Análisis de imágenes en Python

Miguel Ángel Henao Pérez
Estudiante de ingeniería de sistemas y computación

Pereira, Risaralda, Colombia 2 de Junio de 2017

Facultad de ingenierías. Universidad tecnológica de Pereira.

RESUMEN

El siguiente artículo tiene como objetivo mostrar un poco la realidad acerca del análisis de imágenes en Python, además de mostrar un poco como sería el código y que métodos podemos utilizar para dicho análisis, luego de repasar un poco los conceptos básicos se estudian los métodos más utilizados para el análisis específico acerca de la segmentación de objetos.

Ni las imágenes ni los códigos utilizados son de mi autoría y sus respectivos autores se encuentran en la sección de referencias.

Además este artículo no pretende enseñar cual es la raíz teórica de los métodos matemáticos utilizados, solamente repasar la aplicación y el *How to* de estos procesos de análisis.

Recomendaciones o discusiones sobre el tema son recibidas en mi correo Miguelangelutp@gmail.com

Introducción

El análisis de imágenes consiste principalmente en la extracción de información a partir de sensores que en formato de dos o tres dimensiones representan algo gráficamente. El análisis puede ser visual y sin embargo el que nos compete actualmente es el análisis digital de dichas imágenes.

Un poco de historia

El análisis de imágenes no es algo nuevo en absoluto, de la observación de los exploradores y guardias dependía la seguridad de toda la comunidad, no hasta hace muchos años la seguridad se radicaba en la memoria y calidad de observación del informando. Poco a poco las técnicas fueron mejorando hasta llegar a procedimientos un poco más complicados tales como el análisis de las fotografiás, en la cual la información quedaba guardada en un medio físico y mejoraba la calidad de la información.

Las guerras también fueron fundamentales para el desarrollo de estas técnicas, la utilización permitía formar estrategias y anticipar al enemigo.

Para el año de 1930 se utilizaron las primeras imágenes infrarrojas, y su aplicación inicial fue encontrar los fraudes en las obras de arte.

El análisis realizado en los radares también fue muy importante en la segunda guerra mundial y la guerra fría y a partir de ahí los avances han sido muchos, por mencionar algunos tenemos los sensores infrarrojos aéreos, las imágenes por ultrasonido, el radar de apertura sintética entre otras.

Algunas técnicas muy comunes para entender el análisis de imágenes

Dado que la linea que estamos analizando es la segmentación de objetos, describiremos brevemente unas formas de facilitar este trabajo de segmentación.
Las más frecuentes son:

1. **Erosión**
2. **Thresholding**
3. **Detección de esquinas**
4. **Labeling**

Erosión

También se le atribuyen nombres como reducción o achicamiento. Es uno de los dos operadores básicos de la morfología matemática, siendo el otro la dilatación. Usualmente es aplicado a imágenes binarias y su efecto es el de eliminar los bordes de las regiones de pixels, usualmente se realiza primero una técnica de thresholding y estas áreas quedan llenas de pixels blancos. Cuando los bordes se eliminan las regiones dejan de estar en contacto unas con otras y permiten un análisis posterior.

¿Como funciona?

Su descripción es más un tema de morfología matemática pero su definición es básicamente la siguiente: El operador erosión toma dos piezas de datos, la primera es una imagen y la segunda es una matriz conocida por el nombre de elemento estructurador o kernel. La definición del concepto matemático es: El conjunto de puntos tal que la matriz sea un subconjunto de dicho conjunto.

Algunos ejemplos de erosión

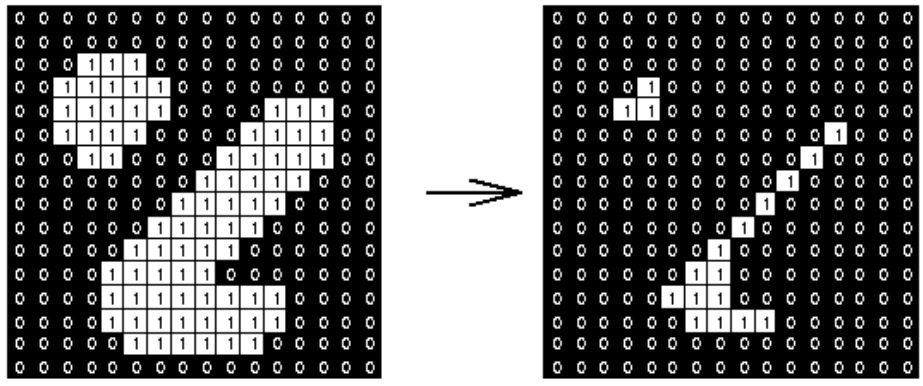
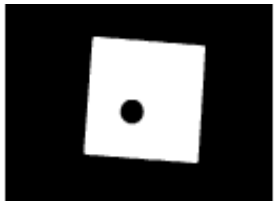


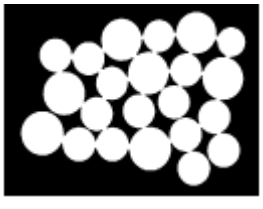
Figure 2 Effect of erosion using a 3×3 square structuring element

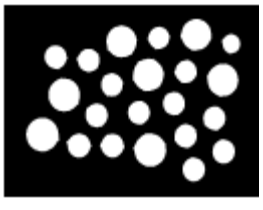
La imagen original

La imagen después de la erosión



Otra aplicación es separar objetos que están en contacto para un análisis de segmentación





En *Python* también tenemos una herramienta para realizar erosión a una imagen en formato binario. El código es el siguiente

scipy.ndimage.morphology.binary_erosion(*input, structure=None, iterations=1, mask=None, output=None, border_value=0, origin=0, brute_force=False***)**

EJEMPLO

```
>>> a = np.zeros((7,7), dtype=np.int)

>>> a[1:6, 2:5] = 1

>>> a

array([[0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0]])

>>> ndimage.binary_erosion(a).astype(a.dtype)

array([[0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0]])
```

Thresholding

La práctica de thresholding es quizás la más sencilla para la segmentación de imágenes, pero precisamente por su sencillez es importante entenderla y dominarla porque es la base de cualquier procesamiento de imágenes.

Este proceso tiene como objetivo la creación de imágenes binarias y para esto no utiliza ni colores ni escalas de grises, sino una imagen con blancos y negros en su totalidad.

El requerimiento para remplazar estos pixels es muy lógico. Si dicho pixel tiene una intensidad inferior a un valor determinado T llamado valor de thresholding entonces se remplaza con un 0 (NEGRO) Pero si por el contrario el pixel supera este valor entonces se remplaza por un 1 (BLANCO) de esta manera se consigue una matriz binaria que facilita el análisis y sobretodo la segmentación de objetos.

Existen varios métodos para seleccionar este valor T dado que para que el proceso sea un autómata es necesario primero seleccionar el mejor valor de T que permita representar la imagen incluso en formato binario.

Los métodos más conocidos son:

1. **Histograma**: Donde se analiza a partir de muestras de los grises de la imagen el valor de T.
2. **Clustering**: Separa las muestras de grises en valores de fondo y objetos, aunque también tiene aplicaciones utilizando dos campanas de Gauss.
3. **Entropía:** Son métodos que utilizan los niveles de entropía entre las imágenes ubicadas al fondo y las al frente. También se utilizan métodos como calcular la entropía entre la imagen original y la imagen ya sometida para verificar en cual de las n muestras hubo un mejor desempeño del valor T.
4. **Métodos espaciales:** Estos métodos ordenan los pixels de acuerdo a distribuciones probabilistas y a la correlación entre los pixels, lo que muestra mejores resultados que la selección yana de un valor T.

Por otra parte existen métodos de Thresholding para múltiples valores. Así es como encontramos el Thresholding de RGB, separando los colores por su proximidad al rojo verde y azul. Aunque estás técnicas no son las que utiliza el ojo humano, si se utilizan en computación y se pueden lograr resultados deseables.

Para ilustrar el thresholding se utilizarán varías técnicas para mostrar la diferencia de resultados que pueden ofrecer.

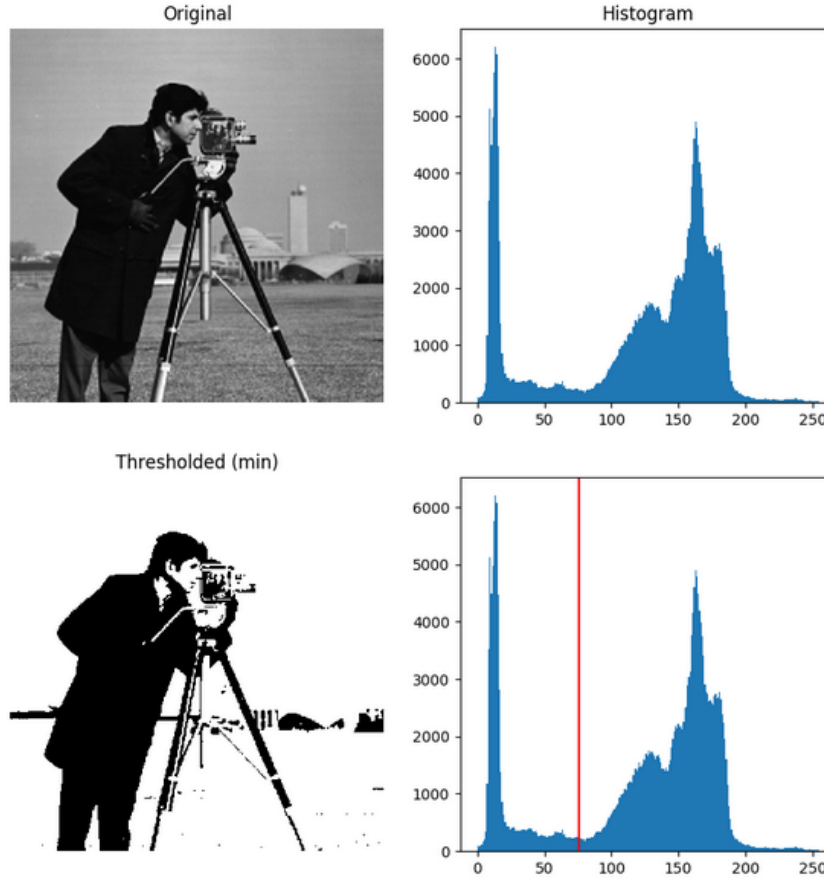
1. **Thresholding utilizando la media de las intensidades (Rendimientos muy bajos)**

```
from skimage.filters import threshold_mean


image = data.camera()
thresh = threshold_mean(image)
binary = image > thresh

fig, axes = plt.subplots(ncols=2, figsize=(8, 3))
ax = axes.ravel()

ax[0].imshow(image, cmap=plt.cm.gray)
ax[0].set_title('Original image')

ax[1].imshow(binary, cmap=plt.cm.gray)
ax[1].set_title('Result')

for a in ax:
    a.axis('off')

plt.show()
```

2.
Histograma bimodal En este tipo de thresholding se opera repetidamente hasta que queden dos picos (0 y 1) y se encuentra el valor entre ellos para separarlos.

```
from skimage.filters import threshold_minimum


image = data.camera()

thresh_min = threshold_minimum(image)
binary_min = image > thresh_min
```

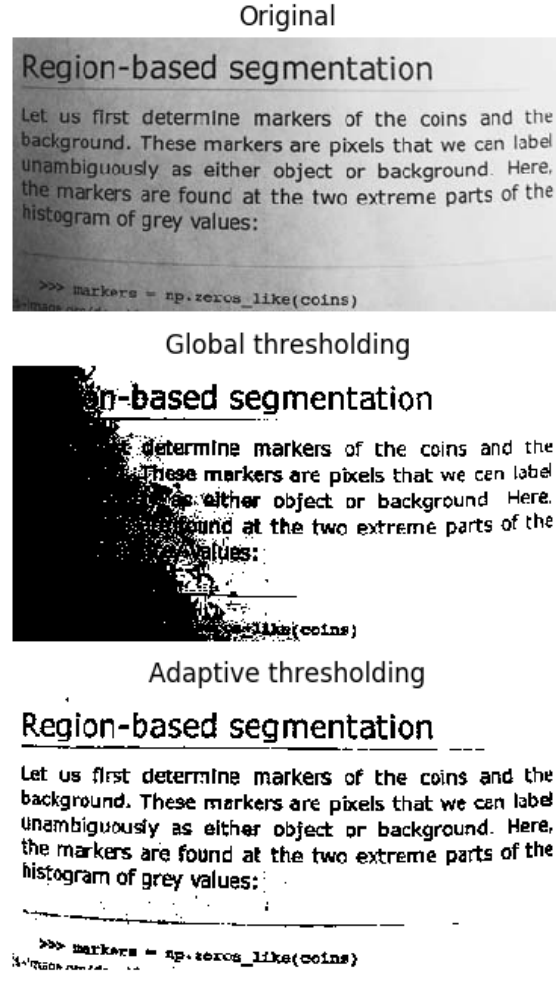
```
fig, ax = plt.subplots(2, 2, figsize=(10, 10))

ax[0, 0].imshow(image, cmap=plt.cm.gray)
ax[0, 0].set_title('Original')

ax[0, 1].hist(image.ravel(), bins=256)
ax[0, 1].set_title('Histogram')

ax[1, 0].imshow(binary_min, cmap=plt.cm.gray)
ax[1, 0].set_title('Thresholded (min)')

ax[1, 1].hist(image.ravel(), bins=256)
ax[1, 1].axvline(thresh_min, color='r')

for a in
ax[:,0]:
a.axis('off')
plt.show()
```



3. Thresholding Local Si existe una gran variación en las intensidades de la imagen se puede usar un Thresholding adaptativo o dinámico que aunque es mucho más lento ofrece mejores resultados.

```
from skimage.filters import threshold_otsu, threshold_local


image = data.page()

global_thresh = threshold_otsu(image)
binary_global = image > global_thresh
```

```
block_size = 35
adaptive_thresh = threshold_local(image, block_size, offset=10)
binary_adaptive = image > adaptive_thresh

fig, axes = plt.subplots(nrows=3, figsize=(7, 8))
ax = axes.ravel()
plt.gray()

ax[0].imshow(image)
ax[0].set_title('Original')

ax[1].imshow(binary_global)
ax[1].set_title('Global thresholding')

ax[2].imshow(binary_adaptive)
ax[2].set_title('Adaptive
thresholding')

for a in ax:
    a.axis('off')

plt.show()
```



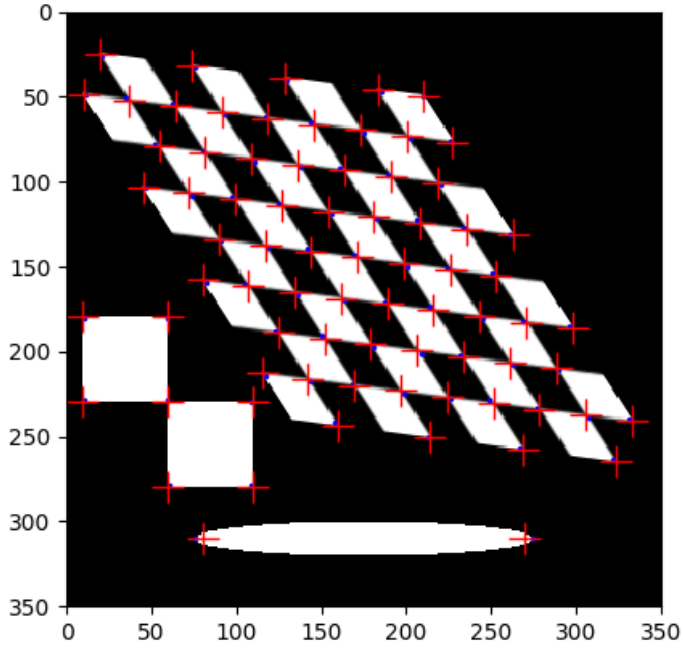
Detección de esquinas

La detección de esquinas es un intento por recrear la visión computarizada, y extrae algunas características de las imágenes, es utilizada muy frecuentemente en la detección de objetos en movimiento, los registros de imágenes, el modelado 3D entre otras. Y se constituye como parte del tema detección de objetos de interés.

Existen muchos modelos matemáticos que explican el funcionamiento de la detección de esquinas, e incluso hay resoluciones de programación genética para encontrar estos puntos de interés.

Para mostrar el método no se utilizarán los modelos matemáticos convencionales, precisamente por su dificultad se van a omitir y se va a utilizar un método más sencillo llamado Esquinas de Harris importado de la librería Scikimage.features.

```
from matplotlib import pyplot as plt

from skimage import data
from skimage.feature import corner_harris, corner_subpix, corner_peaks
from skimage.transform import warp, AffineTransform
from skimage.draw import ellipse

tform = AffineTransform(scale=(1.3, 1.1), rotation=1, shear=0.7,
                        translation=(210, 50))
image = warp(data.checkerboard(), tform.inverse, output_shape=(350, 350))
rr, cc = ellipse(310, 175, 10, 100)
image[rr, cc] = 1
image[180:230, 10:60] = 1
image[230:280, 60:110] = 1

coords = corner_peaks(corner_harris(image), min_distance=5)
coords_subpix = corner_subpix(image, coords, window_size=13)

fig, ax = plt.subplots()
ax.imshow(image, interpolation='nearest', cmap=plt.cm.gray)
ax.plot(coords[:, 1], coords[:, 0], '.b', markersize=3)
ax.plot(coords_subpix[:, 1], coords_subpix[:, 0], '+r', markersize=15)
ax.axis((0, 350,
350, 0))
plt.show()
```



Segmentación de imágenes

Finalizando nuestro tema de interés tenemos un ejemplo que concierte única y exclusivamente a la segmentación de las imágenes. Es un ejemplo donde la región con la imagen se etiqueta y se subraya, diferencíandola así del resto de la imagen. Clasificando monedas de detonación similar.
El proceso para conseguir esta segmentación involucró todos los temas explicados anteriormente exceptuando la detección de esquinas y demuestra la capacidad y el alcance de estas herramientas.

Los pasos fueron los siguientes

1. Se aplicó el método de thresholding y se utilizó un procedimieto especial llamado Otsu method
2. Se encierran los espacios pequeños de la imagen con una herramienta llamada Binary| closing que básicamente es una aplicación de la erosión.
3. Se remueven los objetos que bordean la imagen y podrían ser reconocidos como bojetos cuando quizás sean parte de alguno pero no objetos en si.
4. Se miden las regiones de las imágenes y se filtran los objetos pequeños.

Los pasos se expresan en el siguiente código.

```
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

from skimage import data
from skimage.filters import threshold_otsu
from skimage.segmentation import clear_border
from skimage.measure import label, regionprops
from skimage.morphology import closing, square
from skimage.color import label2rgb

image = data.coins()[50:-50, 50:-50]

# apply threshold
thresh = threshold_otsu(image)
bw = closing(image > thresh, square(3))

# remove artifacts connected to image border
cleared = clear_border(bw)

# label image regions
label_image = label(cleared)
image_label_overlay = label2rgb(label_image, image=image)

fig, ax = plt.subplots(figsize=(10, 6))
ax.imshow(image_label_overlay)
for region in regionprops(label_image):
    # take regions with large enough areas
    if region.area >= 100:
        # draw rectangle around segmented coins
        minr, minc, maxr, maxc = region.bbox
        rect = mpatches.Rectangle((minc, minr), maxc - minc, maxr - minr,
                                  fill=False, edgecolor='red', linewidth=2)
        ax.add_patch(rect)

ax.set_axis_off()
plt.tight_layout()
plt.show()
```

Imagen original

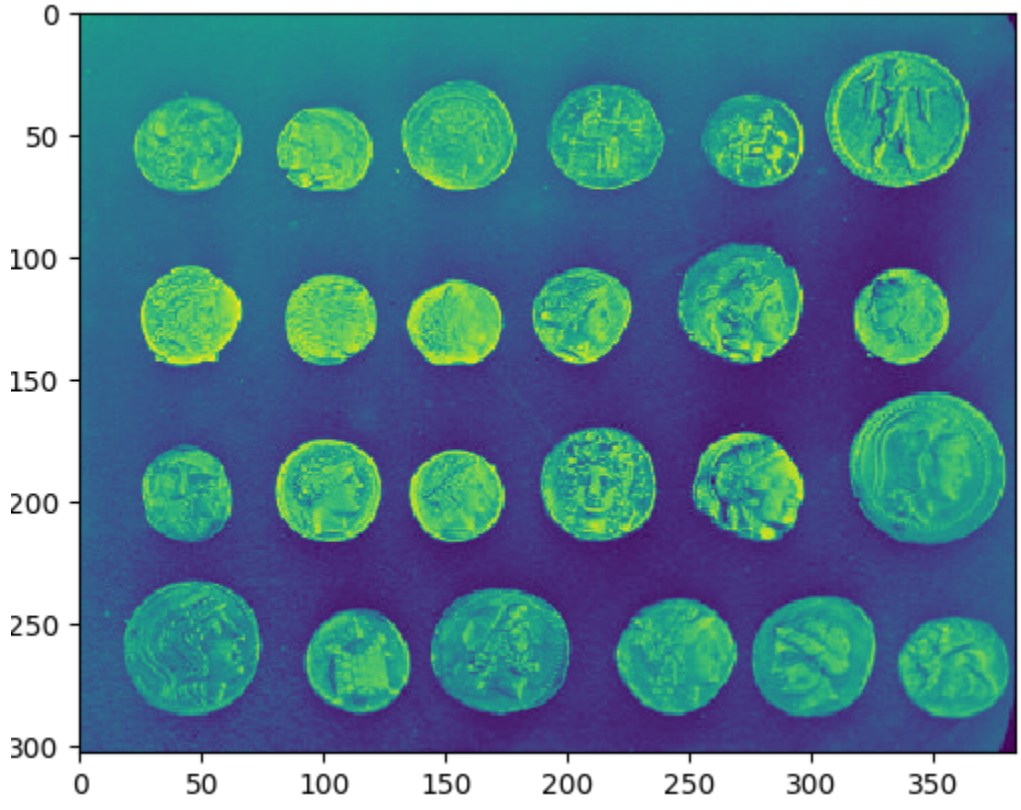
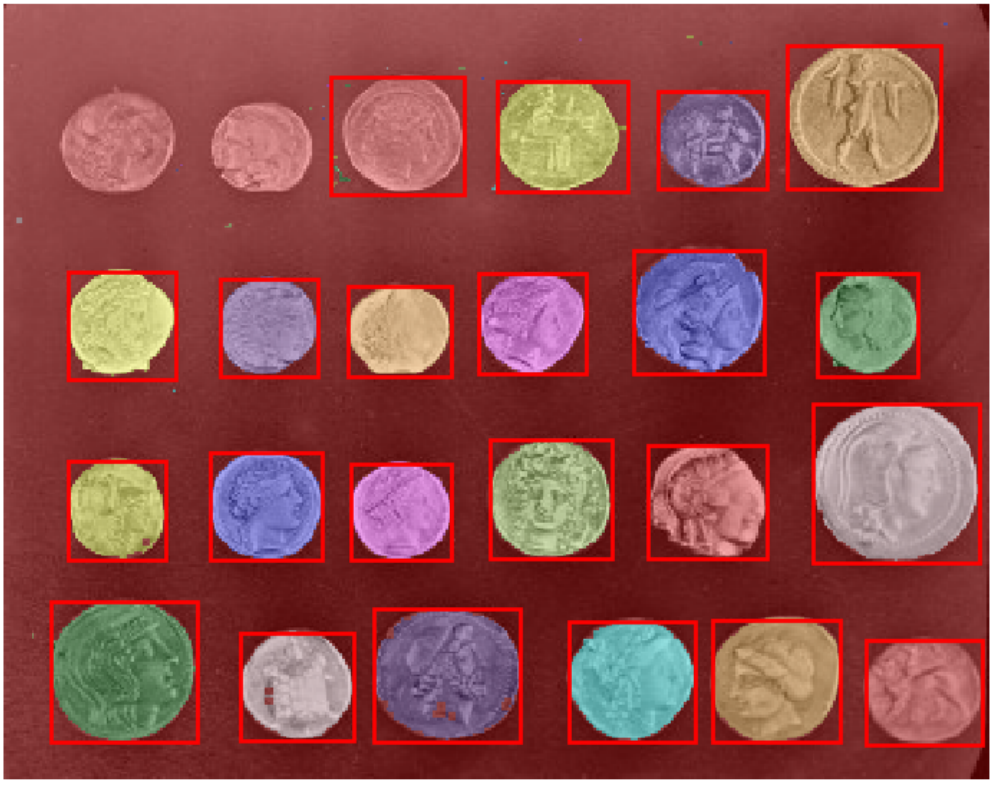


Imagen después del tratamiento



Conclusiones

Existen herramientas y cada vez llegan más rápido, capaces de lograr avances tan grandes como los conseguidos alguna vez en décadas, pero ahora se están generando incluso en semanas.

Los avances nacen de la creatividad de las personas, incluso en temas tan técnicos se demuestra que el ingenio aparece en las ideas innovadoras y en la mezcla de conocimiento.

La ficción no parece tan alejada de la realidad cuando se empiezan a utilizar estas técnicas, métodos con aplicaciones similares se han visto en películas como Terminator y Matrix. El pensar humano extralimita las fronteras a partir de la razón.

Existe la posibilidad de trascender en este tema, pues aunque los sistemas de detección pueden hacer cosas increíbles, existe mucha información no divulgada públicamente. Las convenciones y los tratados militares han ocultado gran parte del avance tecnológico, limitando en tiempo a los investigadores.

Los campos de aplicación son enormes para el análisis y procesamiento de imágenes, tanto que requiere publicaciones e investigaciones constantes. Requiere una profundización y encontrar el provecho en las diferentes temáticas.

Es importante recordar el papel que jugamos en la sociedad, y en base a esto pensar en las posibilidades para mejorarla utilizando todas nuestras herramientas.

Referencias

1.- Gonzalez, Rafael C. & Woods, Richard E. (2002). Thresholding. In Digital Image Processing, pp. 595–611. Pearson Education. ISBN 81-7808-629-8

2.- *Zhang, Y. (2011). "Optimal multi-level Thresholding based on Maximum Tsallis Entropy via an Artificial Bee Colony Approach". Entropy.* ***13*** *(4): 841–859.*

3.- *C. Harris and M. Stephens (1988). "A combined corner and edge detector" (PDF). Proceedings of the 4th Alvey Vision Conference. pp. 147–151.*

4.- *Andrew Willis and Yunfeng Sui (2009). "An Algebraic Model for fast Corner Detection". 2009 IEEE 12th International Conference on Computer Vision. IEEE. pp. 2296–2302. doi:10.1109/ICCV.2009.5459443. ISBN 978-1-4244-4420-5.*

5.-*Bryan S. Morse, Brigham Young University, 1998–2000 SH&B, Section 5.1*

6.-http://scikit-image.org/docs/dev/auto_examples/xx_applications/plot_rank_filters.html#sphx-glr-auto-examples-xx-applications-plot-rank-filters-py

7.-http://scikit-image.org/docs/dev/auto_examples/segmentation/plot_thresholding.html#sphx-glr-auto-examples-segmentation-plot-thresholding-py

8.- http://programacion.net/articulo/procesamiento_de_imagenes_utilizando_python_1451

9.-http://scikit-image.org/docs/dev/auto_examples/segmentation/plot_label.html#sphx-glr-auto-examples-segmentation-plot-label-py