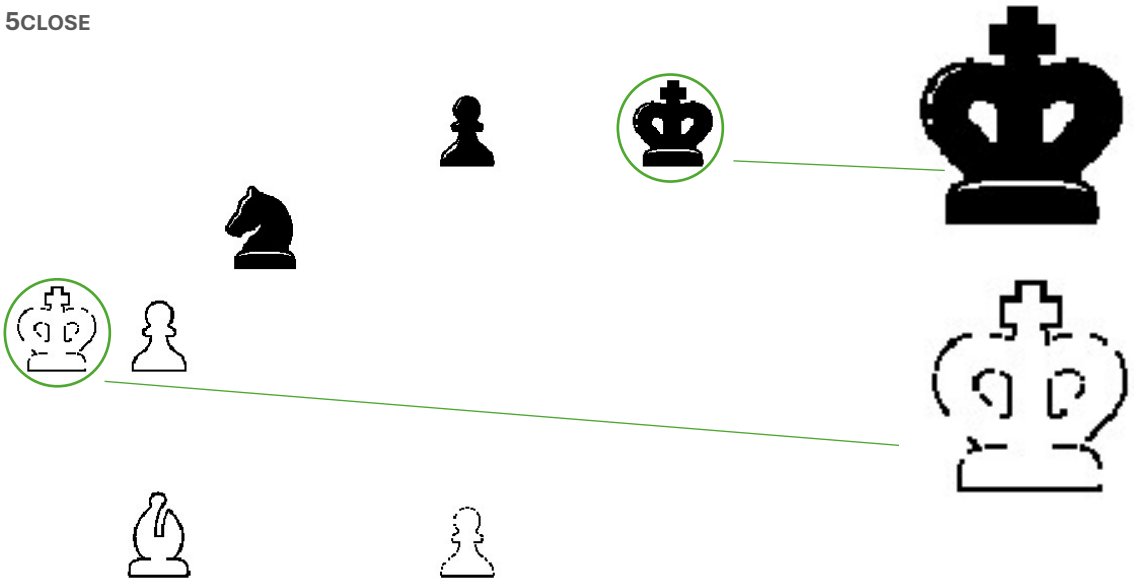
APERTURA vs CIERRE

```
board_OPEN = imopen(binary_board,brush);
imshow(board_OPEN);
title('Open')
board_CLOSE = imclose(binary_board,brush);
imshow(board_CLOSE);
title('Close')
```

4OPEN



5CLOSE



Problema: ¡Las piezas blancas se desvanecen tras operaciones morfológicas!

La dilatación hace que las siluetas de las piezas blancas se desvanecen, puesto que en el CLOSE se realiza primero una dilatación, en el CLOSE también ocurre.

Este estudio sirve para entender los efectos de las operaciones morfológicas en las piezas, pero en el algoritmo de clasificación se hará una máscara para obtener las piezas rellenas y eliminar el fondo.

Hay 2 métodos para hacer eso:

Preprocesamiento. Método 1

1. Cargar imagen del tablero e imagen de tablero vacío
2. Paso a gris de ambos
3. Resta del tablero vacío
4. Binarización
5. Relleno

Se utiliza para la resta una imagen del tablero vacío

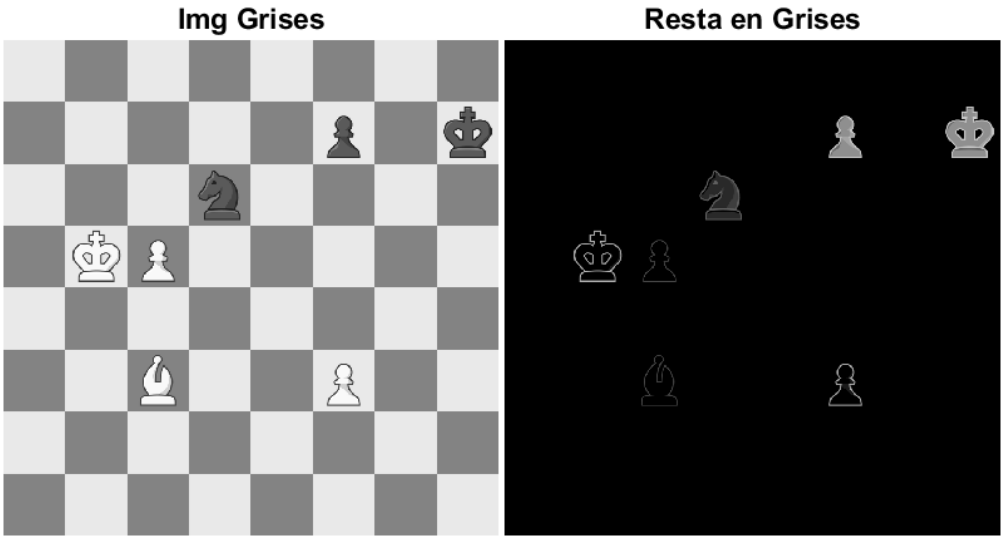
```
board_empty = imread("board_empty.png");
gray_board_empty = im2gray(board_empty);
```

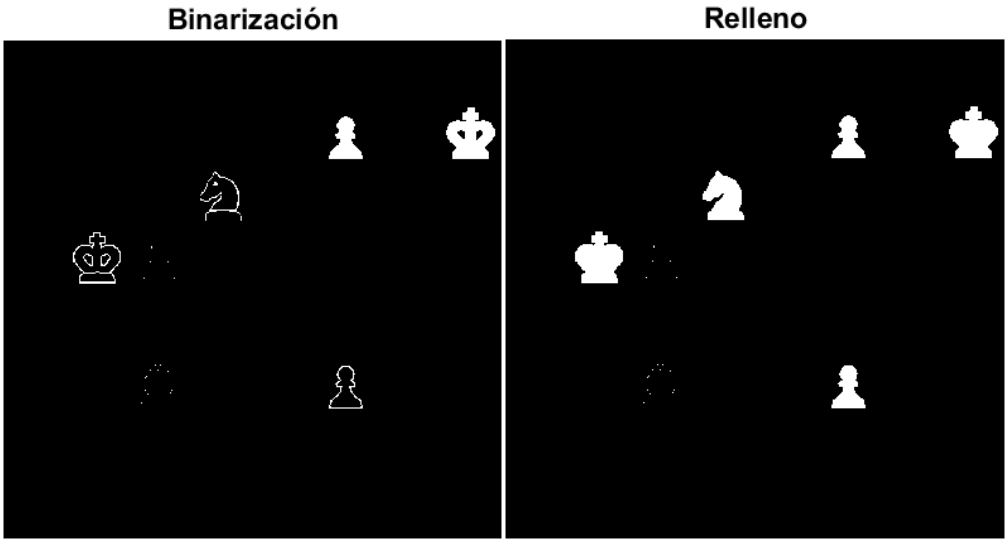
```
% Para obtener la piezas
pieces = gray_board_empty - gray_board;

% Binarización
imbin = imbinarize(pieces);

% Relleno de huecos
im_filled = imfill(imbin,"holes");
```

Para mostrar los resultados:

```
figure
% Mostrar la imagen original en escala de grises
subplot(2, 2, 1);
imshow(gray_board);
title('Img Grises');

% Mostrar la resta en escala de grises
subplot(2, 2, 2);
imshow(pieces);
title('Resta en Grises');
```

```
% Mostrar la imagen binarizada
subplot(2, 2, 3);
imshow(imbin);
title('Binarización');

% Mostrar la imagen con los huecos rellenados
subplot(2, 2, 4);
imshow(im_filled);
title('Relleno');
hold off
```





Problema: ¡la diferencia binarizada resulta en color negro!

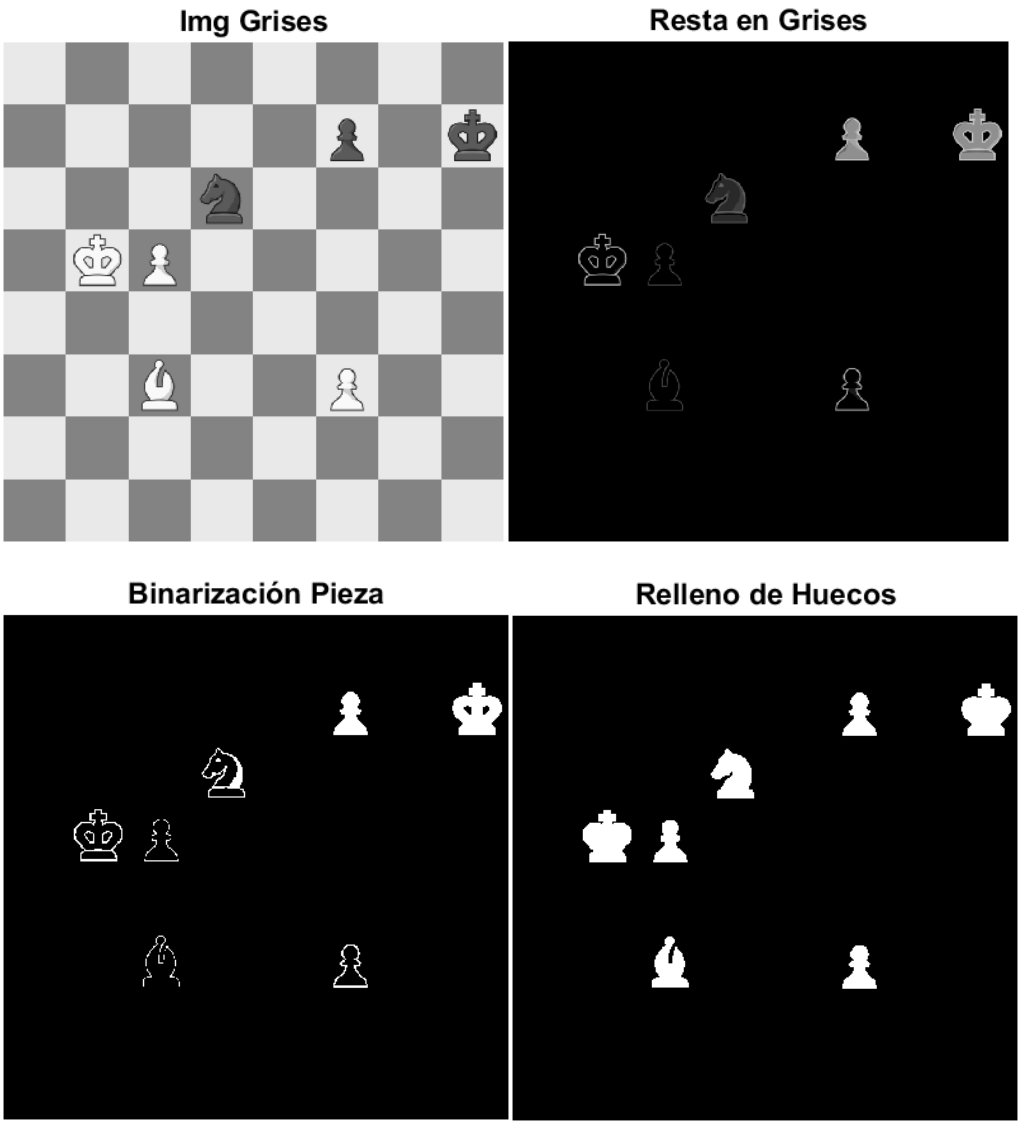
Las piezas blancas sobre casillas negras se desvanecen en la binarización, parece que binarizar una resta de piezas en escala de grises hace que el color resultante sea binarizado a negro.

Solución: Añadir menor sensibilidad a la binarización.

Para valores menores que 0.3 funciona:

```
% Binarización
imbin = imbinarize(pieces,0.2);

% Relleno de huecos
im_filled = imfill(imbin,"holes");
```

Y se vuelven a mostrar los resultados con subplots:



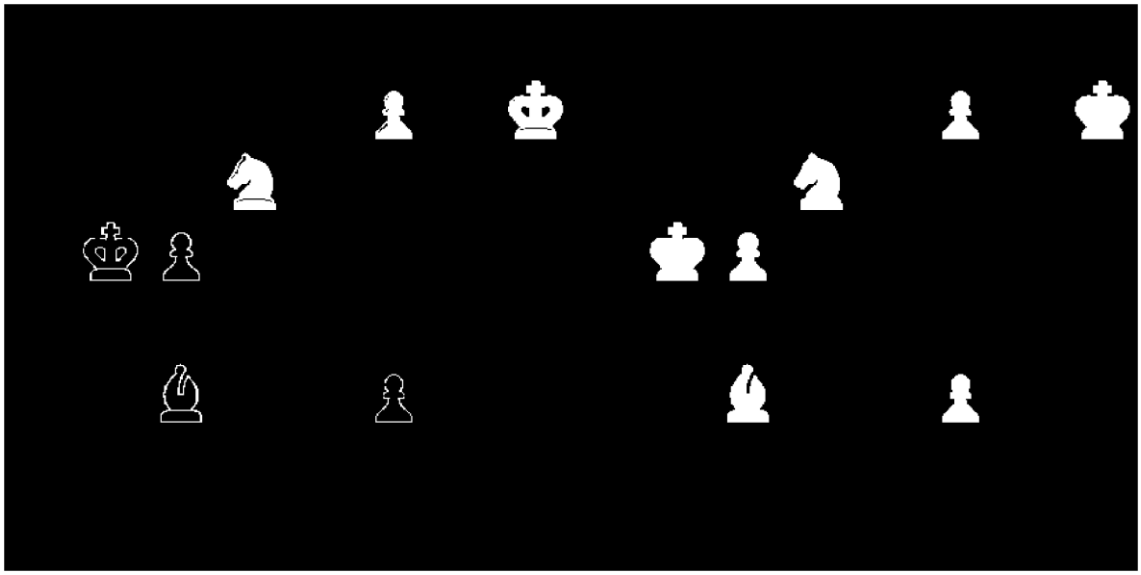
Preprocesamiento. Método 2

1. Paso a gris
2. Binarización con sensibilidad

```
binary_board = ~imbinarize(gray_board,0.5);
filled_board = imfill(binary_board,"holes");
```

Representación de resultados

```
montage({binary_board,filled_board})
```



Por este método obtenemos todas las piezas rellenas de manera más rápida y sin necesitar hacer una resta. No obstante, solo funciona debido a que las piezas blancas tienen una silueta negra. Para otro tipo de imágenes de tableros que no la tengan, el primer método sería más fiable.

Algoritmo de detección de regiones

1. Cargar la imagen

```
board_full = imread("full.png");
imshow(board_full);
```

2. Pasar a escala de grises

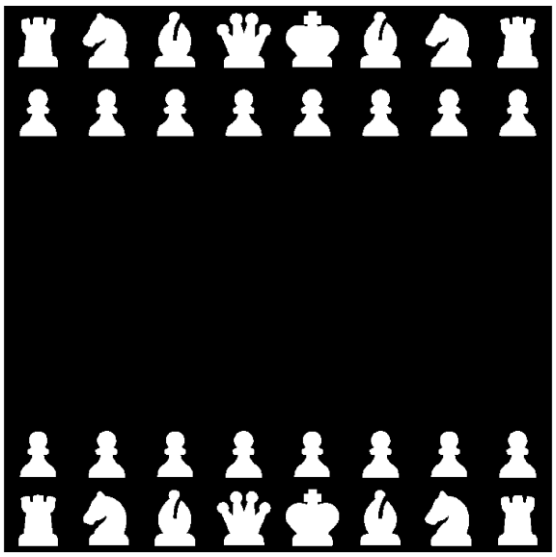
```
gray_board_full = im2gray(board_full);
```

3. Binarizar con sensibilidad 0.4

```
bin_board_full = ~imbinarize(gray_board_full,0.4);
```

4. Rellenar los huecos

```
filled_board_full = imfill(bin_board_full,"holes");
imshow(filled_board_full);
```

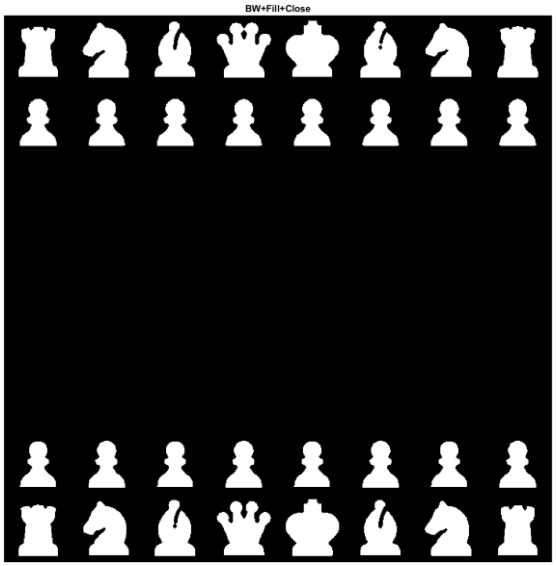


5. Aplicar operación morfológica de cierre

Tras varias pruebas, la operación siguiente es la que facilita los 2 puntos:

- Las áreas de piezas de un mismo tipo deben ser lo más parecido posible.
- Las áreas de piezas de distinto tipo deben ser lo más distintas posibles.

```
brush = strel('disk', 2);
full_close = imclose(filled_board_full,brush);

imshow(full_close);
title('BW+Fill+Close')
```



De esta manera los pequeños espacios negros dentro de las figuras blancas se reducen. Se ve fácilmente la diferencia en la cruz del rey o los dientes de las torres.

6. Identificar el nº de piezas en el tablero con:

- **bwlabel()**
  Detecta y etiqueta cada región conectada (objeto) en una imagen binaria

  - **Input full_close:** el tablero de piezas rellenas y cerradas.
  - **Output labeled_image**: imagen etiquetada en la que cada región conectada de píxeles (cada objeto) tiene un valor único (por ejemplo, 1, 2, 3, ...), permitiendo así identificar y distinguir cada región u objeto en la imagen.
  - **Output num_pieces**: nº total de regiones u objetos conectados encontrados en la imagen.

7. Calcular las áreas y centroides de **labeled_image** con:

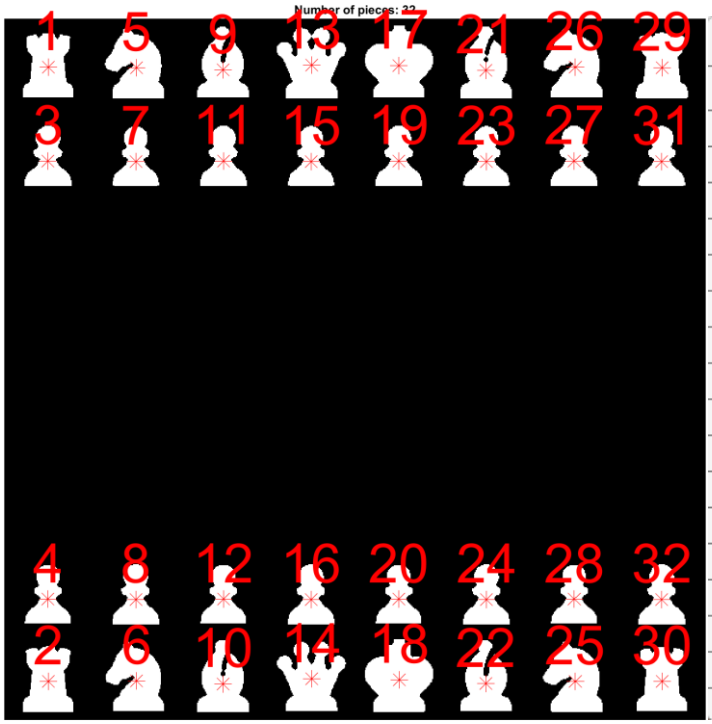
- **regionprops()**
  Calcula varias propiedades geométricas y de forma de las regiones etiquetadas en la imagen.

  - **Input labeled_image**: imagen etiquetada donde cada región tiene un valor único.
  - **Propiedades solicitadas** para cada región:
    - 'Area': Calcula el área de cada región, que es la cantidad de píxeles con valor 1 dentro de esa región.
    - 'Centroid': Calcula el centroide de cada región, que representa el punto medio de la región (similar a un "centro de masa" en términos geométricos).
  - **Output stats**: Es una estructura que contiene la información de cada región etiquetada, incluyendo el área y el centroide de cada una. Cada elemento en stats corresponde a una región en labeled_image, y puedes acceder a sus propiedades individuales.

```
% Etiquetado y análisis de regiones
[labeled_image, num_pieces] = bwlabel(full_close);

% Obtener las propiedades de las regiones
stats = regionprops(labeled_image, 'Area', 'Centroid');
```

Representación de resultados:

```
imshow(full_close);
title(['Number of pieces: ', num2str(num_pieces)]);
hold on;
for k = 1:num_pieces
    plot(stats(k).Centroid(1), stats(k).Centroid(2), 'r*','MarkerSize',20);
    % Añadir un número al centroide
    text(stats(k).Centroid(1), stats(k).Centroid(2)-5, num2str(k), 'Color', 'red', ...
         'FontSize', 26, 'HorizontalAlignment', 'center', 'VerticalAlignment', 'bottom');
end
hold off;
```



p1_chess_final_2.mlx | stats = 32×1 struct

Fields	Area	Centroid
1	2621	[45.5238,50.4548]
2	2726	[45.5202,680.7095]
3	1781	[45.4099,146.8411]
4	1659	[45.4382,596.5371]
5	3032	[136.1438,50.7117]
6	2885	[136.3016,680.2485]
7	1719	[135.4671,147.1739]
8	1763	[135.4385,596.9495]
9	2423	[224.8093,53.2076]
10	2522	[224.7982,683.4877]
11	1781	[225.4099,146.8411]
12	1659	[225.4382,596.5371]
13	3391	[315.8478,48.3996]
14	3202	[315.8186,678.4944]
15	1719	[315.4671,147.1739]
16	1763	[315.4385,596.9495]
17	3554	[405.7167,48.8168]

Con este código ya podemos identificar el número de piezas en el tablero, no obstante, también queremos clasificar cada pieza (peón,caballo,alfil,torre,rey,reina).

Según el área de cada pieza (con un rango de tolerancia) se puede conocer qué pieza es cada región encontrada.

Tras un análisis de áreas obtenemos los rangos de áreas:

8. Establecer rangos de áreas para cada tipo de pieza

Recordatorio que las áreas son el nº de píxeles de cada región, en este caso pieza.

```
% Inicializar la celda para almacenar los nombres de las piezas
pieceNames = cell(num_pieces, 1); % Preallocar espacio para los nombres
```

```
% Asignar el tipo de pieza basado en el área
for k = 1:num_pieces
    area = stats(k).Area; % Obtener el área de la región actual

    % Determinar el tipo de pieza según el área
    if area >= 1000 && area <= 1800
        pieceNames{k} = 'Peón';
    elseif area >= 2300 && area <= 2575
        pieceNames{k} = 'Alfil';
    elseif area >= 2576 && area <= 2800
        pieceNames{k} = 'Torre';
    elseif area >= 2801 && area <= 3050
        pieceNames{k} = 'Caballo';
    elseif area >= 3200 && area <= 3400
        pieceNames{k} = 'Reina';
    elseif area >= 3500 && area <= 3700
        pieceNames{k} = 'Rey';
    else
        pieceNames{k} = 'desconocido'; % Para áreas que no coinciden con
    end
end
```

```
% Añadir la columna 'Pieza' a la estructura stats
for k = 1:num_pieces
    stats(k).Pieza = pieceNames{k}; % Añadir el nombre de la pieza
end
```

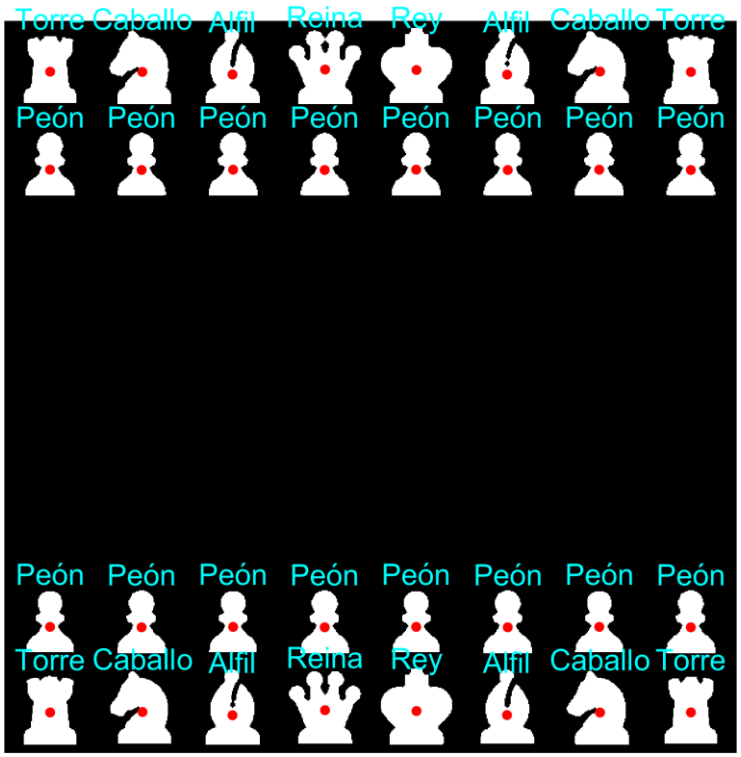
9. Mostrar tabla con resultados que verifiquen si la clasificación fue correcta

```
% Extraer las columnas relevantes en una tabla
resultTable = table([stats.Area]', pieceNames, 'VariableNames', {'Area', 'Pieza'});

% Mostrar la tabla resultante
resultTable
```

	Area	Pieza						
1	2621	'Torre'	11	1781	'Peón'	22	2397	'Alfil'
2	2726	'Torre'	12	1659	'Peón'	23	1719	'Peón'
3	1781	'Peón'	13	3391	'Reina'	24	1763	'Peón'
4	1659	'Peón'	14	3202	'Reina'	25	3007	'Caballo'
5	3032	'Caballo'	15	1719	'Peón'	26	2908	'Caballo'
6	2885	'Caballo'	16	1763	'Peón'	27	1781	'Peón'
7	1719	'Peón'	17	3554	'Rey'	28	1659	'Peón'
8	1763	'Peón'	18	3652	'Rey'	29	2750	'Torre'
9	2423	'Alfil'	19	1781	'Peón'	30	2587	'Torre'
10	2522	'Alfil'	20	1659	'Peón'	31	1719	'Peón'
			21	2554	'Alfil'	32	1763	'Peón'

10. Mostrar la imagen con los centroides y nombre de la pieza

```
imshow(full_close);
hold on; % Mantener la imagen actual para dibujar sobre ella

% Iterar a través de cada región para plotear los centroides y los nombres de las piezas
for k = 1:num_pieces
    % Marcar el centroide
    plot(stats(k).Centroid(1), stats(k).Centroid(2), 'r.', 'MarkerSize', 30);

    % Añadir el nombre de la pieza al centroide
    text(stats(k).Centroid(1), stats(k).Centroid(2)-35, stats(k).Pieza, 'Color', 'cyan',
         'FontSize', 26, 'HorizontalAlignment', 'center', 'VerticalAlignment', 'bottom');
end

hold off;
```



Generalización del algoritmo

Mediante la función pieceClassifier.m

- Inputs: brush con el que se realizarán operaciones morfológicas y board imagen .png de un tablero de chess.com
- Outputs: stats con áreas y centroides; labeled_image y nº de piezas de la imagen

```
function [stats, labeled_image, num_pieces] = pieceClassifier(brush,board)
    % imagen -> grises -> binarización -> apertura
    gray_board = im2gray(board);
    bin_board= ~imbinarize(gray_board,0.4);
    filled_board = imfill(bin_board,"holes");
    close_board = imclose(filled_board,brush);

    % Etiquetado y análisis de regiones
    [labeled_image, num_pieces] = bwlabel(close_board);

    % Obtener las propiedades de las regiones
    stats = regionprops(labeled_image, 'Area', 'Centroid');

    % Inicializar la celda para almacenar los nombres de las piezas
    pieceNames = cell(num_pieces, 1); % Preallocar espacio para los nombres de las piezas
```

```
    % Asignar el tipo de pieza basado en el área
    for k = 1:num_pieces
        area = stats(k).Area; % Obtener el área de la región actual

        % Determinar el tipo de pieza según el área
        if area >= 1000 && area <= 1800
            pieceNames{k} = 'Peón';
        elseif area >= 2300 && area <= 2575
            pieceNames{k} = 'Alfil';
        elseif area >= 2576 && area <= 2800
            pieceNames{k} = 'Torre';
        elseif area >= 2801 && area <= 3050
            pieceNames{k} = 'Caballo';
        elseif area >= 3200 && area <= 3400
            pieceNames{k} = 'Reina';
        elseif area >= 3500 && area <= 3700
            pieceNames{k} = 'Rey';
        else
            pieceNames{k} = 'desconocido'; % Para áreas que no coinciden con ningún tipo
        end
    end

    % Añadir la columna 'Pieza' a la estructura stats
    for k = 1:num_pieces
        stats(k).Pieza = pieceNames{k}; % Añadir el nombre de la pieza
    end

    return
end
```

Mediante la función visualizePieces.m también se evita repetir código.

```
function visualizePieces(num_pieces,stats,board)
    imshow(board);
    hold on;

    % Iterar a través de cada región para plotear los centroides y los nombres de las piezas
    for k = 1:num_pieces
            % Marcar el centroide
            plot(stats(k).Centroid(1), stats(k).Centroid(2), 'r.', 'MarkerSize', 4);

            text(stats(k).Centroid(1), stats(k).Centroid(2)-35, stats(k).Pieza, ...
                 'Color', 'blue', 'FontWeight', 'bold', ...
                 'FontSize', 7, 'HorizontalAlignment', 'center', 'VerticalAlignment', 'bottom');
    end

    hold off;
    return
```

Posibles visualizaciones de resultados

Usando las funciones anteriores, mostramos distintas maneras de representar los resultados.

```
[stats, labeled_image, num_pieces] = pieceClassifier(brush,board_full);
```

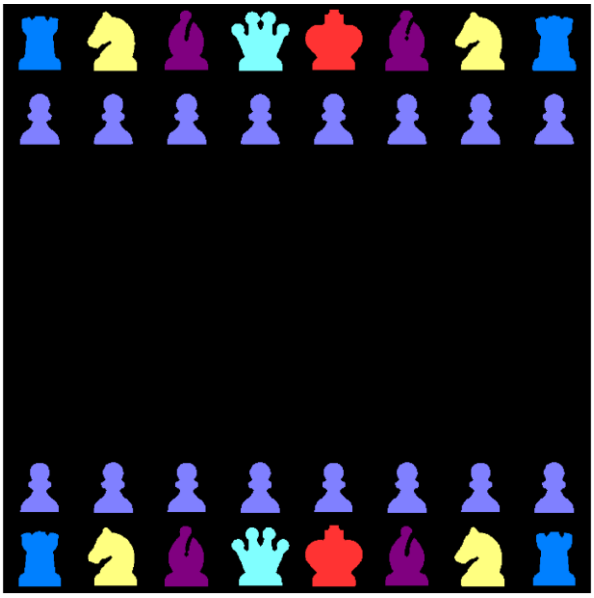
- Imagen original con etiquetas

```
visualizePieces(num_pieces,stats,board_full)
```

- Imagen con color

```
% Definir colores específicos para cada tipo de pieza
color_peon = [0.5, 0.5, 1];     % Lila
color_alfil = [0.5, 0, 0.5];    % Morado
color_torre = [0, 0.5, 1];      % Azul
color_caballo = [1, 1, 0.5];    % Amarillo
color_reina = [0.5, 1, 1];      % Cyan
color_rey = [1, 0.2, 0.2];      % Rojo
color_desconocido = [0, 0, 0]; % Negro para piezas desconocidas

% Inicializar el arreglo para almacenar los colores
pieceColors = zeros(num_pieces, 3);
```

```
% Asignar colores basados en 'Pieza'
for k = 1:num_pieces
    switch stats(k).Pieza
        case 'Peón'
            pieceColors(k, :) = color_peon;
        case 'Alfil'
            pieceColors(k, :) = color_alfil;
        case 'Torre'
            pieceColors(k, :) = color_torre;
        case 'Caballo'
            pieceColors(k, :) = color_caballo;
        case 'Reina'
            pieceColors(k, :) = color_reina;
        case 'Rey'
            pieceColors(k, :) = color_rey;
        otherwise
            pieceColors(k, :) = color_desconocido;
    end
end

% Crear la imagen en color usando labeled_image
coloredImage = zeros(size(labeled_image, 1), size(labeled_image, 2), 3);

% Colorear cada pieza en función de su etiqueta
for k = 1:num_pieces
    color = pieceColors(k, :);
    mask = labeled_image == k;

    % Asignar el color en cada canal
    coloredImage(:,:,1) = coloredImage(:,:,1) + mask * color(1);
    coloredImage(:,:,2) = coloredImage(:,:,2) + mask * color(2);
    coloredImage(:,:,3) = coloredImage(:,:,3) + mask * color(3);
end

imshow(coloredImage);
```

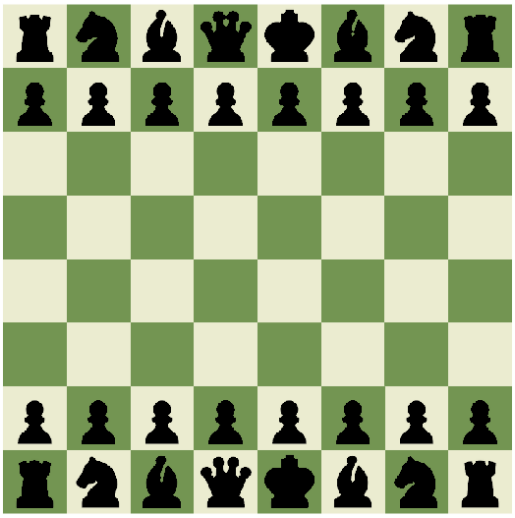


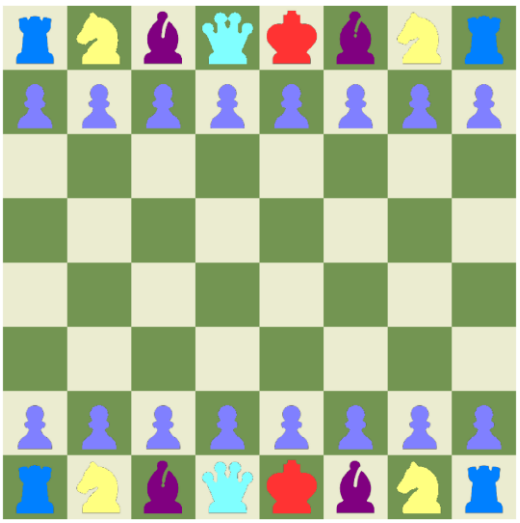
- Superposición con color

```
% Crear una máscara para las regiones que deseas eliminar
mask = labeled_image > 0;  % Máscara con todas las regiones etiquetadas

% Aplicar la máscara para eliminar las regiones en `board_full`
% Para hacerlas negras, establece los valores en 0
board_full_reduced = board_full;  % Crear una copia de `board_full`
board_full_reduced(repmat(mask, [1 1 3])) = 0;  % Establecer en 0 las re

% Mostrar la imagen con las regiones eliminadas
imshow(board_full_reduced);
```

Ambas imágenes deben ser del mismo tipo para poder sumarlas.

```
% Convertir coloredImage a uint8
coloredImage_uint8 = im2uint8(coloredImage);

% Realizar la resta
test = board_full_reduced + coloredImage_uint8;

% Mostrar la imagen resultado
imshow(test);
```





Verificación de precisión del algoritmo

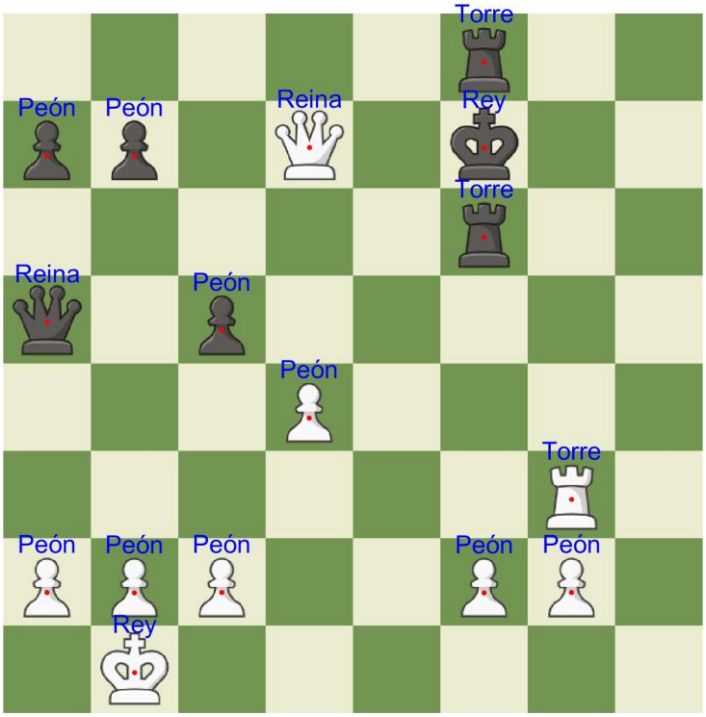
Se clasifican las piezas de otras imágenes de tableros de partidas reales.

Partida Magnus - Sipke

- Magnus Caerlesen (2581ptos.) vs Sipke Erns (2521ptos.)
- Date: 2004.01.24
- Result: 1-0
- Event: It (cat.9)
- Site: Wijk aan Zee (Netherlands)

```
magnus_sipke = imread("magnus_sipke.png");
brush = strel('disk', 2);

[stats, ~, num_pieces] = pieceClassifier(brush,magnus_sipke);
```

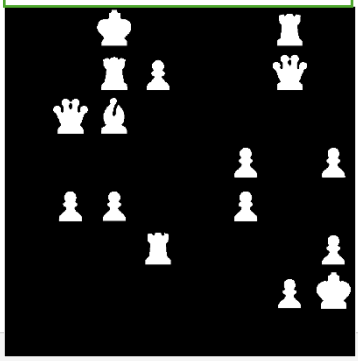
```
visualizePieces(num_pieces,stats,magnus_sipke)
```



Partida Kasparov - Ivanov

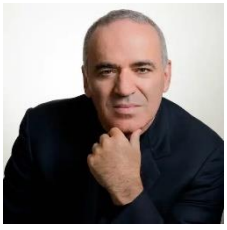
- Anatoly Kasparov vs Igor V Ivanov
- Date: 1979.??.??
- Result: 1-0
- Event: Olympiad URS
- Site: Moscow (Russia)

Labeled_image



```
kasparov_ivanov = imread("kasparov_ivanov.png");
brush = strel('disk', 2);

[stats, labeled_image, num_pieces] = pieceClassifier(brush,kasparov_ivanov);
imshow(labeled_image)
```

```
visualizePieces(num_pieces,stats,kasparov_ivanov)
```

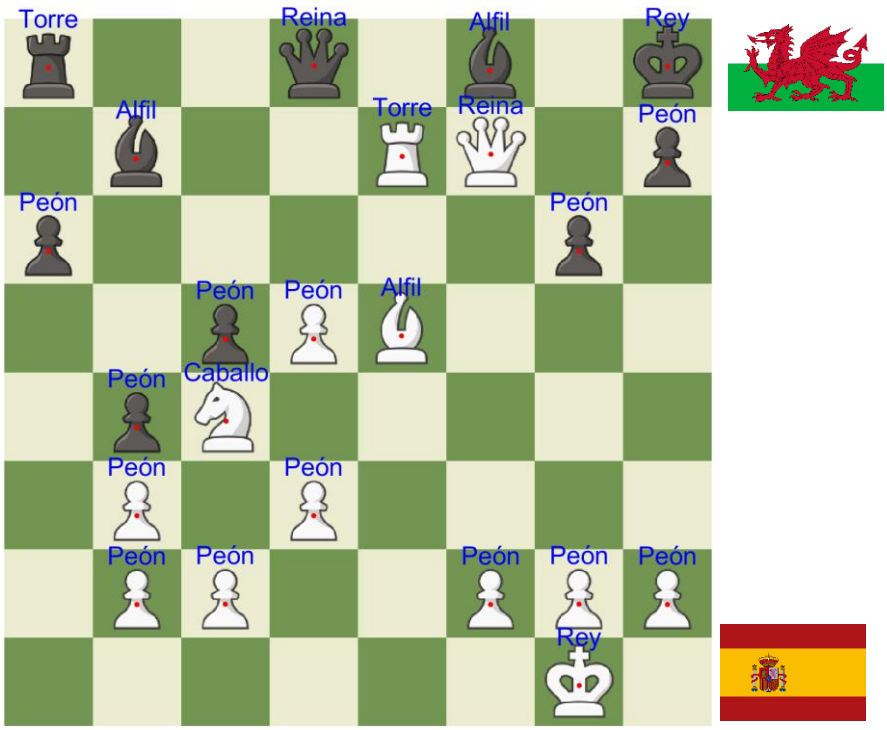


Partida anruki – prhill

- anruki11(1310) vs prhill (1310)
- Date: 02.11.2024
- Result: 1-0
- Site: chess.com

```
ana_phrill = imread("ana_phrill.png");
brush = strel('disk', 2);

[stats, labeled_image, num_pieces] = pieceClassifier(brush,ana_phrill);
```

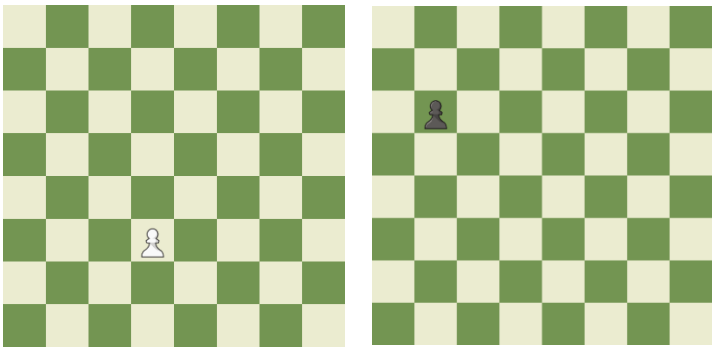
```
visualizePieces(num_pieces,stats,ana_phrill)
```

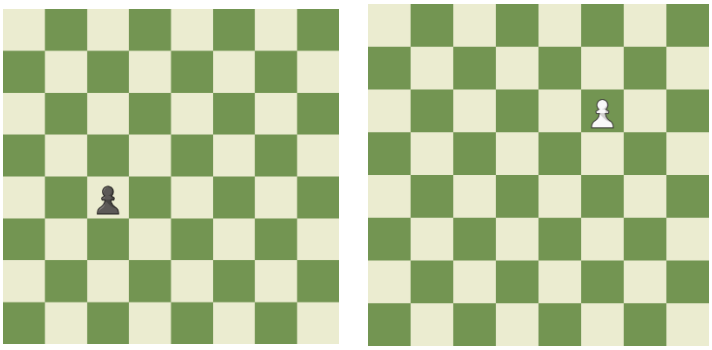


Conclusiones

- Se ha de tener en cuenta el color de la pieza y el de su casilla (negra/blanca) a la hora de binarizar.
- Debido a que las piezas blancas tienen una silueta negra es posible usar threshold y así evitar hacer la resta del tablero vacío. Para otro tipo de estilos de piezas podría ser necesaria la resta.
- Las operaciones morfológicas se deben realizar después de haber rellenado la región con imfill()

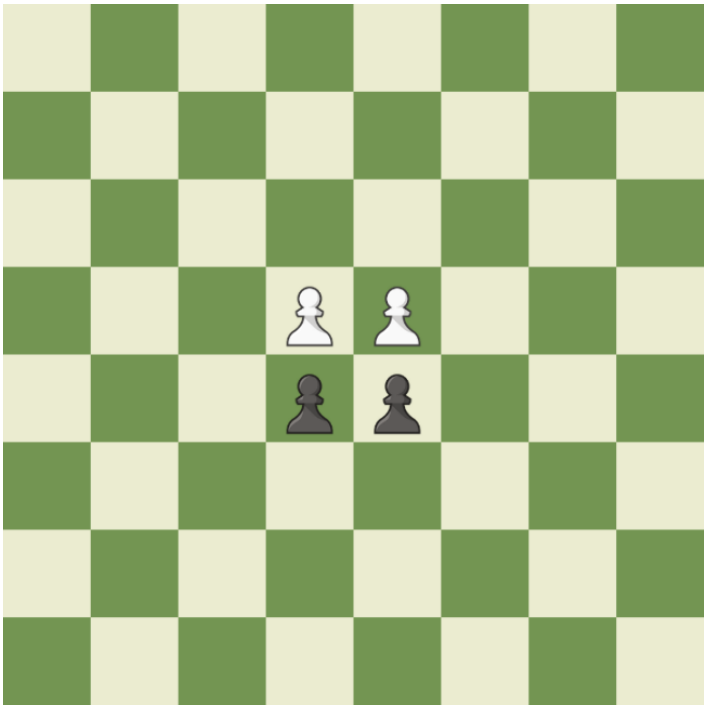
- Incluso para piezas del mismo tipo y color, las áreas calculadas con regionprops() dan resultados ligeramente distintos, que deben ser acotados.
- Utilizar imágenes que contienen distintos casos, hace el procesamiento más eficiente que si se analizase cada caso por separado.





ANALIZAR 4 IMÁGENES

Con cada caso

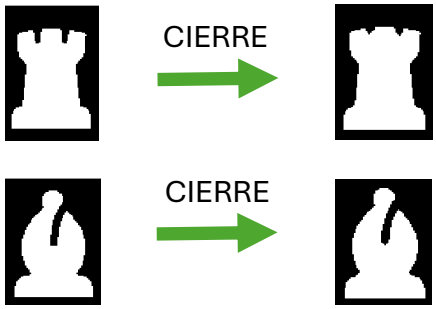


ANALIZAR 1 IMAGEN

Con todos los casos

^ Así se asegura que la solución sea válida para todos los casos

- Teniendo piezas con áreas muy parecidas (alfil y torre) al aplicar un cierre los agujeros se cierran y las áreas se distinguen mejor.



^ El área de la torre aumenta mientras que el área del alfil disminuye.

- Tener intervalos de áreas sin ninguna intersección hace que la clasificación no dé lugar a error ante cualquier imagen propuesta en los ejemplos.

Vista a futuro: con otros modelos de tableros

En la web chess.com se puede personalizar el aspecto de las piezas y el tablero, sería interesante generalizar el algoritmo para cualquier estilo.

Lo que más diferiría serían los rangos de las áreas. Entonces en vez de establecer rangos numéricos ordenar de mayor a menor las áreas y establecer la posición de cada pieza en el ranking de áreas.