

Segmentación de imágenes usando espacios de color

Importamos librerías:

```
In [49]: import cv2
import numpy as np
import matplotlib.pyplot as plt
import colorsys
import skimage
import plotly.graph_objects as go
from pathlib import Path
```

```
In [48]: ASSETS_FOLDER_PATH = "./assets"
OUTPUT_FOLDER_PATH = "./results"
Path(OUTPUT_FOLDER_PATH).mkdir(parents=True, exist_ok=True)
```

Definimos las funciones a utilizar:

```
In [4]: def plot_image(img, title):
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.title(title)
plt.show()
```

```
In [5]: def plot_images(img1, title1, img2, title2):
fig = plt.figure(figsize=(6, 7))

fig.add_subplot(1, 2, 1)
plt.imshow(img1, cmap='gray')
plt.axis('off')
plt.title(title1)

fig.add_subplot(1, 2, 2)
plt.imshow(img2, cmap='gray')
plt.axis('off')
plt.title(title2)

plt.subplots_adjust(wspace=0.05, hspace=0)
plt.show()
```

Ej 1.a - Finding Nemo

Espacios de color al leer imágenes

Guardamos en `flags` las posibles conversiones de espacios de colores que nos ofrece OpenCV:

```
In [6]: flags = [i for i in dir(cv2) if i.startswith('COLOR_')]
print(f"len: {len(flags)}")
```

```
print(f"first 3: {flags[:3]}")
```

len: 374

first 3: ['COLOR_BAYER_BG2BGR', 'COLOR_BAYER_BG2BGRA', 'COLOR_BAYER_BG2BGR_EA']

donde el 2 separa los espacios de colores origen y destino.

Al abrir la imagen del pez payaso, vemos que tiene los canales rojo y azul invertidos:

```
In [7]: nemo_bgr = cv2.imread('./assets/nemo_images/nemo0.jpg')  
plot_image(nemo_bgr, 'Nemo BGR')
```

Nemo BGR



Esto es porque OpenCV lee por default las imágenes en formato BGR. Podemos arreglarlo con `cvtColor(image, flag)` especificando el flag que convierte de BGR a RGB:

```
In [8]: nemo_rgb = cv2.cvtColor(nemo_bgr, cv2.COLOR_BGR2RGB)  
plot_image(nemo_rgb, 'Nemo RGB')
```

Nemo RGB



RGB vs HSV

Leemos la imagen en formato HSV:

```
In [9]: nemo_hsv = cv2.cvtColor(nemo_bgr, cv2.COLOR_BGR2HSV)
```

Comparamos la distribución de colores de los pixeles de los espacios de color de RGB y HSV:

```
In [10]: def rgb_3d_scatter(img, title):
r, g, b = cv2.split(img)
pixel_colors = img.reshape((np.shape(img)[0]*np.shape(img)[1], 3)) / 255.0
fig = go.Figure(data=[go.Scatter3d(x=r.flatten(), y=g.flatten(), z=b.flatten(),
mode='markers',
marker=dict(color=pixel_colors, size=2))])

fig.update_layout(title=title,
scene=dict(xaxis_title='Red', yaxis_title='Green', zaxis_title='Blue'),
scene_camera = dict(eye=dict(x=1.5, y=-1.5, z=0.5), center=dict(x=0, y=0, z=0),
margin=dict(t=60, b=35, l=25, r=0))

fig.show()
```

```
In [45]: def hsv_3d_scatter(img, title):
h, s, v = cv2.split(img)
rgb_values = [
    colorsys.hsv_to_rgb(h_val/179, s_val/255, v_val/255)
    for h_val, s_val, v_val in zip(h.flatten(), s.flatten(), v.flatten())
]

fig = go.Figure(data=[go.Scatter3d(x=h.flatten(), y=s.flatten(), z=v.flatten(),
mode='markers',
marker=dict(color=rgb_values, size=2))])

fig.update_layout(title=title,
scene=dict(xaxis_title='Hue', yaxis_title='Saturation', zaxis_title='Value'),
scene_camera = dict(eye=dict(x=1.5, y=-1.5, z=1), center=dict(x=0, y=0, z=0),
margin=dict(t=60, b=35, l=25, r=0))

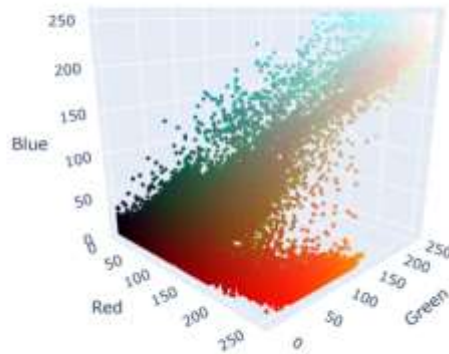
fig.show()
```

```
margin=dict(t=60, b=35, l=25, r=0))  
fig.show()
```

```
In [44]: rgb_3d_scatter(nemo_rgb, 'Nemo RGB pixels scatter plot')
```

```
In [51]: my_image = skimage.io.imread(fname=f"{OUTPUT_FOLDER_PATH}/scatter_rgb_nemo.jpg")  
plot_image(my_image, '')
```

Nemo RGB pixels scatter plot

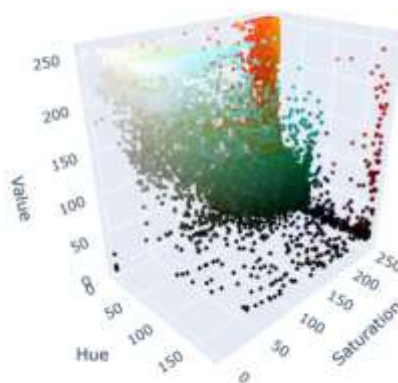


De esta forma, podemos notar por ejemplo que los pixeles naranjas con distintos niveles de brillo o saturación se encuentran repartidos en el espacio RGB, complicando el poder hallar un área definida que los englobe para separarlos del resto.

```
In [13]: hsv_3d_scatter(nemo_hsv, 'Nemo HSV pixels scatter plot')
```

```
In [52]: my_image = skimage.io.imread(fname=f"{OUTPUT_FOLDER_PATH}/scatter_hsv_nemo.jpg")  
plot_image(my_image, '')
```

Nemo HSV pixels scatter plot



Con HSV podemos separar mejor los naranjas, ya que se encuentran localizados en un área determinada. Esta ventaja se debe a que el espacio RGB no tiene una separación clara entre la información de color y la de brillo, lo que puede complicar la segmentación cuando hay variaciones de iluminación como vimos con los naranjas. El canal Hue en

HSV permite identificar colores con mayor precisión sin importar su nivel de brillo o saturación, permitiendo segmentar más fácilmente.

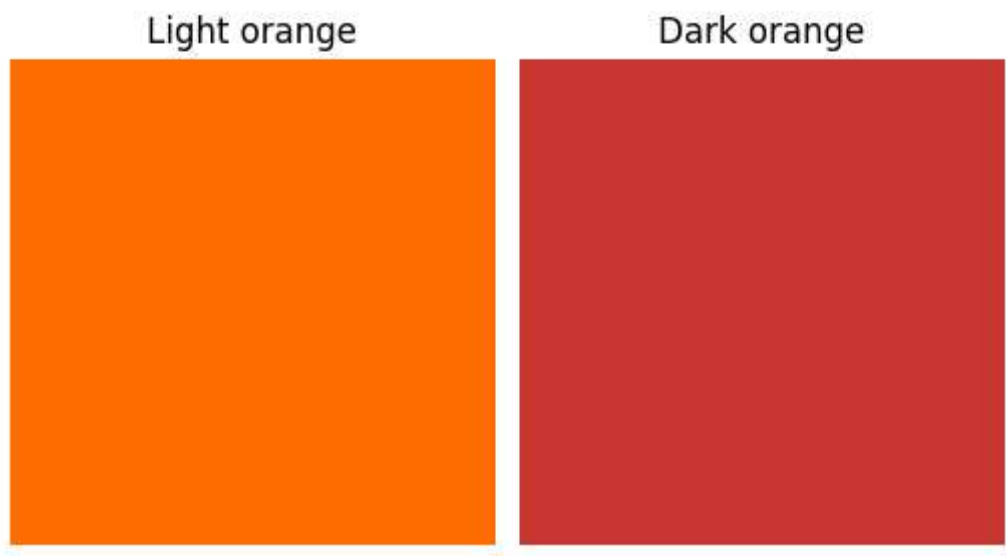
Creamos las máscaras

```
In [14]: from matplotlib.colors import hsv_to_rgb
```

```
In [15]: def plot_colors(color1, title1, color2, title2):
    lo_square = np.full((10, 10, 3), color1, dtype=np.uint8) / 255.0
    do_square = np.full((10, 10, 3), color2, dtype=np.uint8) / 255.0
    plt.subplot(1, 2, 1)
    plt.imshow(hsv_to_rgb(do_square))
    plt.axis('off')
    plt.title(title1)
    plt.subplot(1, 2, 2)
    plt.imshow(hsv_to_rgb(lo_square))
    plt.axis('off')
    plt.title(title2)
    plt.subplots_adjust(wspace=0.05, hspace=0)
    plt.show()
```

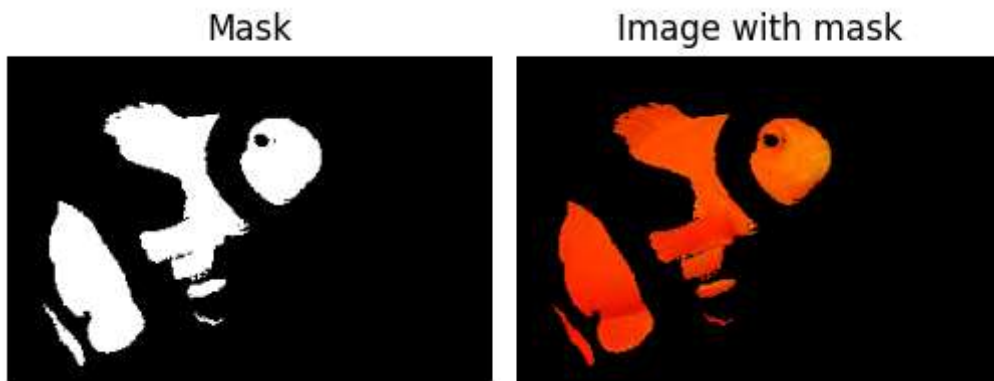
Tomamos dos colores como límites inferior y superior:

```
In [16]: light_orange = (1, 190, 200)
    dark_orange = (18, 255, 255)
    plot_colors(light_orange, "Light orange", dark_orange, "Dark orange")
```



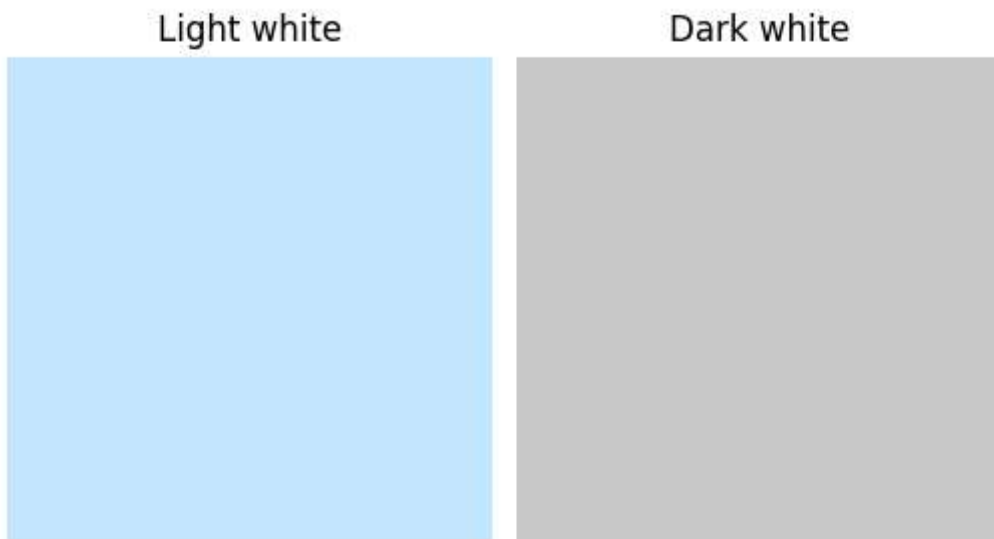
```
In [17]: def plot_mask_result(mask, result):
    plt.subplot(1, 2, 1)
    plt.imshow(mask, cmap="gray")
    plt.title("Mask")
    plt.axis('off')
    plt.subplot(1, 2, 2)
    plt.imshow(result)
    plt.axis('off')
    plt.title("Image with mask")
    plt.subplots_adjust(wspace=0.05, hspace=0)
    plt.show()
```

```
In [18]: mask = cv2.inRange(nemo_hsv, light_orange, dark_orange)
result = cv2.bitwise_and(nemo_rgb, nemo_rgb, mask=mask)
plot_mask_result(mask, result)
```

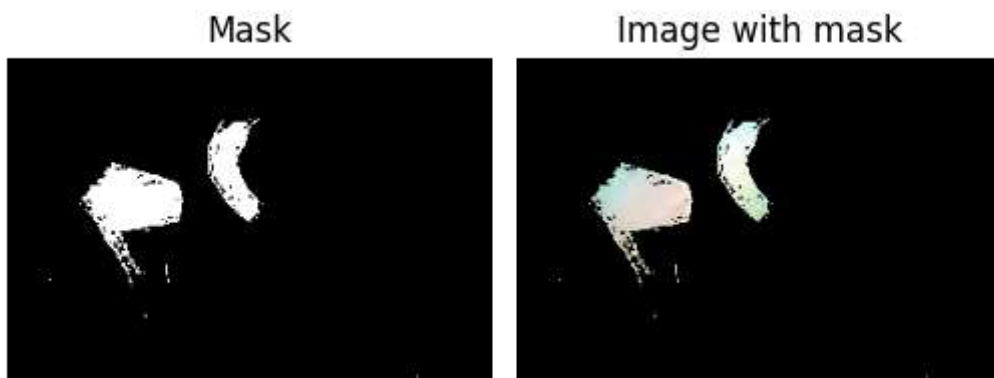


Aislamos también las rayas de Nemo creando una máscara para colores blancos:

```
In [19]: light_white = (0, 0, 200)
dark_white = (145, 60, 255)
plot_colors(light_white, "Light white", dark_white, "Dark white")
```

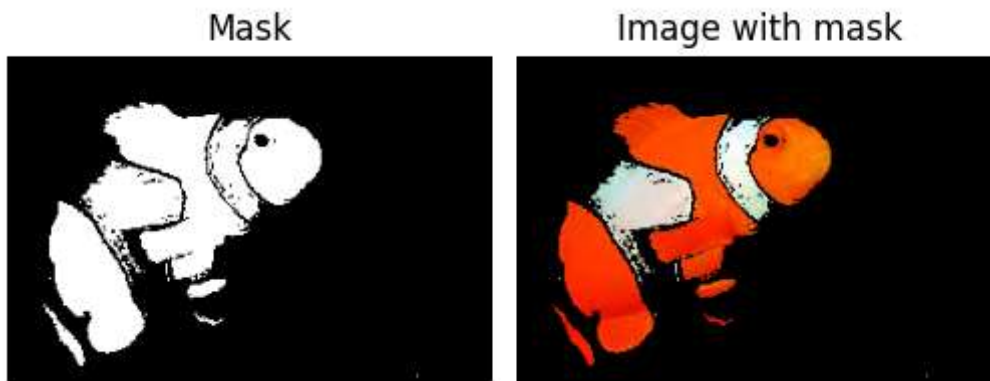


```
In [20]: mask_white = cv2.inRange(nemo_hsv, light_white, dark_white)
result = cv2.bitwise_and(nemo_rgb, nemo_rgb, mask=mask_white)
plot_mask_result(mask_white, result)
```



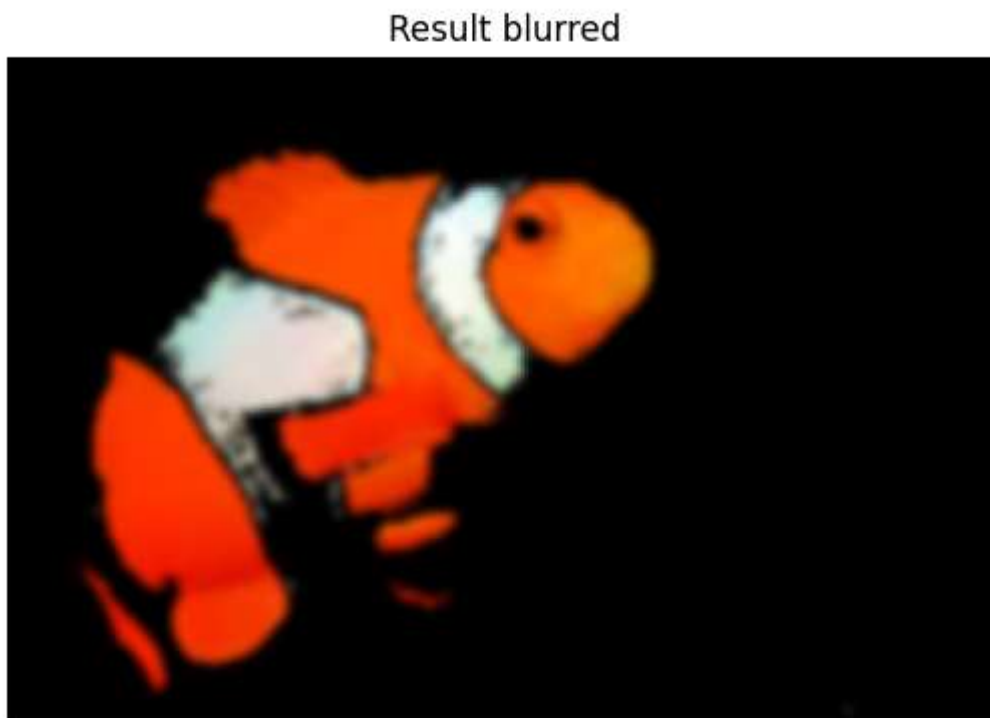
Combinamos las máscaras:

```
In [21]: final_mask = mask + mask_white
result = cv2.bitwise_and(nemo_rgb, nemo_rgb, mask=final_mask)
plot_mask_result(final_mask, result)
```



Para mejorar los bordes, aplicamos Gaussian Blur:

```
In [22]: result_blur = cv2.GaussianBlur(result, (7, 7), 0)
plot_image(result_blur, "Result blurred")
```



Aplicación de la segmentación a otras imágenes

```
In [23]: def segment_clownfish(image):
hsv_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
mask = cv2.inRange(hsv_image, light_orange, dark_orange)
mask_white = cv2.inRange(hsv_image, light_white, dark_white)
final_mask = mask + mask_white
result = cv2.bitwise_and(image, image, mask=final_mask)
blur = cv2.GaussianBlur(result, (7, 7), 0)
return blur
```

```
In [24]: path = "./assets/nemo_images/nemo"
```



```
nemos_friends = []
for i in range(6):
    friend = cv2.cvtColor(cv2.imread(path + str(i) + ".jpg"), cv2.COLOR_BGR2RGB)
    nemos_friends.append(friend)

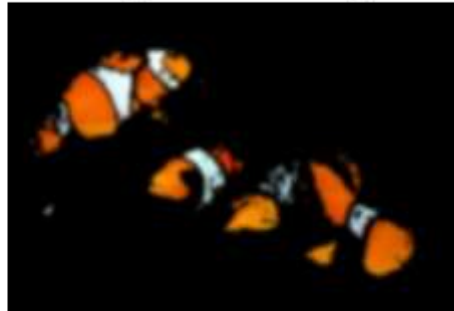
results = [segment_clownfish(friend) for friend in nemos_friends]
```

```
In [25]: for i in range(1,6):
        plot_images(nemos_friends[i], 'Original image', results[i], 'Segmented image')
```

Original image



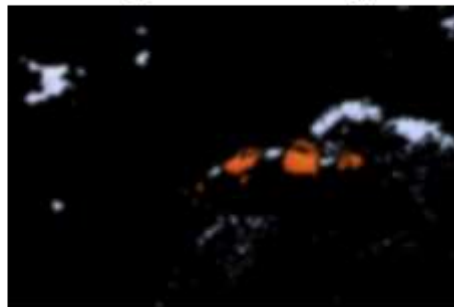
Segmented image



Original image



Segmented image



Original image



Segmented image



Original image



Segmented image



Original image



Segmented image



En general los resultados fueron buenos, pero al generalizar el rango de colores sobre determinada imagen con cierta iluminación y fondo, algunas generalizaciones fallan.

Ej 1.b: Segmentación de un pájaro

Aplicamos lo visto a otra imagen:

```
In [26]: bird_bgr = cv2.imread('./assets/bird.jpg')
bird_bgr = cv2.resize(bird_bgr, (0,0), fx=0.25, fy=0.25)
bird_rgb = cv2.cvtColor(bird_bgr, cv2.COLOR_BGR2RGB)
plot_image(bird_rgb, 'Bird')
```

Bird

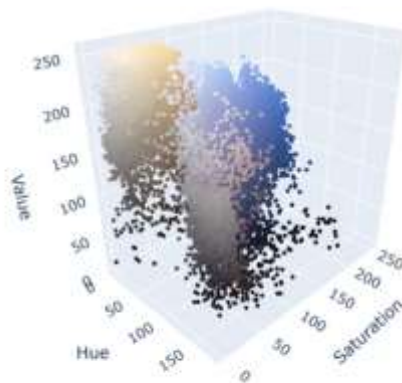


Visualizamos los píxeles en el espacio HSV:

```
In [27]: bird_hsv = cv2.cvtColor(bird_bgr, cv2.COLOR_BGR2HSV)
hsv_3d_scatter(bird_hsv, 'Bird HSV pixels scatter plot')
```

```
In [53]: my_image = skimage.io.imread(fname=f"{OUTPUT_FOLDER_PATH}/scatter_hsv_bird.jpg")
plot_image(my_image, '')
```

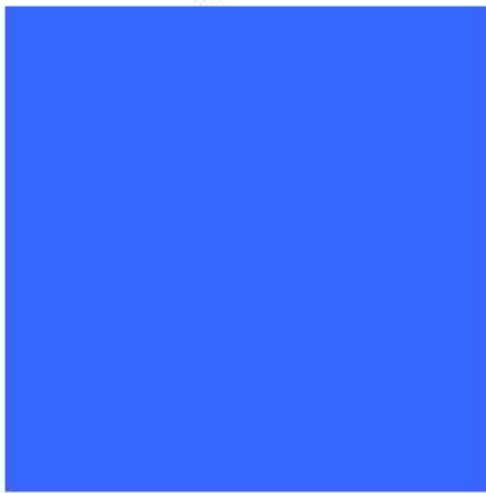
Bird HSV pixels scatter plot



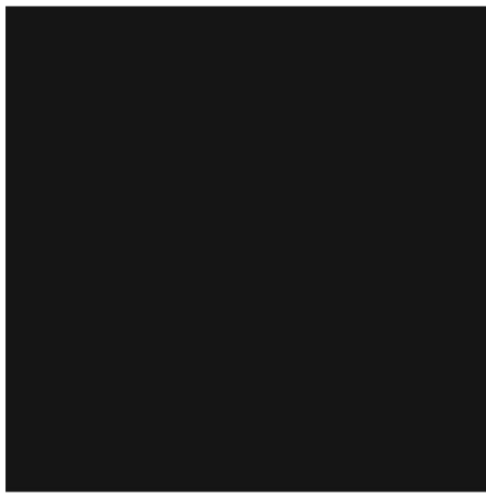
Se eligió esta librería de gráficos en particular para poder tomar más fácilmente el valor de los colores para el rango, la versión interactiva solo es visible desde Jupyter.

```
In [28]: light_blue = (90, 0, 20)
dark_blue = (160, 200, 255)
plot_colors(light_blue, "Light blue", dark_blue, "Dark blue")
```

Light blue



Dark blue



```
In [29]: mask_blue = cv2.inRange(bird_hsv, light_blue, dark_blue)
result_bird = cv2.bitwise_and(bird_rgb, bird_rgb, mask=mask_blue)
plot_mask_result(mask_blue, result_bird)
```

Mask



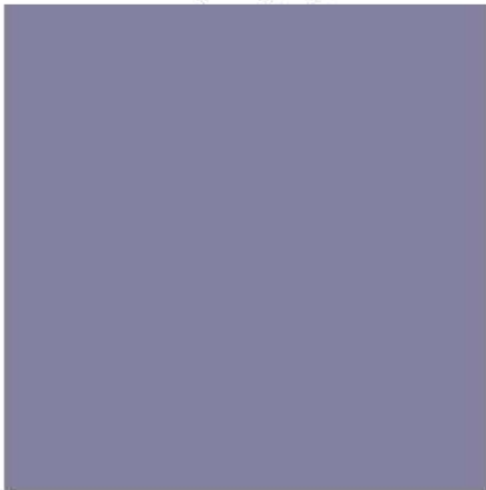
Image with mask



Nos quedan ciertos tonos grises del fondo, por lo que hacemos otra máscara:

```
In [30]: light_grey = (75, 0, 80)
dark_grey = (175, 50, 160)
plot_colors(light_grey, "Light grey", dark_grey, "Dark grey")
```

Light grey



Dark grey



Creamos una máscara inversa con dichos tonos de gris, y la aplicamos sobre el resultado anterior:

```
In [31]: mask_grey = (255 - cv2.inRange(bird_hsv, light_grey, dark_grey))
result = cv2.bitwise_and(result_bird, result_bird, mask=mask_grey)
plot_mask_result(mask_grey, result)
```

Mask



Image with mask



Ej 1.c: Segmentación de una rosa

```
In [32]: girl_bgr = cv2.imread('./assets/Girl_and_rose.jpg')
girl_rgb = cv2.cvtColor(girl_bgr, cv2.COLOR_BGR2RGB)
plot_image(girl_rgb, 'Girl')
```

Girl

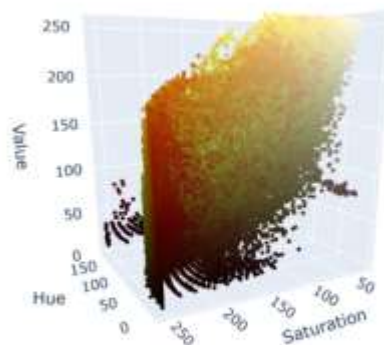


Analizamos los pixeles en el espacio HSV:

```
In [33]: girl_hsv = cv2.cvtColor(girl_bgr, cv2.COLOR_BGR2HSV)
hsv_3d_scatter(girl_hsv, 'Girl HSV pixels scatter plot')
```

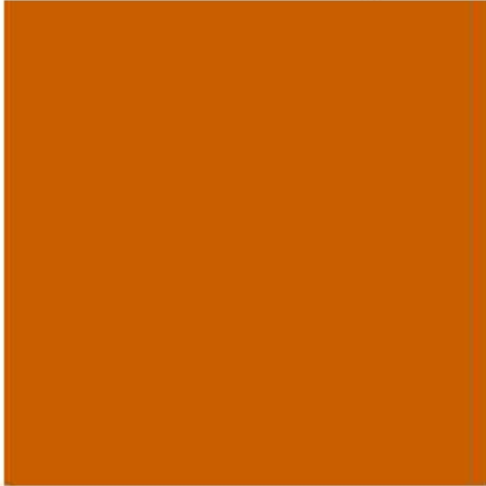
```
In [54]: my_image = skimage.io.imread(fname=f"{OUTPUT_FOLDER_PATH}/scatter_hsv_girl.jpg")
plot_image(my_image, '')
```

Girl HSV pixels scatter plot

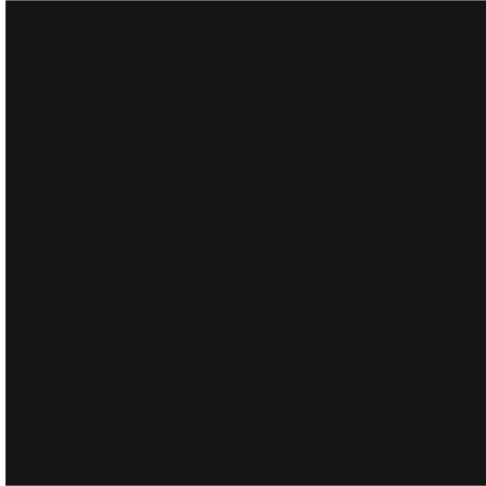


```
In [34]: light_orange = (1, 0, 20)
dark_orange = (20, 255, 200)
plot_colors(light_orange, "Lower boundary", dark_orange, "Higher boundary")
mask_rose = cv2.inRange(girl_hsv, light_orange, dark_orange)
result_girl = cv2.bitwise_and(girl_rgb, girl_rgb, mask=mask_rose)
plot_mask_result(mask_rose, result_girl)
```

Lower boundary



Higher boundary



Mask

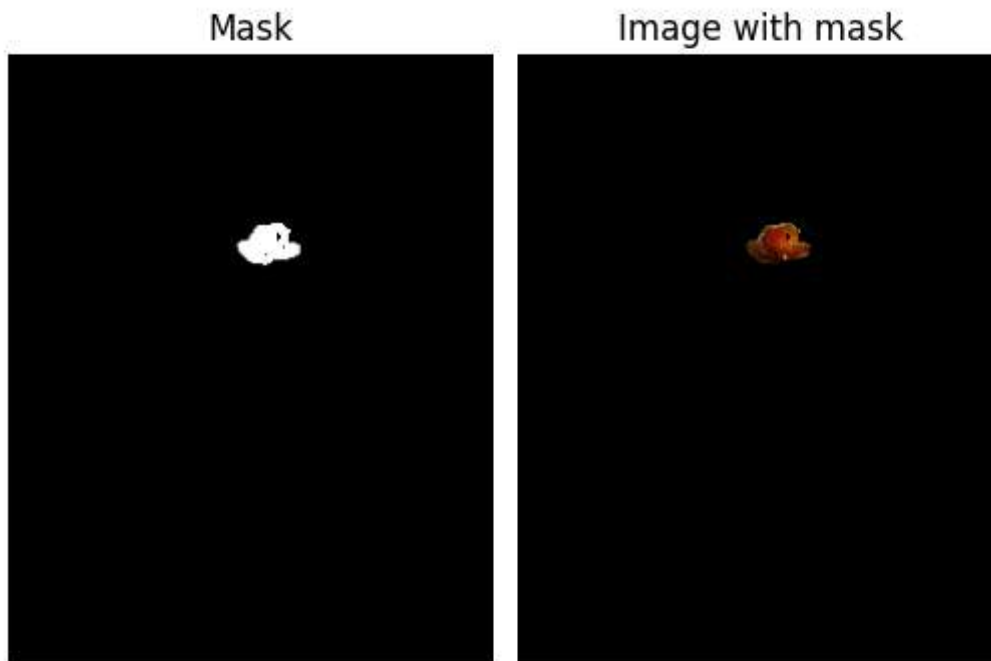


Image with mask



Como queremos solo la rosa:

```
In [35]: mask_rose[:80, :] = 0
mask_rose[116:315, :] = 0
mask_rose[80:116, :115] = 0
mask_rose[80:116, 151:] = 0
result_rose = cv2.bitwise_and(girl_rgb, girl_rgb, mask=mask_rose)
plot_mask_result(mask_rose, result_rose)
```



Quitamos la rosa de la imagen original:

```
In [36]: mask_without_rose = cv2.bitwise_not(mask_rose)
girl_without_rose = cv2.bitwise_and(girl_rgb, girl_rgb, mask=mask_without_rose)
```

Desaturamos la imagen para volverla blanco y negro:

```
In [37]: girl_desaturated = cv2.cvtColor(girl_without_rose, cv2.COLOR_RGB2HSV)
girl_desaturated[:, :, 1] = 0
girl_desaturated = cv2.cvtColor(girl_desaturated, cv2.COLOR_HSV2RGB)
plot_image(girl_desaturated, 'Girl desaturated')
```



Combinando ambas máscaras:


```
In [38]: result_girl_rose = girl_desaturated + result_rose  
plot_image(result_girl_rose, 'Girl bw + rose')
```

Girl bw + rose

