FIB Facultat d'Informàtica de Barcelona

Departament d'Enginyeria de Sistemes, Automàtica i Informàtica Industrial
UNIVERSITAT POLITÈCNICA DE CATALUNYA

VISIÓN POR COMPUTADOR

Ejercicio 7 de Laboratorio

Facultad de Informática de Barcelona

Adrian Cristian Crisan

Filip Gedung Dorm

Pablo Vega Gallego

Barcelona, noviembre de 2021

Índice

Introducción

El objetivo de la sesión es implementar un sistema semiautomático para segmentar objetos o animales en imágenes a color. Utilizaremos una segmentación simple por color utilizando kmeans. Las imágenes a tratar presentaran un objeto muy diferenciado del fondo por sus colores. Un ejemplo de estas imágenes y resultados que podríamos esperar son las que aparecen en las siguientes figuras (fig 1 y 2). El procedimiento del programa a realizar sería el siguiente: el usuario indicará manualmente el objeto a segmentar con un marco rectangular (fig 1) y la respuesta de la aplicación será el objeto segmentado indicado con su perfil en rojo.

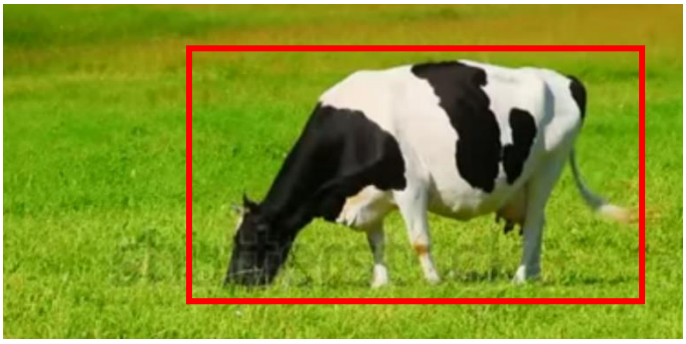

Fig. 1 - El usuario enmarca aproximadamente el animal a segmentar

Fig. 2 – Una posible respuesta del sistema indicada con el perfil en rojo

Metodología

Abrimos la imagen y comprobamos que es una imagen RGB, utilizando la función size.

```
imagen_original = imread('nenufar.jpg');
tamano = size(imagen_original)
```

```
  tamano = 1×3
           873        1200           3
```

Vemos que la imagen es RGB, se nos muestra en la tercera columna del vector devuelto.

Mostramos la imagen por pantalla

```
imshow(imagen_original)
```

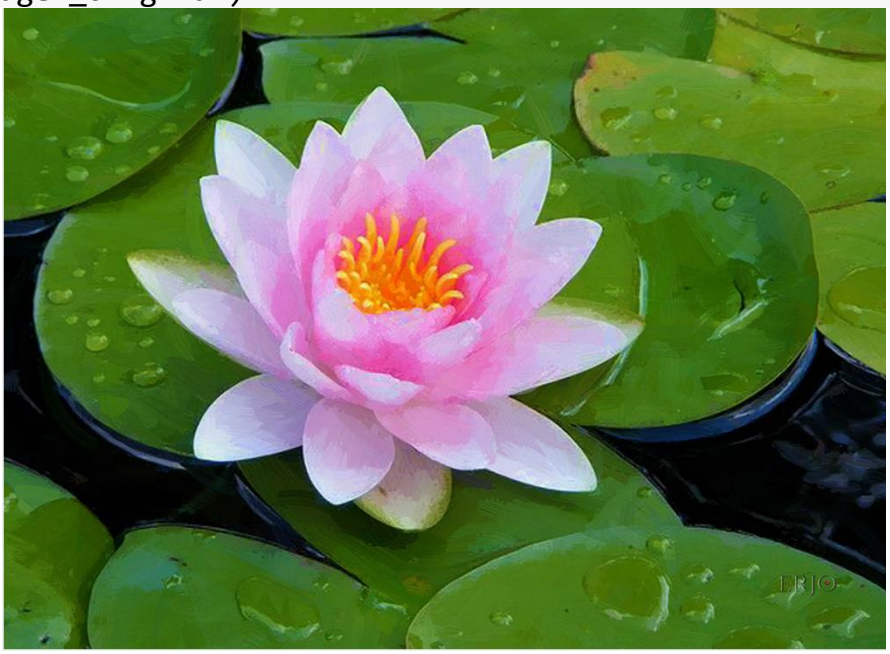

Fig. 3 – Flor que destaca fácilmente de su entorno por sus colores

Obtenemos el rectángulo que enmarca el objeto (fig. 4) que se quiere segmentar.

```
rect = getrect; % (x, y, w, h)
```

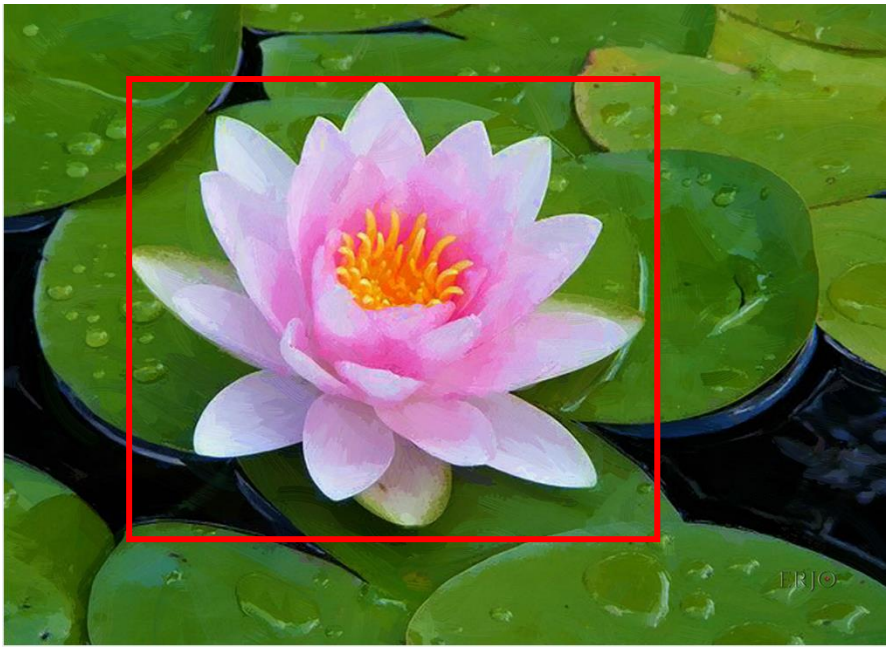

Fig. 4 – Imagen enmarcada por el usuario

Obtenemos la imagen HSV

```
HSV = rgb2hsv(imagen_original);
```

Preparamos la tabla kmeans. Hay que construir una tabla **O** con tantes files como pixeles y las siguientes columnas:

[Hx Hy S V]

Donde Hx Hy son las coordenades circulares 2D del hue.

```
[f, c, w] = size(HSV);

Hue = HSV(:, :, 1) * 2 * pi;  % |Hy/H
Huex = cos(Hue);              % | /
Huey = sin(Hue);              % |/_________Hx

S = HSV(:, :, 2);
V = HSV(:, :, 3);

Hx = reshape(Huex, [tamano(1) * tamano(2), 1]);
Hy = reshape(Huey, [tamano(1) * tamano(2), 1]);
S = reshape(S, [tamano(1) * tamano(2), 1]);
V = reshape(V, [tamano(1) * tamano(2), 1]);
```

Agrupamos los colores en **k** clases con kmeans y obtenemos la classificación **C.**

```
O = [Hx Hy S V];
k = 20;


C = kmeans(O, k);
```

Obtenemos el labelling de los píxeles en pseudo-color.

```
IC = reshape(C,f,c);
```

```
rgb = label2rgb(IC);
imshow(rgb);
```

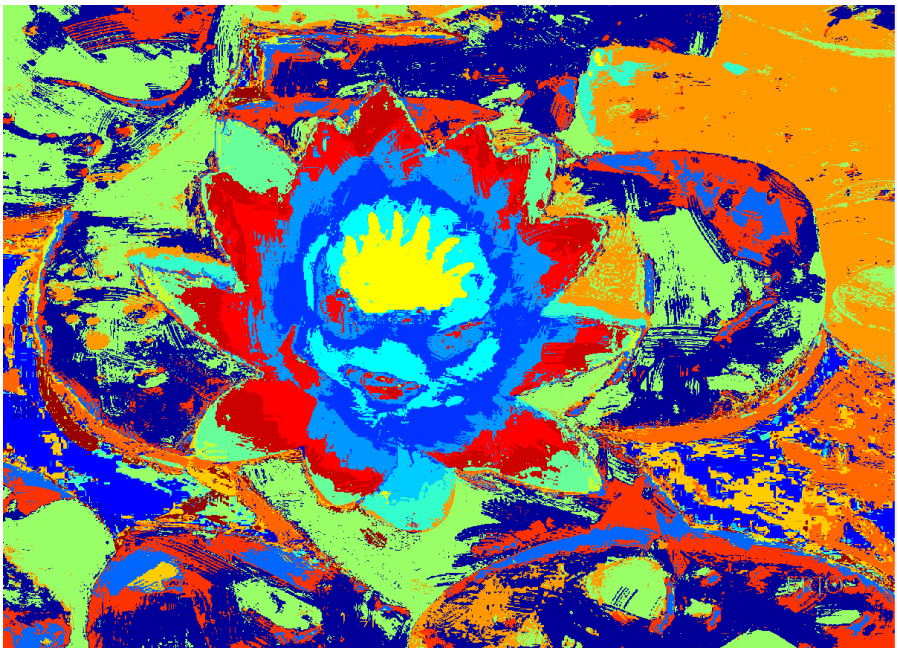

Fig. 5 – Labelling de los píxeles en pseudo-color

Para ver que colores caen dentro del rectángulo y cuales fuera, construimos una imagen de valores booleanos, llamada **MASK** del mismo tamaño que **I** con los valores a cero si está fuera del rectángulo y uno si están dentro (fig. 6).

```
MASK = false(f, c);
MASK(floor(rect(2):rect(2)+rect(4)), floor(rect(1):rect(1)+rect(3)))
= true;
imshow(MASK);
```

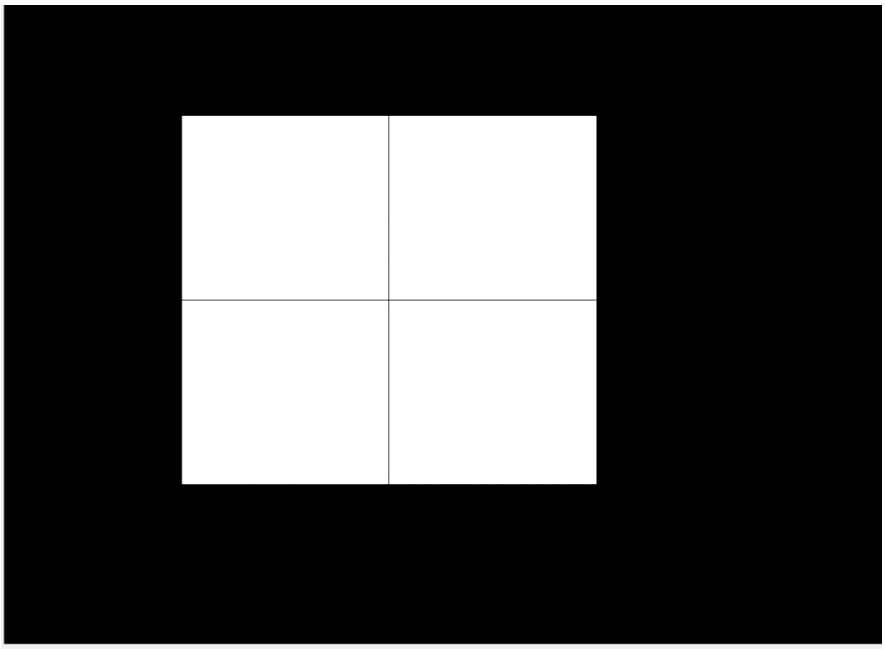

Fig. 6 – Máscara que nos indica si un píxel está dentro o no del marco

Construimos un vector **H** que indica si un color cae dentro del rectángulo o no.

```
H = [C, MASK(:)];
```

A continuación, contamos por cada color **C** cuántos píxeles han caído fuera y cuantos dentro. Guardamos los resultados en dos arrays **Hist0** e **Hist1**.

```
Hist0 = zeros(1, k);
Hist1 = zeros(1, k);

for i = 1:f*c
    pos = H(i, 1);
    res = H(i, 2);

    if res
        Hist1(pos) = Hist1(pos) + 1;
    else
        Hist0(pos) = Hist0(pos) + 1;
    end

end
```

Decidimos si un representante de color pertenece a la figura que se quiere segmentar comparando sus apariciones dentro y fuera del rectángulo. Guardamos la decisión en un vector llamado **RES**.

```
RES = Hist1 > Hist0;
```

Decidimos para cada píxel de la matriz **H** si forma parte de la figura o no utilizando la información que contiene el vector **RES**. Guardamos el resultado en un vector **M** y lo mostramos por pantalla (fig. 6).

```
M = zeros(f*c, 1);
for i = 1:f*c
    pos = H(i, 1);
    if RES(pos)
        M(i) = 1;
    end
end
M = reshape(M, [f, c]);

imshow(M);
```

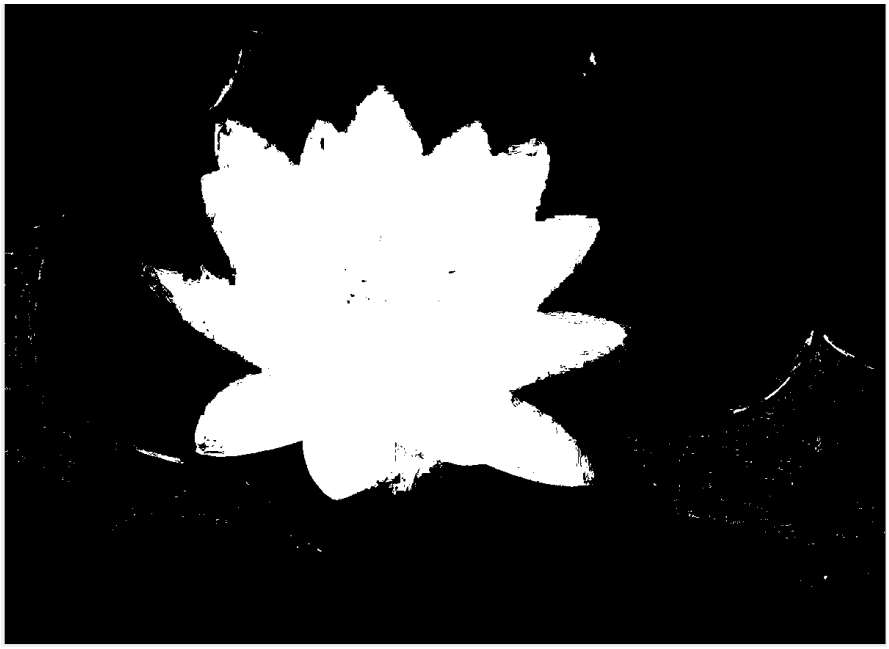

Fig. 7 – Resultado provisional

Por último, filtramos la imagen para que el resultado no tenga ruido y sea más compacta.

Si añadimos la siguiente instrucción obtenemos como resultado sólo lo que hay dentro del rectángulo seleccionado por el usuario al inicio de la ejecución, si no, aparecen resultados finales que seleccionan también objetos fuera del rectángulo como podemos ver en la figura 14.

```
M = M & MASK;
```

Pero con esta instrucción el resultado final queda menos perfilado, por lo que hemos decidido no añadir esta instrucción al código final, pero no parecía relevante comentar esta posibilidad.

Eliminamos pequeñas imperfecciones del resultado:

```
SE = strel('disk', 6);
imagen_cerrada = imopen(M, SE);
imshow(imagen_cerrada);
```

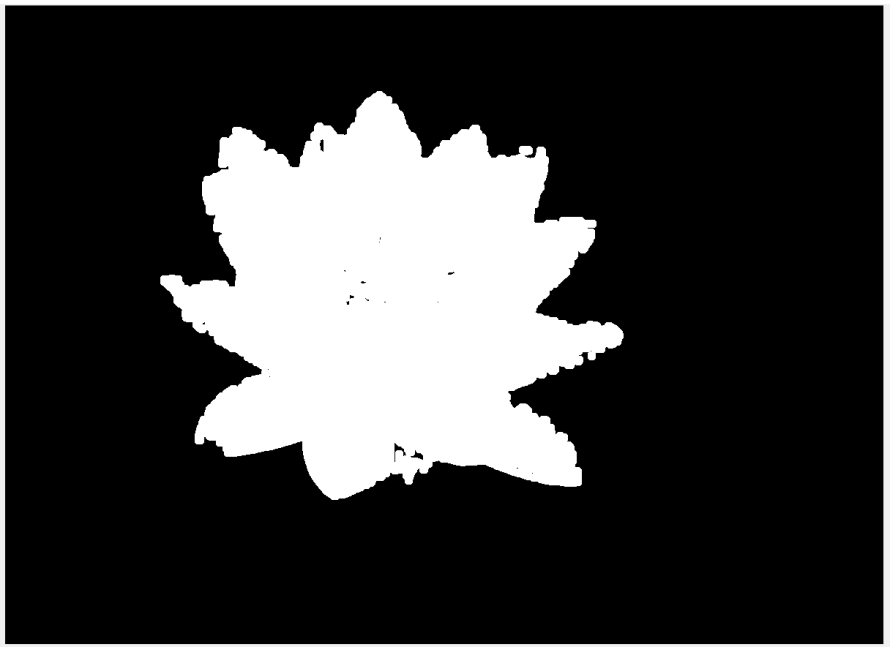

Fig. 8 – Resultado quitando el ruido de fondo

```
SE1 = strel('disk', 50);
imagen_cerrada = imclose(imagen_cerrada, SE1);
imshow(imagen_cerrada);
```

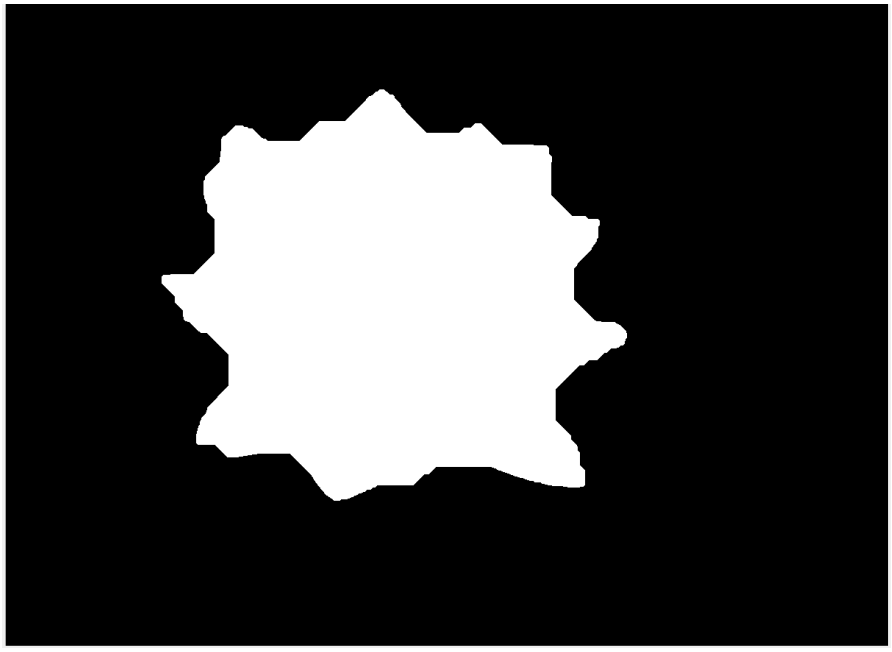

Fig. 9 – Resultado rellenando los huecos

Indicamos la respuesta con un perfil en rojo.

```
disk = strel('disk',5);
imagen_casi_bordes = imerode(imagen_cerrada,disk);

bordes = imagen_cerrada - imagen_casi_bordes;

I = imoverlay(imagen_original, bordes, 'red');
imshow(I);
```

Fig. 10 – Resultado final

Prueba con otras imágenes

Fig. 11 – Imagen original (prueba nº2)

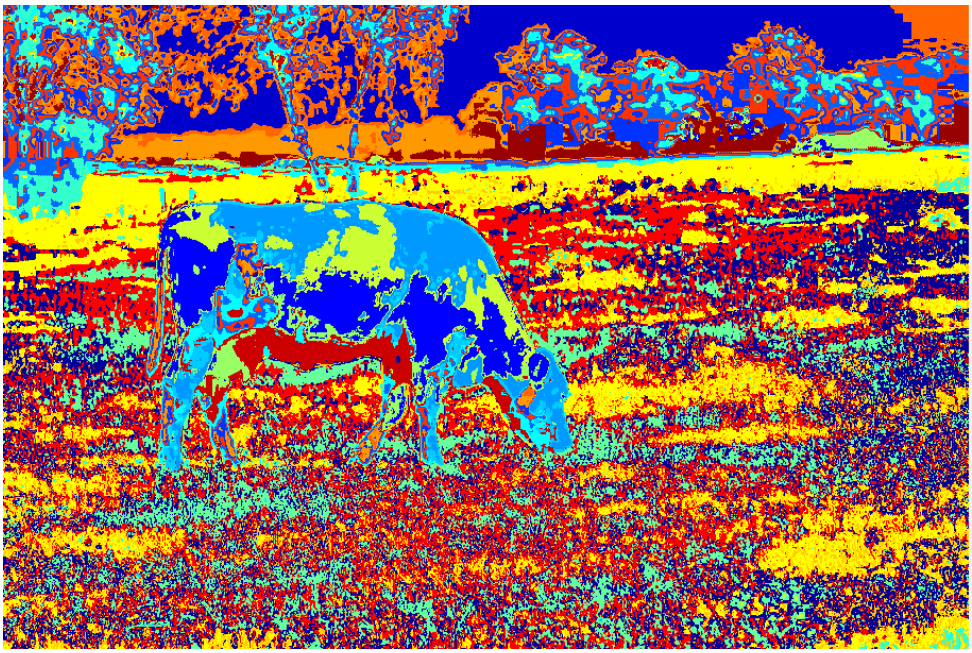

Fig. 12 – Labelling de la imagen 2 de los píxeles en pseudo-color

Fig. 13 – Resultado provisional de la imagen 2

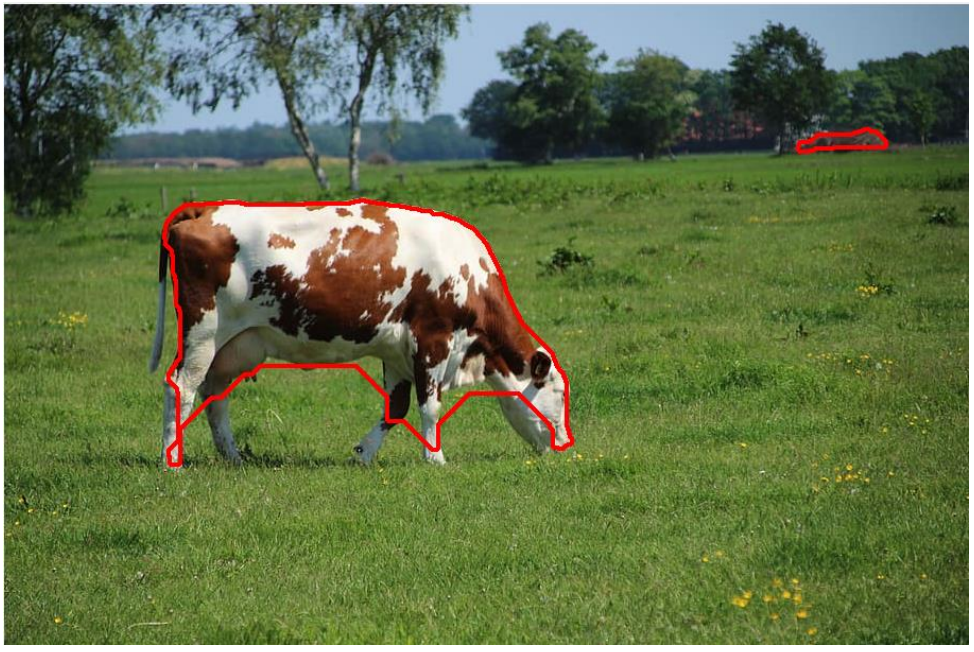

Fig. 14 – Resultado final de la imagen 2

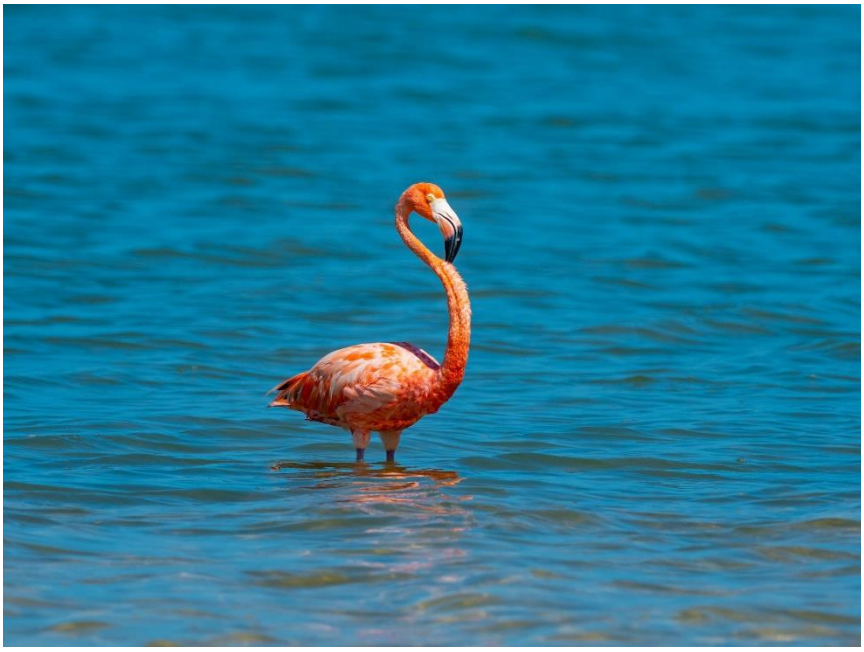

Fig. 15 – Imagen original (prueba nº3)

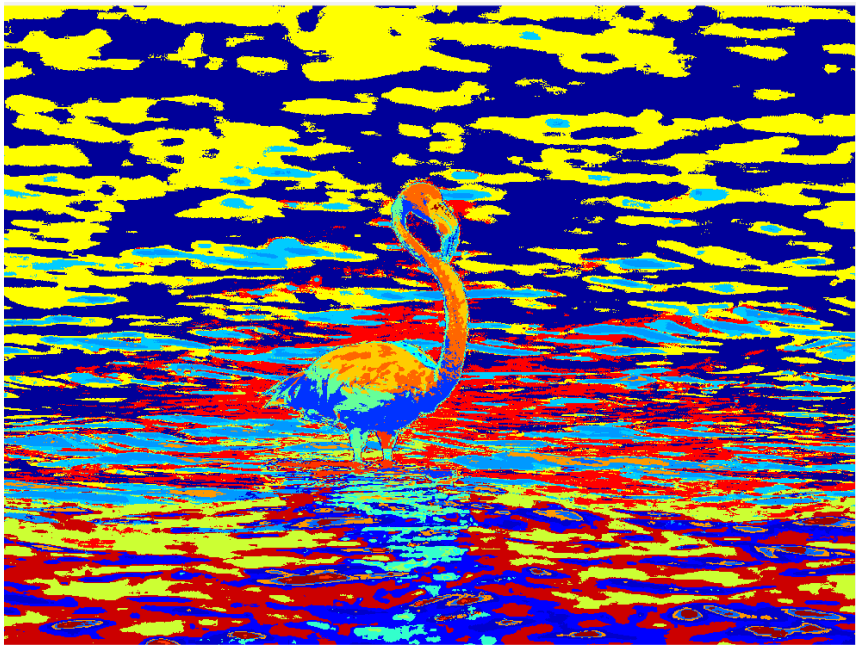

Fig. 16 – Labelling de la imagen 3de los píxeles en pseudo-color

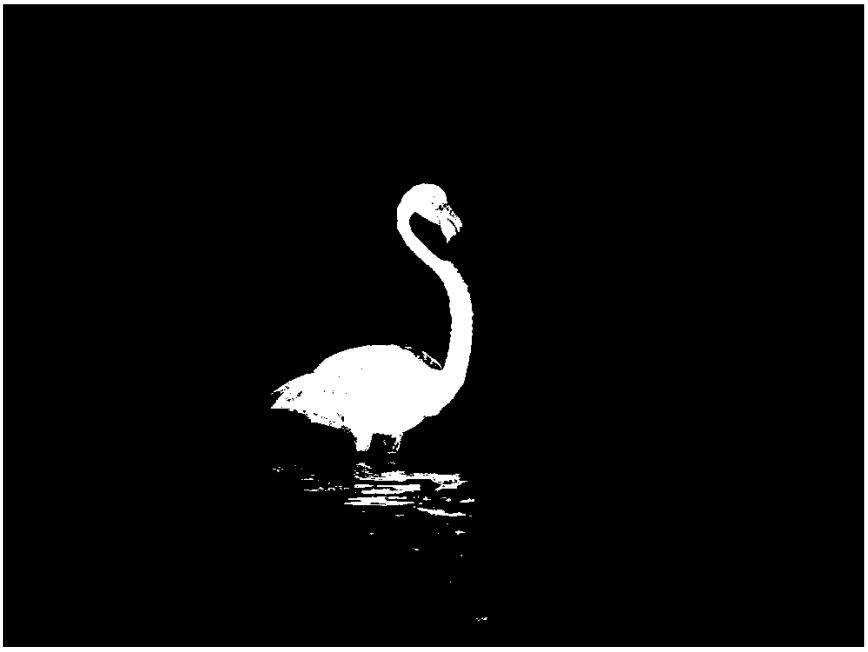

Fig. 17 – Resultado provisional de la imagen 3

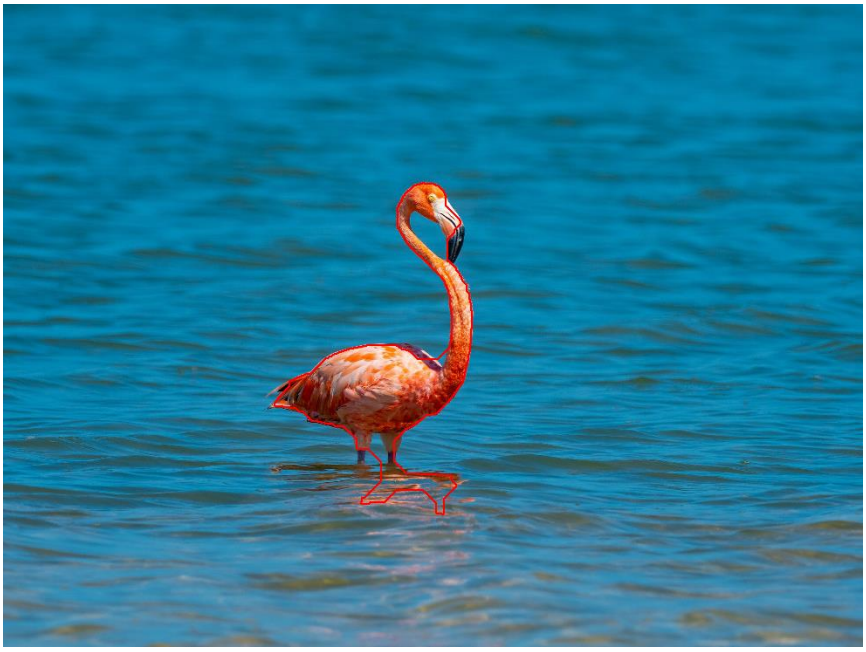

Fig. 18 – Resultado final de la imagen 3