



# Procesamiento de Imágenes

07/07/2021

---

## Integrantes

Gervasi, Sofia

Venanzoni, Martina

Rodriguez, Javier

## Profesores

Raponi, Marcelo

Rios, Betina

## Introducción

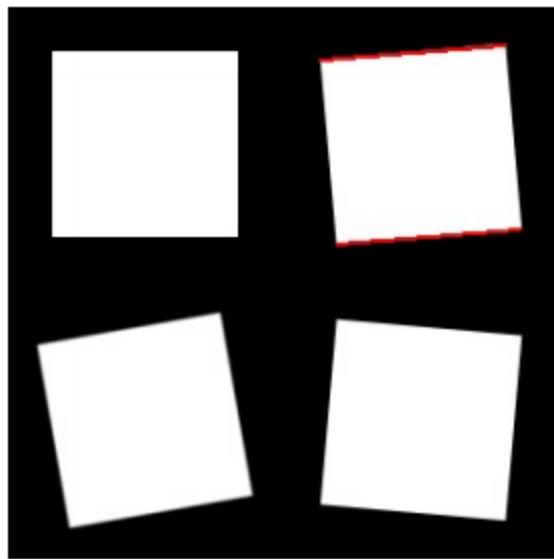
En el presente Trabajo Práctico Final para la materia de **Procesamiento de Imágenes**, mostramos los conocimientos adquiridos a lo largo de la cursada en el desarrollo de los scripts para el análisis de distintos tipos de imágenes, y empleando métodos y criterios para hallar el resultado esperado.

En cada ejercicio resuelto, se explica al detalle cuáles fueron las **herramientas computacionales** utilizadas y el desarrollo de la lógica implementada, como así también que se tuvo en cuenta para ciertos casos para evitar resultados erróneos y/o consideraciones a tener en cuenta.

## Consignas

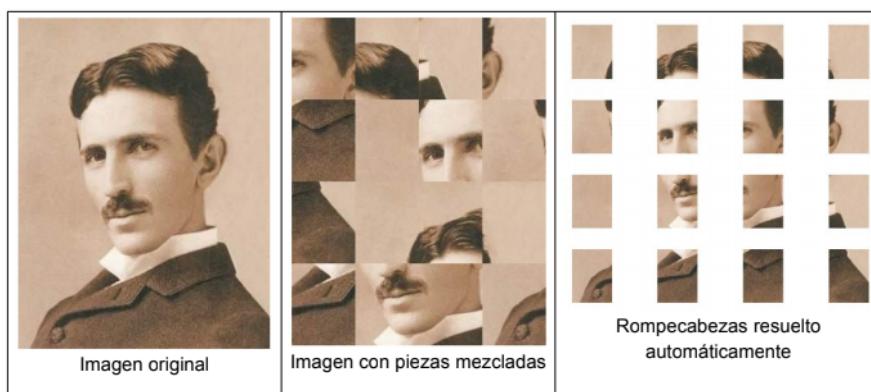
### Problema 1: Detección de bordes orientados

El objetivo de este problema es implementar un algoritmo que permita detectar sólo los bordes marcados en rojo.



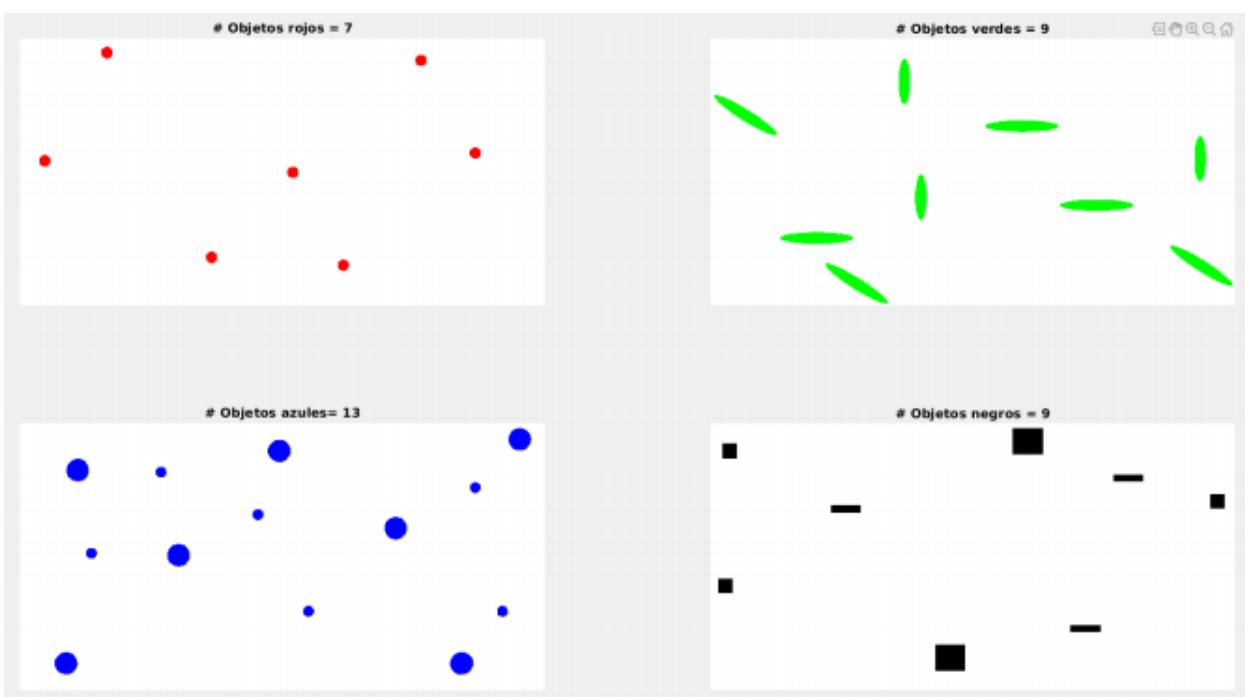
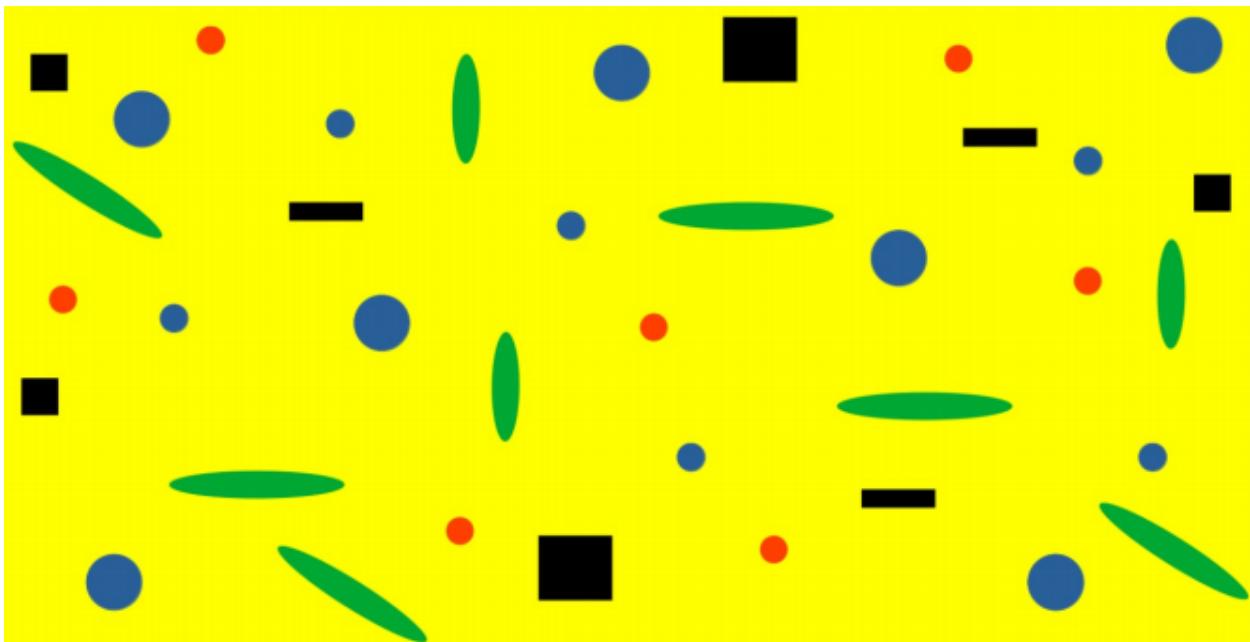
### Problema 3: Rompecabezas (Puzzle)

El objetivo de este ejercicio es seleccionar una imagen color, subdividirla en 16 matrices (cada una será un pieza del rompecabezas). Luego deberá mezclar dichas piezas y comenzar a unirlas, una por una y de manera automática, hasta formar la imagen original. Deberá probar que el algoritmo trabaje apropiadamente con diferentes imágenes.



## Problema 4: Segmentación basada en color

El objetivo de este ejercicio es generar una imagen con fondo uniforme (amarillo) y figuras geométricas (círculos, elipses, cuadrados, rectángulos) de diferentes tamaños y color (rojo, verde, azul, negro). Luego, aplicando algoritmos de segmentación adecuados, deberán clasificar y contar los objetos que hay de cada color. Se recomienda pasar del espacio de color RGB al L\*a\*b\* para realizar la segmentación.



## Resultados

### Problema 1

El objetivo de este ejercicio es implementar un algoritmo que permita detectar sólo los bordes indicados en la imagen.

Para comenzar, definimos el eje de coordenadas que utilizaremos de referencia para trabajar.

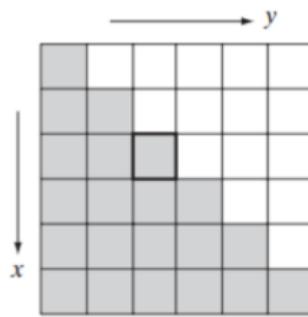


Figura 1. Representación del sistema de referencia elegido

Dado nuestro sistema de referencia, se utilizó el comando **fspecial('prewitt')** para obtener la matriz de convolución vertical y posteriormente se la rotó 90° para conseguir la horizontal.

**dX =**

$$\begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix}$$

**dY =**

$$\begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix}$$

Figura 2. Máscaras obtenidas usando *fspecial*

Mediante la función **conv2** aplicamos convolución sobre nuestra imagen en escala de grises y los kernels obtenidos anteriormente. Esto nos permite calcular el gradiente de intensidad de cada píxel y observando esta variación de intensidad, podremos detectar la presencia de un borde.

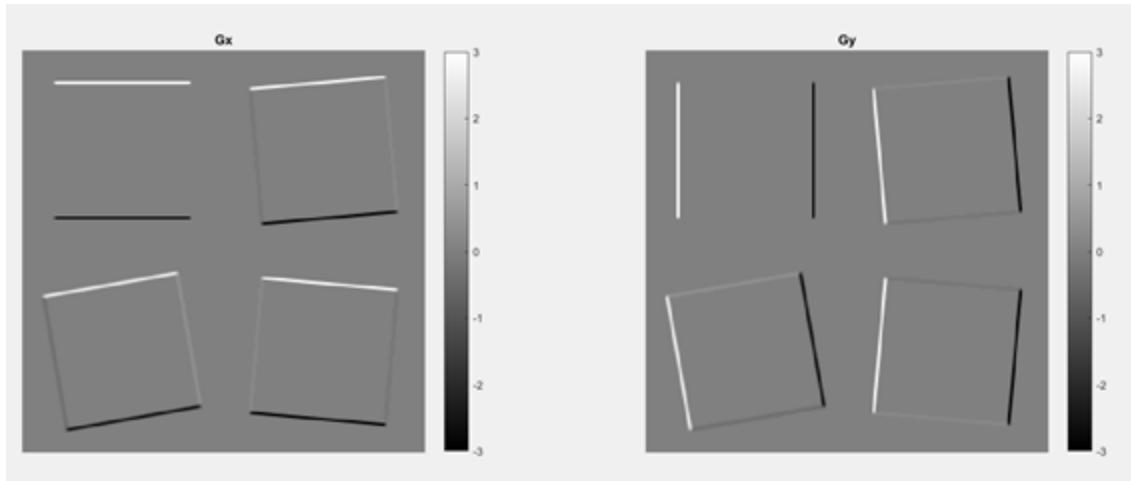


Figura 3. Gráfico del gradiente vertical y horizontal

Como los bordes que deseamos detectar son orientados, utilizaremos el gradiente para determinar la dirección del borde en un determinado punto. Esta operación puede realizarse debido a que el borde es perpendicular a la dirección del vector gradiente en el punto que se esté evaluando.

Para calcular la matriz de ángulos del gradiente, debemos obtener el ángulo comprendido entre la componente horizontal del vector gradiente ( $G_y$ ) y la vertical ( $G_x$ ). Por cómo fueron orientados nuestros ejes, este ángulo se obtendrá calculando el  $\tan^{-1}(G_y/G_x)$ . Para ello se utiliza la función **atan**, la cual calcula el arcotangente y devuelve el valor del ángulo obtenido en radianes. Luego, convertimos los radianes a grados usando **rad2deg**.

Como mencionamos anteriormente, la dirección del vector gradiente es perpendicular al borde, por ende, podremos calcular la matriz de ángulos del borde restándole  $90^\circ$  a la matriz de ángulos del gradiente.

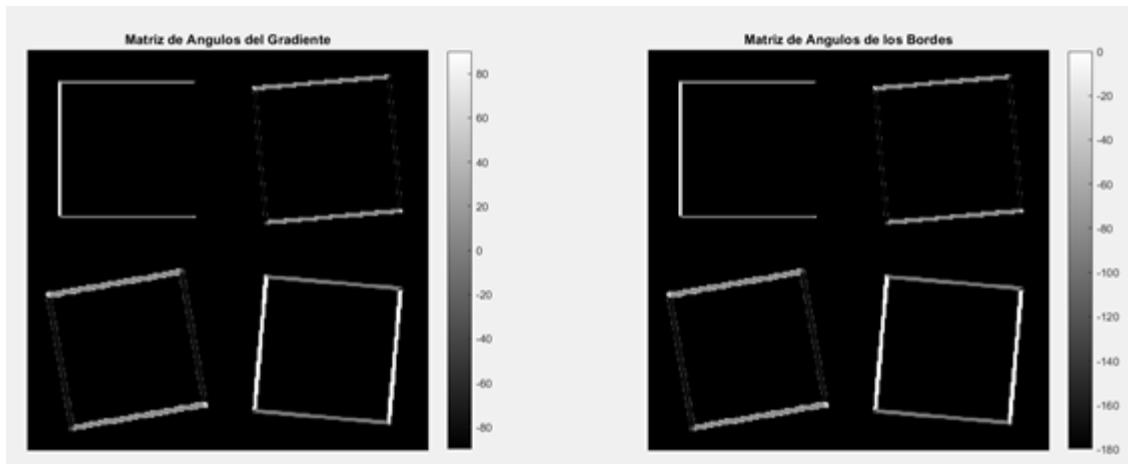


Figura 4. Gráfico de la matriz de ángulos del gradiente y del borde

Mediante el gráfico de esa matriz podremos detectar el rango de valores que toman los ángulos del borde que deseamos segmentar.

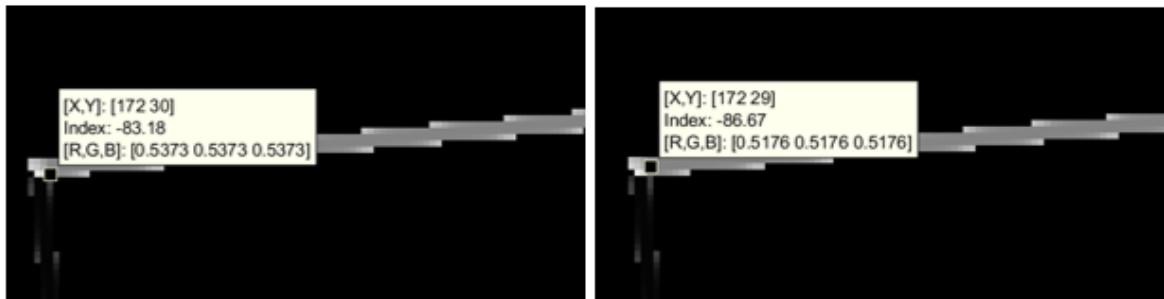


Figura 5. Determinación gráfica de los umbrales superior e inferior

Una vez determinado el intervalo de valores que toman los ángulos del borde, los utilizo como parámetros de la doble umbralización que aplico para segmentar.

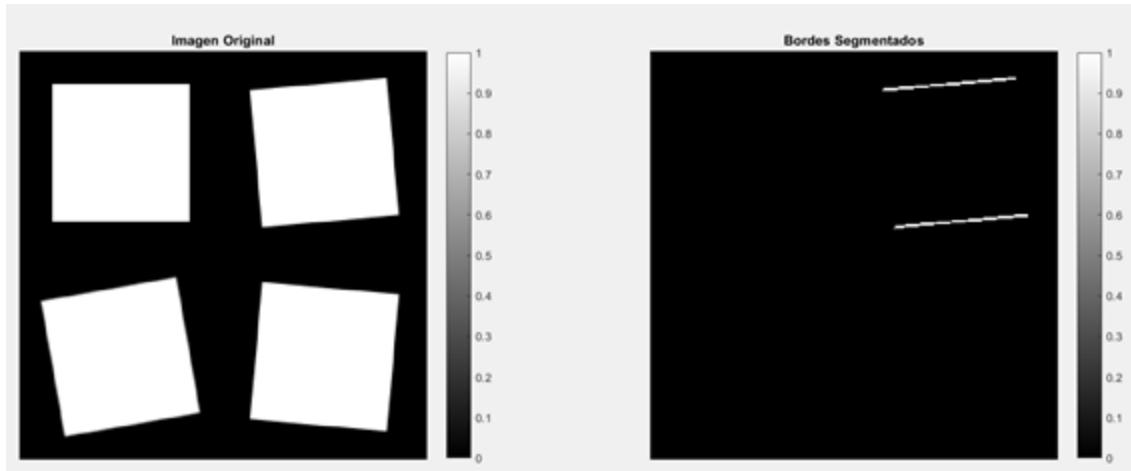


Figura 6. Bordes segmentados

### Problema 3

El objetivo de este problema es armar un rompecabezas a partir de una imagen dada y resolverlo lo más automáticamente posible. Para esto, vamos a usar de referencia una pieza del rompecabezas que llamamos **semilla**. Esta semilla se va a guardar y ubicar en su correspondiente ubicación en el rompecabezas (va a ser la primera pieza bien ubicada).

Para empezar, partimos de la lectura de una imagen. Deberemos tener en cuenta que vamos a dividir la imagen en 16 piezas iguales, por lo que necesitaremos asegurar que las

filas y/o columnas de la matriz (píxeles) que contiene la imagen sean pares. También debemos tener en cuenta los distintos tipos de imágenes con los que nos podríamos encontrar (*grayscale*, *indexed*, *logical*, etc).



Figura 7. Imagen original sin procesar

Lo siguiente será dividir en 16 partes iguales y colocarlos dentro de una celda matricial. Cada elemento de la celda, va a ser una matriz, representando cada pieza. Utilizamos la función ***mat2cell*** para facilitar esta tarea.

Hecho esto, procedemos a mezclar aleatoriamente la imagen con la función creada ***shuffleCell*** con argumentos ***cell\_img*** (celda que contiene las matrices que conforman la imagen), ***N*** (el número total de piezas) y ***n*** (cantidad de filas y columnas). Internamente, *shuffleCell* utiliza las funciones ***randperm*** y ***sort*** para lograr la mezcla.



Figura 8. Imagen original desordenada

Como se mencionó anteriormente, utilizaremos una de estas piezas como semilla y la ubicamos en su posición original haciendo un intercambio de piezas:



Figura 9. Imagen desordenada, ubicando la semilla en su posición original

A partir de ahora, empezamos a buscar los vecinos de la semilla. El criterio que utilizaremos es el análisis de bordes de cada pieza: Se extraen los bordes necesarios de cada pieza para realizar una comparación con la función ***imcolordiff***, que internamente compara matrices de  $m \times n \times 3$ , y nos devuelve como resultado una matriz de diferencias de colores en formato CIELAB con el estándar CIE94.

Utilizamos ***imcolordiff*** en una función desarrollada ***search\_similar\_border*** con los argumentos ***cell\_img*** (celda con las piezas de la imagen), ***coord*** (coordenadas de la pieza que usaremos de referencia), ***border\_dir*** (el borde que analizamos de la pieza de referencia) y ***valid*** (matriz lógica del mismo tamaño de la celda para saber si una pieza ya fue encontrada y colocada en su posición original).

Y dentro de la función ***search\_similar\_border***, usamos otra función desarrollada ***get\_border*** para extraer los bordes deseados de cada pieza, con argumentos ***elem*** (número de fila o columna a extraer), ***row\_or\_col*** (cadena para extraer todas las columnas o todas las filas) y ***M*** (pieza de la imagen para extraer su borde)

Definimos que dos piezas son vecinas si sus bordes tienen la menor diferencia de colores. Lo siguiente es un esquema pensado para ubicar los vecinos:

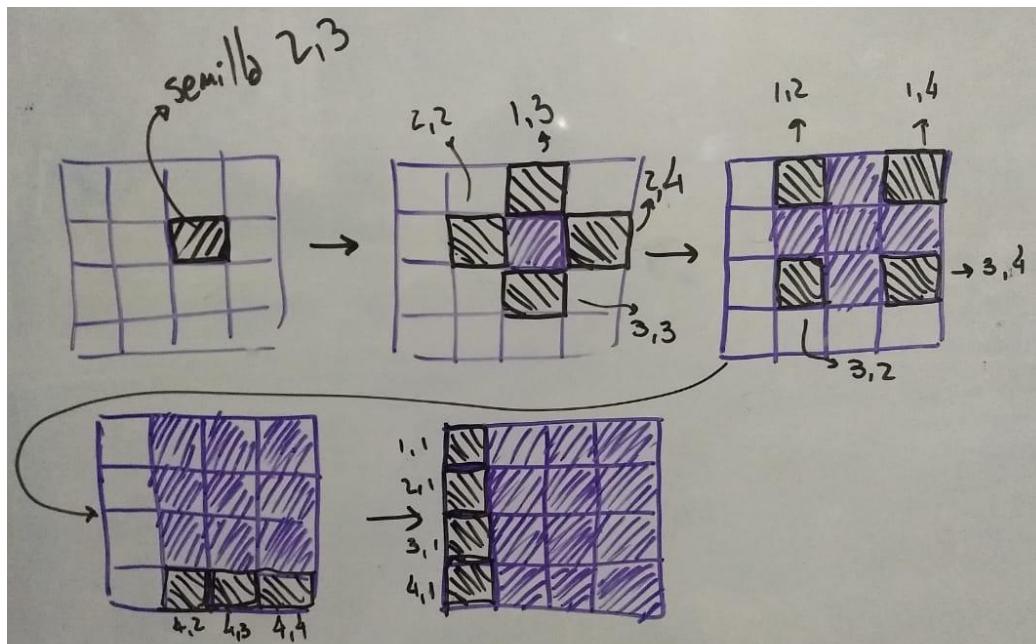


Figura 10. Estrategia para ordenar el rompecabezas a partir de la semilla

Aplicando este criterio, procedemos a hallar los vecinos:

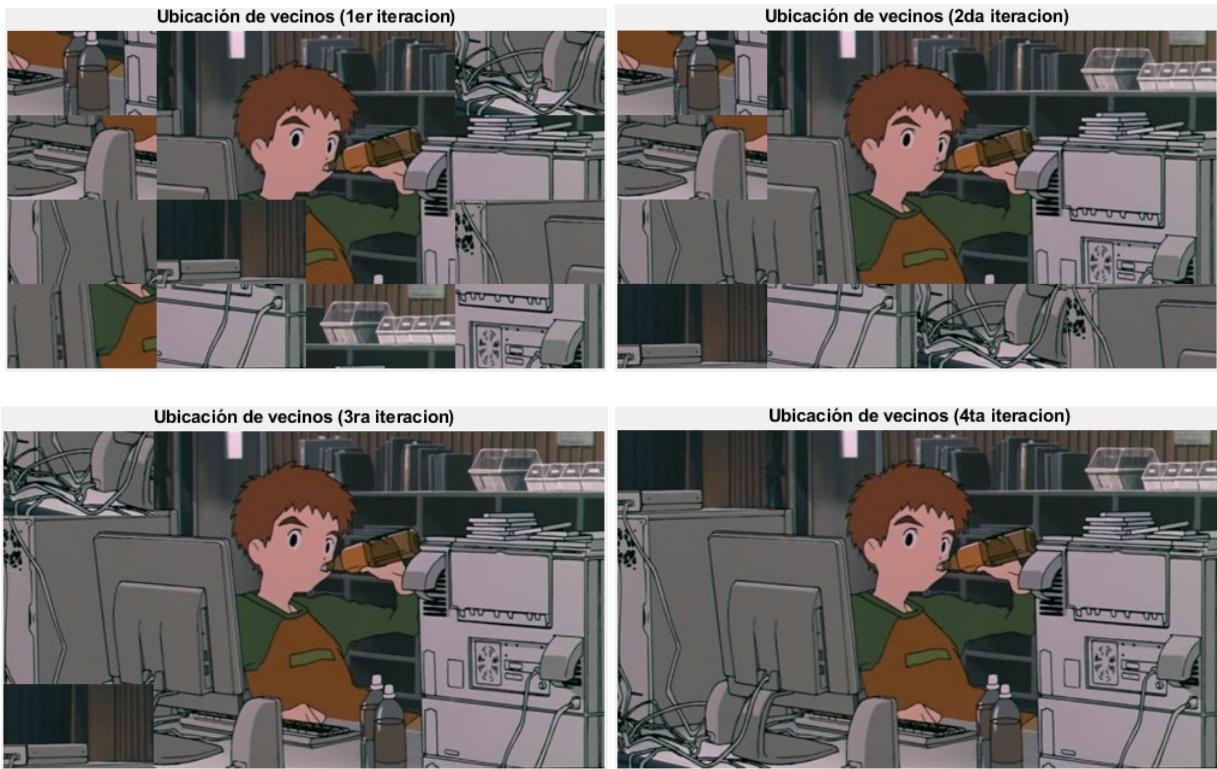


Figura 11. Ordenamiento de la imagen

De esta forma, obtenemos el resultado esperado, la cual es la imagen original.

A continuación, se puede apreciar otra solución en donde hay similitud en bordes que podrían provocar un incorrecto armado del rompecabezas:



Figura 12. 2da imagen desordenada



Figura 13. 2da imagen ordenada

También podemos hacer la prueba en patrones como el siguiente:

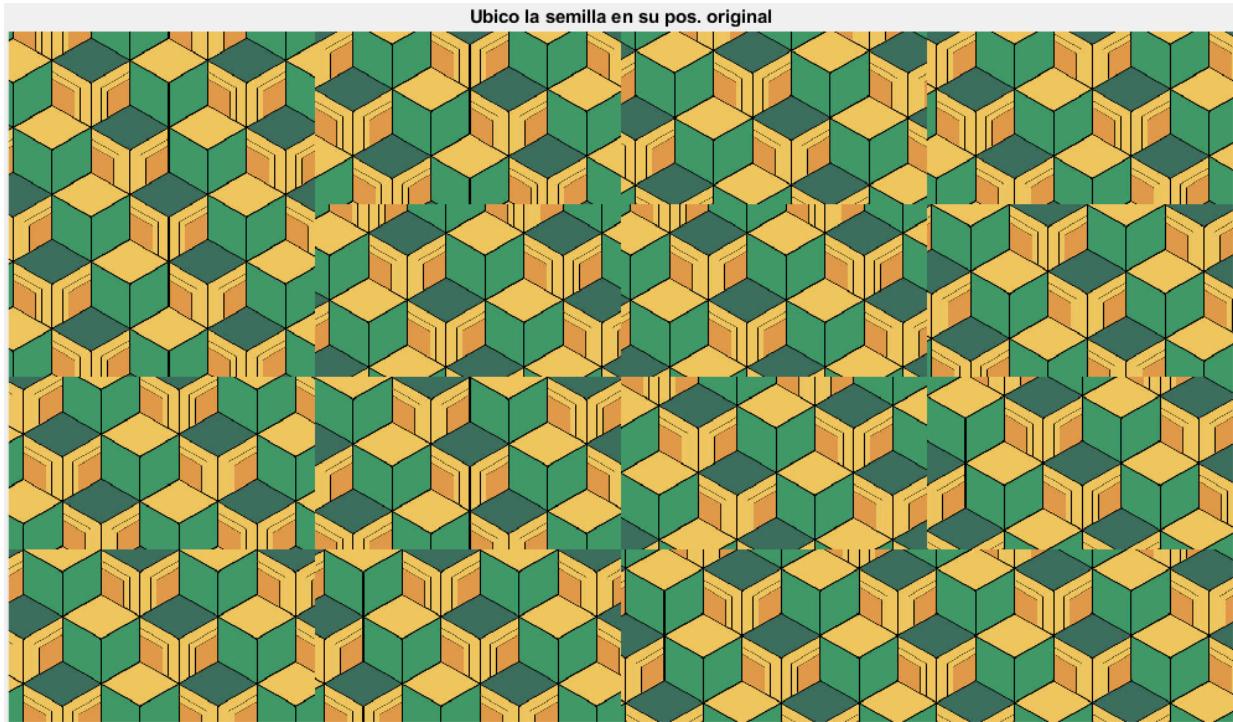


Figura 14. 3era imagen desordenada

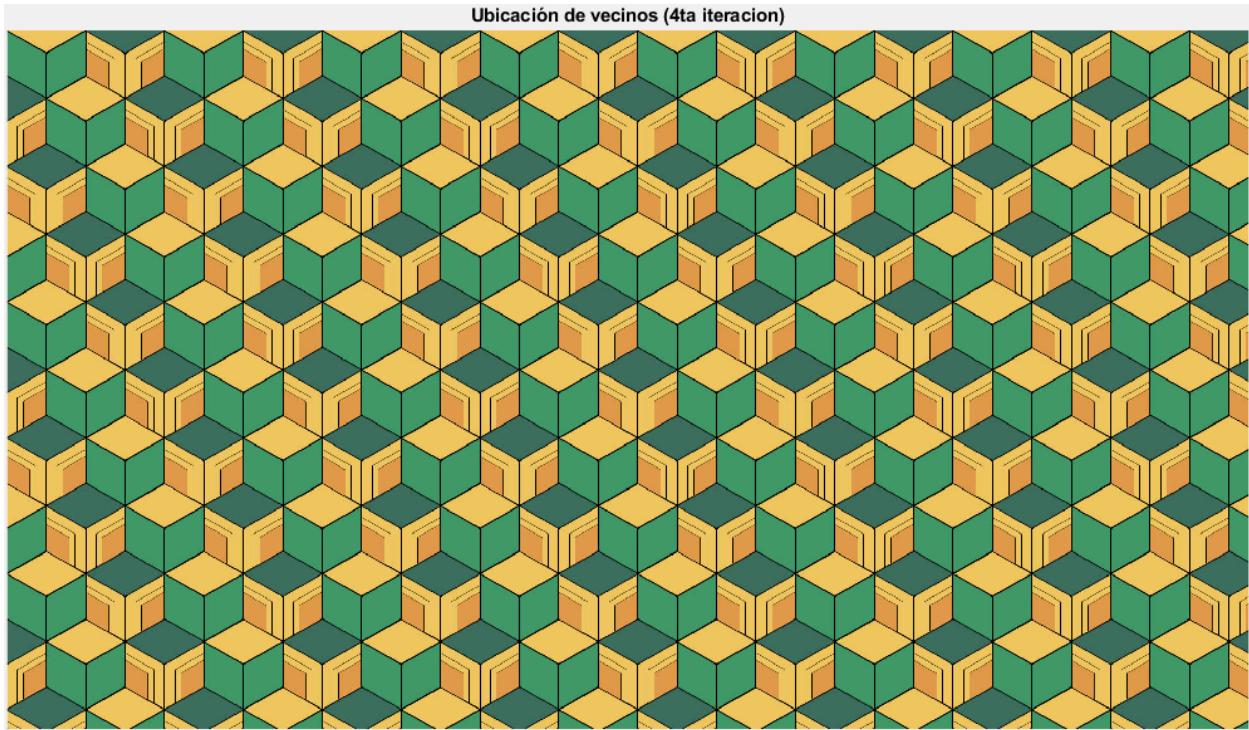


Figura 15. 3era imagen ordenada

Probando otros tipos de imágenes (distintos tamaños y disposición de colores y patrones), se pueden hallar algunas deficiencias del algoritmo. Esto se puede deber a que la división

de piezas para el rompecabezas tendría que ser mayor a 16 para evitar la mayor cantidad posible de detalles similares en los bordes.

Por ejemplo, tenemos esta imagen sacada de una publicación sobre el framework Django:



Figura 16. Imagen en donde el algoritmo falla en la detección de bordes

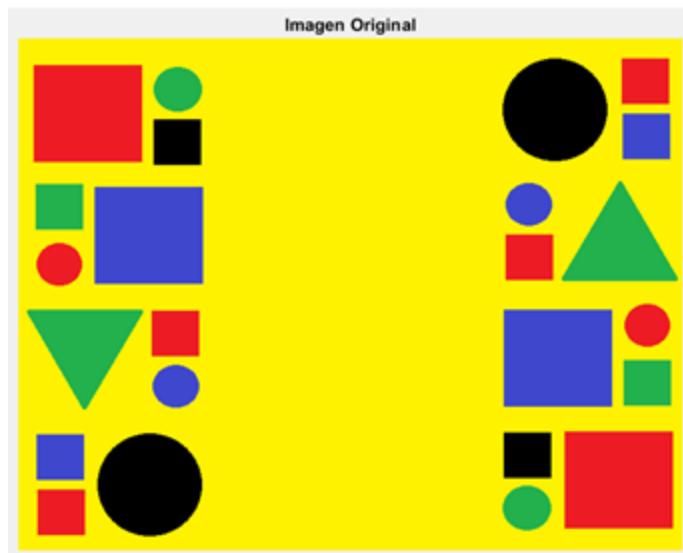
En esta imagen, al realizar la detección de los bordes de cada pieza de la imagen, coinciden en los colores pero en realidad no tiene en cuenta si hay otras opciones válidas. Una posible solución, como se dijo anteriormente, es de particionar la imagen en más de 16 piezas iguales.

Otra solución que se puede aplicar es la de tener en cuenta métodos de medición de error, aunque exigiría mayor complejidad de código.

El algoritmo empleado funciona, pero podría ser mejor aplicar cierta lógica para automatizar aún más el armado del rompecabezas.

## Problema 4

Comenzando generando una imagen con fondo uniforme y figuras geométricas de diferentes tamaños y color.

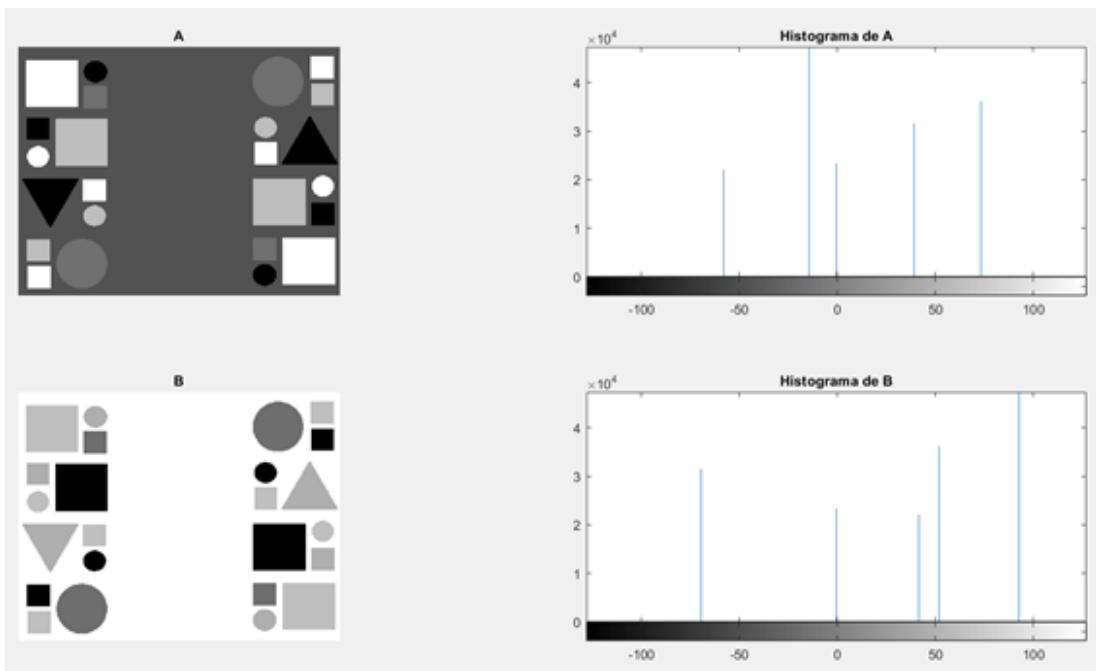




*Figura 17. Imagen generada para el ejercicio*

Posteriormente, procedemos a transformar el espacio de color de la imagen de RGB a LAB, utilizando el comando **rgb2lab**. Definimos al canal uno como la luminancia (L), al dos como el correspondiente a las coordenadas cromáticas rojas/verdes (A) y al tres como las coordenadas cromáticas amarillas y azules (B).

Graficamos los canales A y B junto con sus respectivos histogramas.



*Figura 18. Gráfico de los canales A y B junto con sus histogramas*

Con esta información, podemos determinar los valores del umbral necesarios para segmentar las figuras según su color.

A modo de ejemplo, sabemos que los valores positivos del canal A corresponden a colores rojos y los negativos a verde. Entonces analizando el histograma de A, concluimos que si tomamos como condición que  $A > 73$ , segmentaremos todos los objetos rojos y con  $A < -57$ , los verdes.

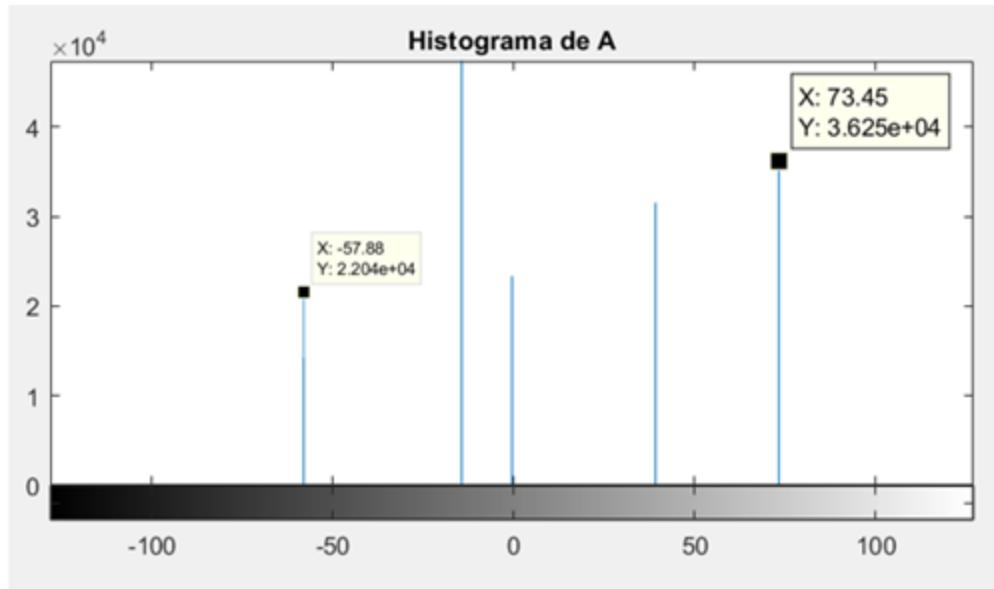


Figura 19. Determinación gráfica de los umbrales para los colores rojos y verdes

Repetimos este procedimiento para segmentar los elementos azules con el canal B. Notar que en el caso del negro podemos poner como condición que el canal L sea igual a 0 ó que los canales A y B sean 0.

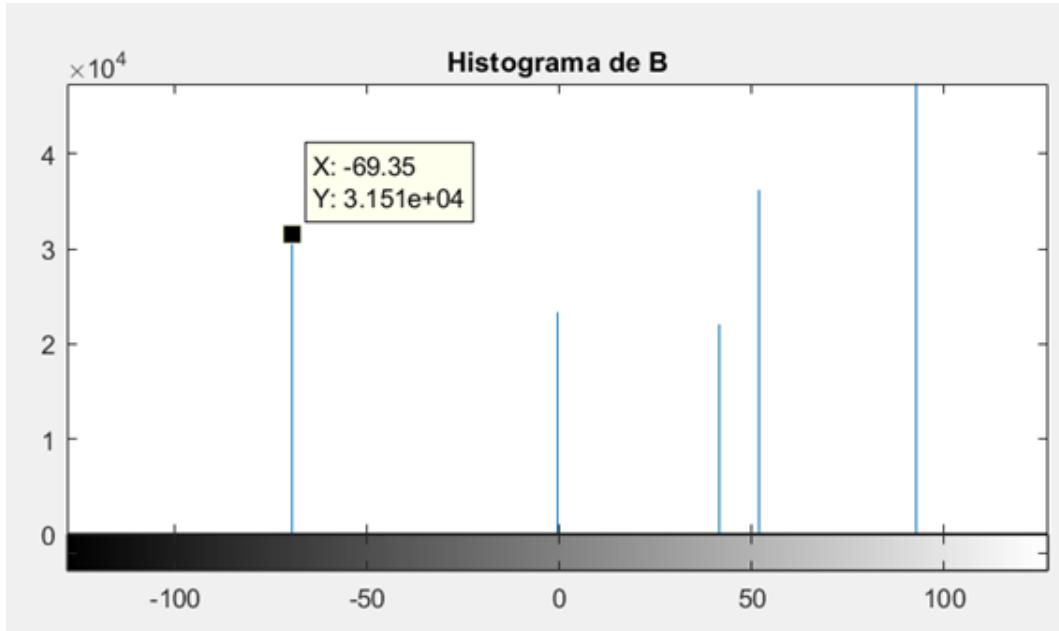


Figura 20. Determinación gráfica del umbral para el color azul

Una vez segmentadas todas las figuras por color, procedemos a crear un mapa de color y a contar la cantidad de elementos en cada conjunto. Para el recuento utilizamos la función

bwlabel, la cual devuelve el número de objetos conectados encontrados en la matriz ingresada.

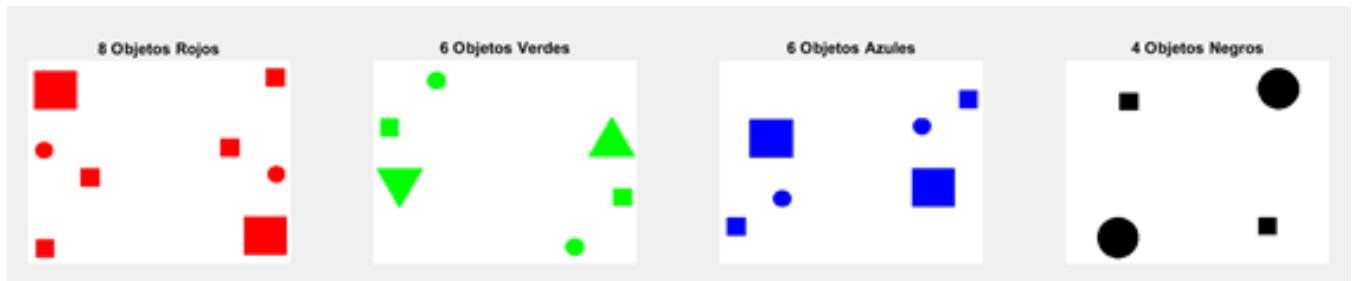


Figura 21. Clasificación y conteo de elementos de cada color

## Anexo

### Scripts

Los scripts para cada ejercicio, con sus correspondientes funciones, están alojados en un repositorio de GitHub: [https://github.com/JaviCeRodriguez/TPF\\_ProcesamientoDeImagenes](https://github.com/JaviCeRodriguez/TPF_ProcesamientoDeImagenes)