# Reporte PP

December 2, 2021

# 1 ALGORITMO BASADO EN REDES NEURONALES, PARA MEDIR OBJETOS TRIDIMENSIONALES EN COORDENADAS RECTANGULARES

## 1.1 *Reporte del desarrollo de análisis de imágenes y partículas con OpenCV y Python*

Facultad de Ciencias de la Electrónica Benemérita Universidad Autónoma de Puebla Coordinador: Dr. Gustavo Mendoza Torres

Alumnos: - *Francisco Hernández Rodríguez 201619851 - Heidi Rebeca Rojas Montalvo 201508965 - Juan Miguel Ovando Castillo 201627820 - José Ángel Garzón Gonzáles 201635406*

Verano 2021

## 1.2 Objetivo

Mediante el lenguaje de programación Python 3 desarrollar un algoritmo para obtener mediciones sobre partículas de ceniza volcánica.

### 1.2.1 Actividades

- Buscar el software adecuado para llevar a cabo las mediciones
- Seleccionamos el software que cumple con nuestros requisitos
- Analizar y estudiar el software seleccionado
- Análisis de ventajas y limitantes del software
- Vincular el software seleccionado con Python
- Programación para identificación de las imágenes
- Análisis de las características geométricas del objeto y mediciones
- Resultados finales

### 1.2.2 Importancia del análisis de partículas

El análisis del tamaño de las partículas se utiliza para caracterizar la distribución del tamaño de las partículas en una muestra determinada. El análisis del tamaño de partículas se puede aplicar a materiales sólidos, suspensiones, emulsiones e incluso aerosoles. Hay muchos métodos diferentes empleados para medir el tamaño de las partículas. Algunos métodos de tamaño de partículas se pueden usar para una amplia gama de muestras, pero algunos solo se pueden usar para aplicaciones específicas. Es muy importante seleccionar el método más adecuado para diferentes muestras, ya que diferentes métodos pueden producir resultados bastante diferentes para el mismo material.

### 1.2.3 Sobre las partículas de ceniza volcánicas

Las partículas de ceniza volcánica tienen una amplia distribución de formas y tamaños. Su análisis morfométrico emplea diferentes técnicas, como la observación por microscopio electrónico de barrido (SEM) o el análisis de difracción láser. Las partículas de ceniza, derivadas del vidrio, roca y cristal volcánico, presentan una estructura de tipo vesicular con numerosas cavidades, en la que generalmente dominan las formas irregulares con bordes afilados y dentados, lo cual torna a la ceniza en un material abrasivo. El tamaño, por consenso de un grupo de expertos para referirse a ceniza volcánica, es de 2 milímetros de diámetro o menos. El límite superior en el tamaño de las partículas se determina mediante procesos de sedimentación, el cual depende del diámetro y la forma de la partícula. [1]

### 1.2.4 Trabajo previo

El primer acercamiento al analisis de partículas de ceniza volcánica se realizó con ImageJ el cual es un software abierto para el procesamiento de imágenes. El cual contiene una gran variedad de Funciones y operaciones que podemos aplicar sobre imágenes, ya sea utilizando su IDE FIJI o utilizando una de sus herramientas más poderosas, la posibilidad de realizar scripts en diferentes lenguajes como Python para ejecutar macros o comandos sobre nuestras imágenes. De esta forma un primer script utilizando Python 2 se ideo para obtener las mediciones de las particulas de ceniza volcánica de algunas imágenes tomadas con microscopio:

```
from ij import IJ
from ij.io import FileSaver
from ij.io import OpenDialog
import os
from ij.process import ImageStatistics as IS
import ij.gui.PolygonRoi
import ij.gui.Roi
from ij.plugin.frame import RoiManager
from ij import IJ, ImagePlus
from ij.plugin import CanvasResizer
from ij.process import ImageProcessor
from ij.plugin.filter import ThresholdToSelection
from time import sleep

od = OpenDialog("Chose a file:", None)

filename = od.getFileName()
directory = od.getDirectory()
path = od.getPath()

imp = IJ.openImage(path);
imp.getProcessor().setThreshold(60, 255, ImageProcessor.NO_LUT_UPDATE)
roi = ThresholdToSelection.run(imp)
imp.setRoi(roi)
maskimp = ImagePlus("Mask", imp.getMask())
maskimp.show()
```

```
IJ.run(maskimp, "Analyze Particles...", "size=2000.00-Infinity display include add");

IJ.run("From ROI Manager","");

IJ.renameResults("Results","Resultados_Particulas");

IJ.saveAs("Results", "C:/Users/JMCas/Downloads/Resultados_Particulas.csv");

IJ.deleteRows(0,103);

rm = RoiManager().getInstance()
total_rois = rm.getCount()

lectura_1=total_rois;

for roi in range(total_rois):
    rm.select(roi)
    IJ.run('Convex Hull')
    IJ.run('Measure')
    sleep(1)
print(rm)

IJ.renameResults("Results", "Cascaras_Convexas");

IJ.saveAs("Results","C:/Users/JMCas/Downloads/Cascaras_Convexas.csv")

IJ.deleteRows(0,103);

print("Fin del Programa")
```

Este código tiene los siguientes pasos en su algoritmo: - Búsqueda y apertura de los archivos de imágenes - Ajustar Threshold - Aplicar máscara de Threshold - Ejecutar análisis de partículas ( excluyendo las de menor tamaño) - Abrir ROI (Region of Interest) Manager para obtener medidas individuales - Guardar medidas - Aplicar cáscara convexa para cada partícula - Guardar medidas - Fin del programa

Las limitaciones iniciales de este algoritmo están asociadas al software ImageJ el cuál cuenta con funciones limitadas al momento de ejecutar scripts puesto que su lenguaje nativo es Java, de modo que solo ciertas herramientas están disponibles para Python el cual solo se puede utilizar en su versión Python 2, afortunadamente este algoritmo cienta la base sobre la cual se desarrollará el código en Python 3 con el cuál se tratará de explotar el potencial del lenguaje.

## 2 Primer acercamiento: PyImageJ

PyimageJ gestiona una version del software en Python 3, de modo que de acuerdo a la configuración que le indiquemos podemos acceder a diferentes versiones de ImageJ (1-2) o bien acceder al launcher y sus plugins a los cuales accedemos con Fiji. En el repositorio de pyimagej podemos encontrar los metodos de inicialización antes mencionados.

En la pagina de lenguajes compatibles de ImageJ se menciona que el modulo de python tiene algunos errores con ImageJ1.x y es más estable con ImageJ2. Inicialización de ImageJ gracias a PyImageJ que nos permite ejecutarlo desde Python 3:

[1]:
```
import imagej
ij = imagej.init('net.imagej:imageJ:2.1.0', headless=False)
ij.getVersion()
```

[1]:
```
'2.1.0/1.53c'
```

Informacion de la version de ImageJ, motor Java (recordemos que ImageJ esta basado en SciJava) y cantidad de memoria asignada a la ejecucion:

[2]:
```
ij.getApp().getInfo(True)
```

[2]:
```
'ImageJ 2.1.0/1.53c; Java 11.0.9.1 [amd64]; 40MB of 4082MB'
```

Pedimos información del paquete:

[ ]:
```
help(ij.py)
```

Al haber llamado una version de Fiji, contamos con todos los plugins para imageJ:

[4]:
```
pluginCount = ij.plugin().getIndex().size()
pCount = ij.py.from_java(pluginCount)
print("There are " + str(pCount) + " plugins available.")
```

```
There are 1671 plugins available.
```

A si mismo, contamos con todos los menus:

[5]:
```
menuItemCount = ij.menu().getMenu().size()
mICount = ij.py.from_java(menuItemCount)
print("There are " + str(mICount) + " menu items total.")
```

```
There are 466 menu items total.
```

Ya que en esencia, PyimageJ hace un llamado a una version local de la API, tenemos acceso a la mayoría de metodos y atributos:

[ ]:
```
print(ij.op().help())
```

No existe documentacion especifica sobre el uso de estas operaciones y los argumentos que toman en PyimageJ (Python) más allá de la sintaxis y algunos ejemplos que encontramos referentes a la API usando Grooy (Java). Tomemos por ejemplo la operacion Threshold con el método Otsu, la información de ayuda es poco amigable y no brinda información suficiente para su uso:

[7]:
```
ij.op().help("threshold.otsu")
```

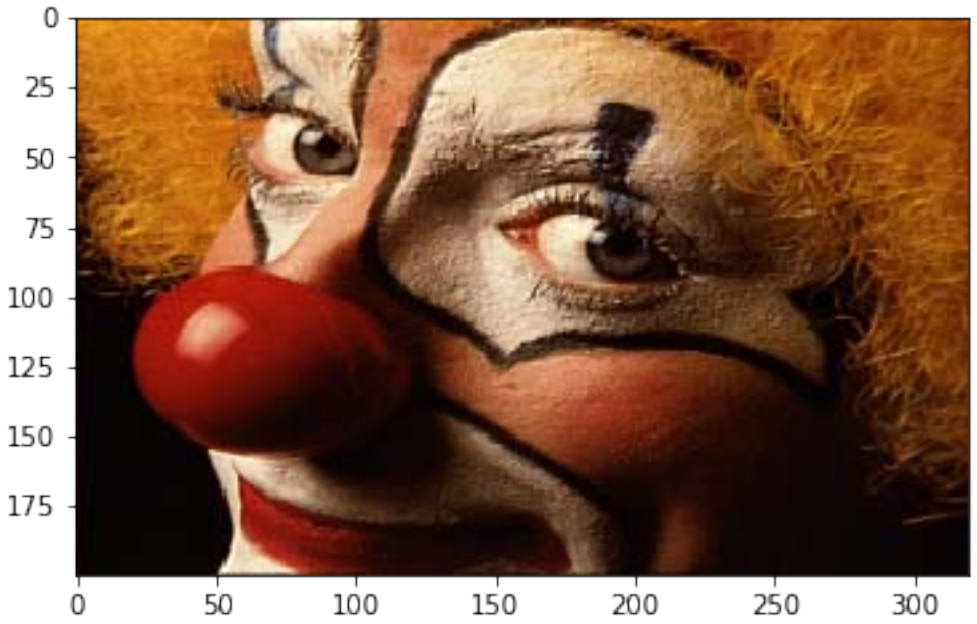
[7]:
```
'Available operations:
        (RealType out?) =
        net.imagej.ops.threshold.otsu.ComputeOtsuThreshold(
```

```
        RealType out?,
        Histogram1d in)
(IterableInterval out?) =
net.imagej.ops.threshold.ApplyThresholdMethod$Otsu(
        IterableInterval out?,
        IterableInterval in)
(IterableInterval out) =
net.imagej.ops.threshold.ApplyThresholdMethodLocal$LocalOtsuThreshold(
        IterableInterval out,
        RandomAccessibleInterval in,
        Shape shape,
        OutOfBoundsFactory outOfBoundsFactory?)'
```

ImageJ brinda un pequeño tutorial sobre como realizar algunas instrucciones básicas, a continuación realizamos algunos ejemplos en nuestra instancia de imageJ:

[8]:
```
from skimage import io   #llamado a la biblioteca skimage que nos permite abrir␣
↪imagenes desde internet
import numpy as np       #numpy para algebra lineal
clown = io.imread("https://imagej.net/images/clown.png")  #abrimos con la url
```

[9]:
```
ij.py.show(clown)
```




Se puede observar que con el método to_java podemos transformar de un tipo de dato numpy (matriz) a uno de java:

[10]:
```
type(clown)
```

[10]:
```
numpy.ndarray
```

[11]:
```
type(ij.py.to_java(clown))
```

[11]:
```
<java class 'net.imglib2.python.ReferenceGuardingRandomAccessibleInterval'>
```

[12]:
```
type(ij.op().transform().flatIterableView(ij.py.to_java(clown)))
```

[12]:
```
<java class 'net.imglib2.view.IterableRandomAccessibleInterval'>
```

Algo interesante es que el tipo de dato de java tambien puede ser de dos tipos distintos: * ReferenceGuardingRandomAccessibleInterval * IterableRandomAccessibleInterval

Observando la ayuda del método Otsu vemos que existen 3 formas de llamar la operación, algunas aceptan datos de tipo Iterable y otras de tipo Random, sin embargo no es posible saber con exactitud cual de ellas estamos llamando desde PyImageJ, se hace la suposición de que se toma el indicado para el tipo de argumentos que le indiquemos. Intentemos hacer el filtro a nuestra imagen de payaso:

[13]:
```
# image = ij.op().threshold().otsu(clown)
image = ij.op().run("threshold.otsu", ij.py.to_java(clown))
```

[14]:
```
ij.py.show(image, cmap='gray')
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
<ipython-input-14-130065067259> in <module>
----> 1 ij.py.show(image, cmap='gray')

D:\Programas\Anaconda\envs\ImageJ\lib\site-packages\imagej\__init__.py in␣
 ↪show(self, image, cmap)
    615             from matplotlib import pyplot
    616
--> 617             pyplot.imshow(self.from_java(image),␣
 ↪interpolation='nearest', cmap=cmap)
    618             pyplot.show()
    619

D:\Programas\Anaconda\envs\ImageJ\lib\site-packages\imagej\__init__.py in␣
 ↪from_java(self, data)
    550             except Exception as exc:
    551                 _dump_exception(exc)
--> 552                 raise exc
    553             return sj.to_python(data)
    554

D:\Programas\Anaconda\envs\ImageJ\lib\site-packages\imagej\__init__.py in␣
 ↪from_java(self, data)
```

```
    547                 if self._ij.convert().supports(data,␣
 ↪RandomAccessibleInterval):
    548                     rai = self._ij.convert().convert(data,␣
 ↪RandomAccessibleInterval)
--> 549                     return self.rai_to_numpy(rai)
    550             except Exception as exc:
    551                 _dump_exception(exc)

D:\Programas\Anaconda\envs\ImageJ\lib\site-packages\imagej\__init__.py in␣
 ↪rai_to_numpy(self, rai)
    307             """
    308             result = self.new_numpy_image(rai)
--> 309             self._ij.op().run("copy.rai", self.to_java(result), rai)
    310             return result
    311

D:\Programas\Anaconda\envs\ImageJ\lib\site-packages\imagej\__init__.py in␣
 ↪to_java(self, data)
    372             """
    373             if self._is_memoryarraylike(data):
--> 374                 return imglyb.to_imglib(data)
    375             if self._is_xarraylike(data):
    376                 return self.to_dataset(data)

D:\Programas\Anaconda\envs\ImageJ\lib\site-packages\imglyb\util.py in␣
 ↪to_imglib(source)
    137 def to_imglib(source):
    138     scyjava.start_jvm()
--> 139     return ReferenceGuardingRandomAccessibleInterval(_to_imglib(source)␣
 ↪ReferenceGuard(source))
    140
    141

D:\Programas\Anaconda\envs\ImageJ\lib\site-packages\imglyb\util.py in␣
 ↪_to_imglib(source)
    147
    148     if not source.dtype in numpy_dtype_to_conversion_method:
--> 149         raise NotImplementedError("Cannot convert dtype to ImgLib2 type␣
 ↪yet: {}".format(source.dtype))
    150     elif source.flags['CARRAY']:
    151         return numpy_dtype_to_conversion_method[source.
 ↪dtype](long_address, *long_arr_source)

NotImplementedError: Cannot convert dtype to ImgLib2 type yet: bool
```

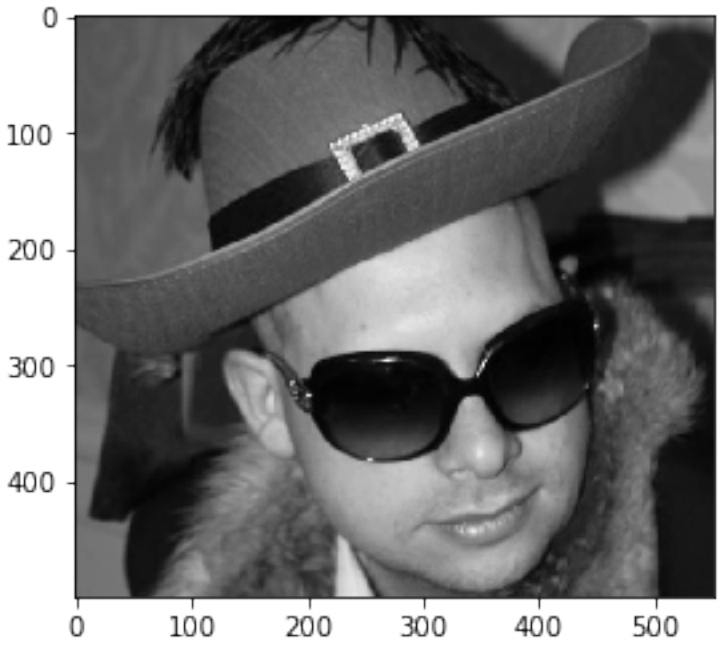
El error obtenido resulta del propio motor de procesamiento de imágenes de ImageJ: ImgLib2 por lo que muchos resultados son en esencia datos de un 3er tipo. Hasta este punto no fue posible

encontrar una función que adecuará el tipo de dato que resulto de hacer el threshold otsu a la foto de payaso. Intentando adecuar los tipos de datos bajo la suposición del mejor método:

[ ]:
```
result = ij.py.new_numpy_image(clown)

imgIterable = ij.op().transform().flatIterableView(ij.py.to_java(clown))
resIterable = ij.op().transform().flatIterableView(ij.py.to_java(result))

ij.op().threshold().otsu(resIterable)

ij.py.show(resIterable, cmap='gray')
```

La imagén resultante no se aprecia como debería. Para descartar un error en el funcionamiento de PyImageJ replicamos otro tutorial en el que se hace uso de un filtro basado en diferencia gaussiana:

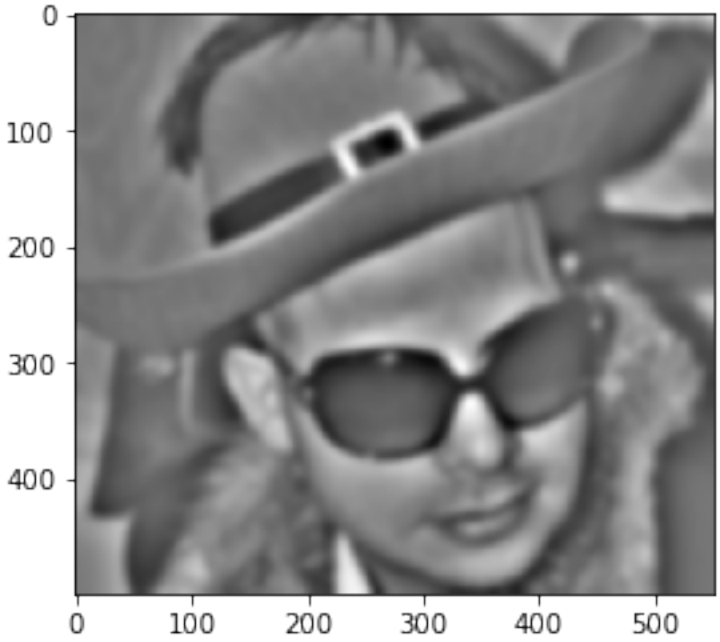
[15]:
```
from skimage import io
import numpy as np
img = io.imread('https://samples.fiji.sc/new-lenna.jpg')
img = np.mean(img[500:1000,300:850], axis=2)
ij.py.show(img, cmap='gray')
```




[16]:
```
np.shape(img) #tamaño de la matriz de pixeles
```

[16]:
```
(500, 550)
```

[17]:
```
result = np.zeros(img.shape)
sigma1 = 20
sigma2 = 4
ij.op().filter().dog(
    ij.py.to_java(result),
    ij.py.to_java(img),
    sigma1,
    sigma2)
ij.py.show(result, cmap='gray')
```




[18]:
```
type(img)
```

[18]:
```
numpy.ndarray
```

Observamos que después de aplicar el filtro como una operación desde ImageJ se obtuvo un buen resultado y esta vez se concervo el tipo de dato numpy.ndarray.

Las diferentes transformaciones entre datos y operaciones que funcionen en PyImageJ no están documentadas en ningun lado, ni siquiera en la wiki. Lo que generalmente se menciona es que al tener una versión local de fiji tenemos acceso a todas la operaciones, plugins, métodos y macros de ImageJ pero no se menciona la forma correcta o sintáxis adecuada para ello.

Hasta el momento las operaciones utilizadas aparecen listadas como métodos net.imageJ.ops que encontramos en la API.

A su vez existen otras API: * Una especifica para Fiji de la cual desconocemos la forma de acceso de PyImageJ * Una para modulos IJ utilizados en scripts programados en diferentes lenguajes como

Jython que se usan dentro de ImageJ

Nuevamente intenamos hacer un threshold sobre nuestra imagen:

[19]:
```
img_iterable = ij.op().transform().flatIterableView(ij.py.to_java(img))
img_th = ij.op().threshold().otsu( img_iterable )
```

[20]:
```
type(img_th)
```

[20]:
```
<java class 'net.imglib2.img.array.ArrayImg'>
```

Volvemos a obtener el mismo error:

[21]:
```
ij.py.show(img_th, cmap="gray")
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
<ipython-input-21-573d3931768a> in <module>
----> 1 ij.py.show(img_th, cmap="gray")

D:\Programas\Anaconda\envs\ImageJ\lib\site-packages\imagej\__init__.py in␣
 ↪show(self, image, cmap)
    615             from matplotlib import pyplot
    616
--> 617             pyplot.imshow(self.from_java(image),␣
 ↪interpolation='nearest', cmap=cmap)
    618             pyplot.show()
    619

D:\Programas\Anaconda\envs\ImageJ\lib\site-packages\imagej\__init__.py in␣
 ↪from_java(self, data)
    550             except Exception as exc:
    551                 _dump_exception(exc)
--> 552                 raise exc
    553             return sj.to_python(data)
    554

D:\Programas\Anaconda\envs\ImageJ\lib\site-packages\imagej\__init__.py in␣
 ↪from_java(self, data)
    547                 if self._ij.convert().supports(data,␣
 ↪RandomAccessibleInterval):
    548                     rai = self._ij.convert().convert(data,␣
 ↪RandomAccessibleInterval)
--> 549                     return self.rai_to_numpy(rai)
    550             except Exception as exc:
    551                 _dump_exception(exc)

D:\Programas\Anaconda\envs\ImageJ\lib\site-packages\imagej\__init__.py in␣
 ↪rai_to_numpy(self, rai)
```

```
    307             """
    308             result = self.new_numpy_image(rai)
--> 309             self._ij.op().run("copy.rai", self.to_java(result), rai)
    310             return result
    311

D:\Programas\Anaconda\envs\ImageJ\lib\site-packages\imagej\__init__.py in␣
 ↪to_java(self, data)
    372             """
    373             if self._is_memoryarraylike(data):
--> 374                 return imglyb.to_imglib(data)
    375             if self._is_xarraylike(data):
    376                 return self.to_dataset(data)

D:\Programas\Anaconda\envs\ImageJ\lib\site-packages\imglyb\util.py in␣
 ↪to_imglib(source)
    137 def to_imglib(source):
    138     scyjava.start_jvm()
--> 139     return ReferenceGuardingRandomAccessibleInterval(_to_imglib(source) ␣
 ↪ReferenceGuard(source))
    140
    141

D:\Programas\Anaconda\envs\ImageJ\lib\site-packages\imglyb\util.py in␣
 ↪_to_imglib(source)
    147
    148     if not source.dtype in numpy_dtype_to_conversion_method:
--> 149         raise NotImplementedError("Cannot convert dtype to ImgLib2 type␣
 ↪yet: {}".format(source.dtype))
    150     elif source.flags['CARRAY']:
    151         return numpy_dtype_to_conversion_method[source.
 ↪dtype](long_address, *long_arr_source)

NotImplementedError: Cannot convert dtype to ImgLib2 type yet: bool
```

Ya que nuestros esfuerzos hasta ahora no han rendido frutos, probaremos otra opción: utilizar macros con ayuda de este tutorial.

El lenguaje de macros de ImageJ nació como una acercamiento de alto nivel a las funcionalidades de ImageJ, en general con la implementación de las operaciones (que previamente intentamos pero no parecen funcionar del todo bien) se esperaba hacer a un lado las macros, pero su implementación es tan popular y utilizada aún hoy en día que se quedó como una opción permanente.

ImageJ tiene una opción para grabar los pasos que realizamos en el launcher de windows, de modo que todos los filtros y pasos de procesamiento que realicemos se traducen a código de macro que podemos implementar fácilmente en PyImageJ con la función run_macro().

Al momento que llamamos ImageJ colocamos como argumento “headless = False” esto quiere decir que a partir del llamado de algunas funciones utilzadas de las macros se mandará a llamar al

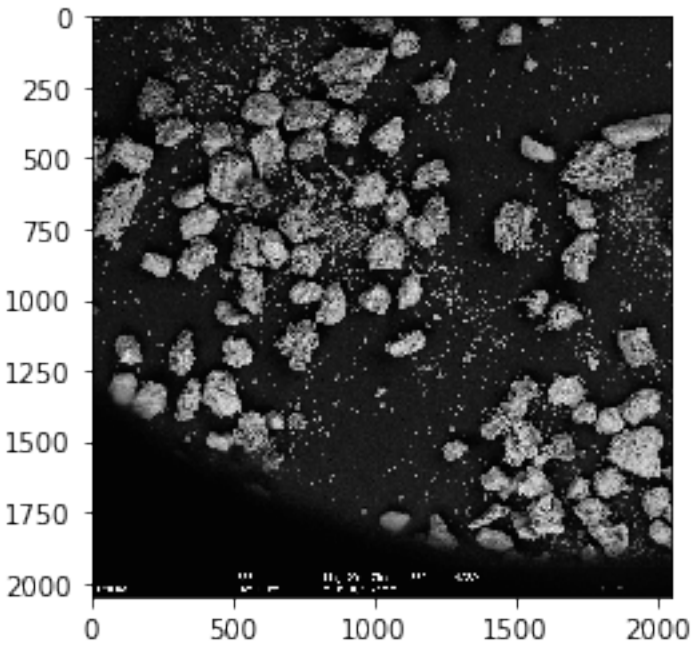
programa ImageJ que está corriendo localmente en python y nos mostrará algunas operaciones y resultados. Sin más que agregar, se prueba utilizar una macro para abrir, filtrar y analizar las particulas de una imagen:

[22]:
```
from scyjava import jimport #para observar la ventana
WindowManager = jimport('ij.WindowManager') # un método de la api original␣
↪(https://imagej.nih.gov/ij/developer/api/ij/ij/WindowManager.html)
ij.py.run_macro("""open("C:/Users/JMCas/Downloads/dust_promété001.tif");""")␣
↪#Macro sencilla para abrir la imagen
blobs = WindowManager.getCurrentImage() #extraemos la imagen de la ventana, ya␣
↪que no la abrimos localmente
print(blobs)
```

```
img["dust_promété001.tif" (-6), 8-bit, 2048x2048x1x1x1]
```

Podemos llamar la imagen al mismo tiempo que esta abierta en la ventana:

[23]:
```
ij.py.show(blobs, cmap = 'gray')
```




Declaramos la macro y ejecutamos:

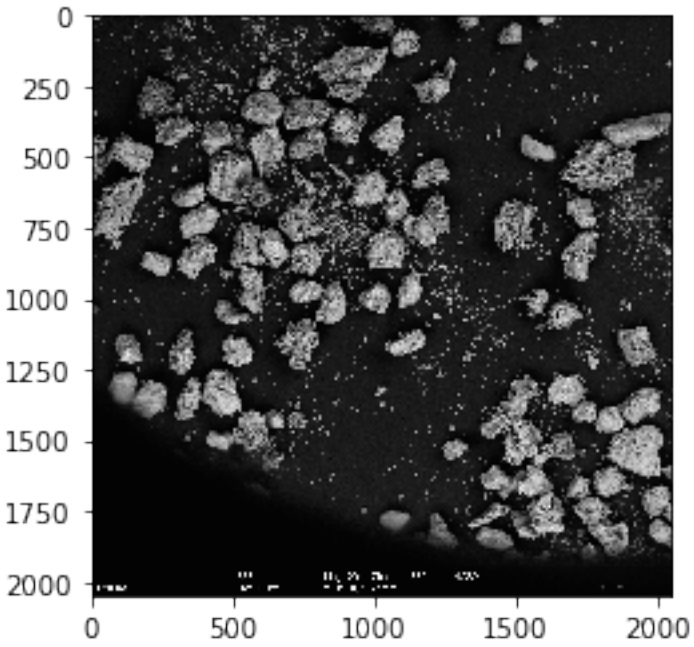
[24]:
```
macro = """
run("8-bit");
setAutoThreshold("Triangle");
//run("Threshold...");
//setThreshold(0, 151);
setOption("BlackBackground", true);
```

```
run("Convert to Mask");
run("Watershed");
run("Set Measurements...", "area mean center perimeter fit shape display␣
 ↪redirect=None decimal=5");
run("Set Scale...", "distance=1324 known=1 pixel=1 unit=mm global");
run("Analyze Particles...", "size=.00-1 circularity=0.10-1.00 show=[Overlay␣
 ↪Outlines] display exclude clear include");
saveAs("Results","results.csv" );
"""
ij.py.run_macro(macro)
```

[24]:
```
<java object 'org.scijava.script.ScriptModule'>
```

Automaticamente se abre la ventana con resultados del análisis de particulas, además de guardan en un archivo csv al que podemos acceder con excel. Mostramos la imagén filtrada:

[25]:
```
result = WindowManager.getCurrentImage()
result = ij.py.show(result, cmap='gray')
```




Para correr otras macros lo recomendable es cerrar la ventana con la que estabamos trabajando:

[26]:
```
ij.window().clear()
```

Sin embargo este procedimiento no cierra del todo las ventanas abiertas, solo los datos en memoria:

[27]:
```
print(ij.py.from_java(ij.window().getOpenWindows()))
```

[]

Para cerrar las ventanas automaticamente podemos añadir codigo a la macro como sigue: * run("Clear Results"); * Close)();

Un tutorial con un procemiento similar se encuentra aquí, aunque se recurre a una combinación de macros y Jython. En caso de no usar el *recorder* de Fiji, podemos encontrar una lista de macros aquí.

Aunque este primer acercamiento utilizando PyImageJ tuvo ciertos resultados, las limitaciones de tratar de combinar un software basado en Java con Python y la poca documentación encontrada solos nos dieron un mejor motivo para incursionar en herramientas diseñadas especificamente para Python, de modo que realizamos otro acercamiento, está vez con el módulo OpenCV.

## 3 Segundo Acercamiento: OpenCV

Open CV es la abreviatura de Open source vision library, como su nombre lo indica es una librería construida para proveer de una herramienta para las aplicaciones que involucren visión para así acelerar el uso de la percepción a partir de la tecnología en productos.

Mediante sus más de 2500 algoritmos, es posible usar el programa para reconocer rostros, clasificar acciones en videos, seguir movimientos de cámara, objetos móviles, o encontrar imágenes similares a otra en una base de datos. Así mismo el software gana terreno en aplicaciones tales como detección de intrusos en cámaras de vigilancia, navegación de robots.

La interacción con esta librería puede darse por medio de C++, Phyton, Java y Matlab. Para las aplicaciones de este proyecto se optó por la interacción por medio de Phyton. A su vez Phyton es un lenguaje de programación de alto nivel, el cual destaca por la ra de su aplicación, así como el mantenimiento del programa, además de la compatibilidad con múltiples herramientas y librerías.

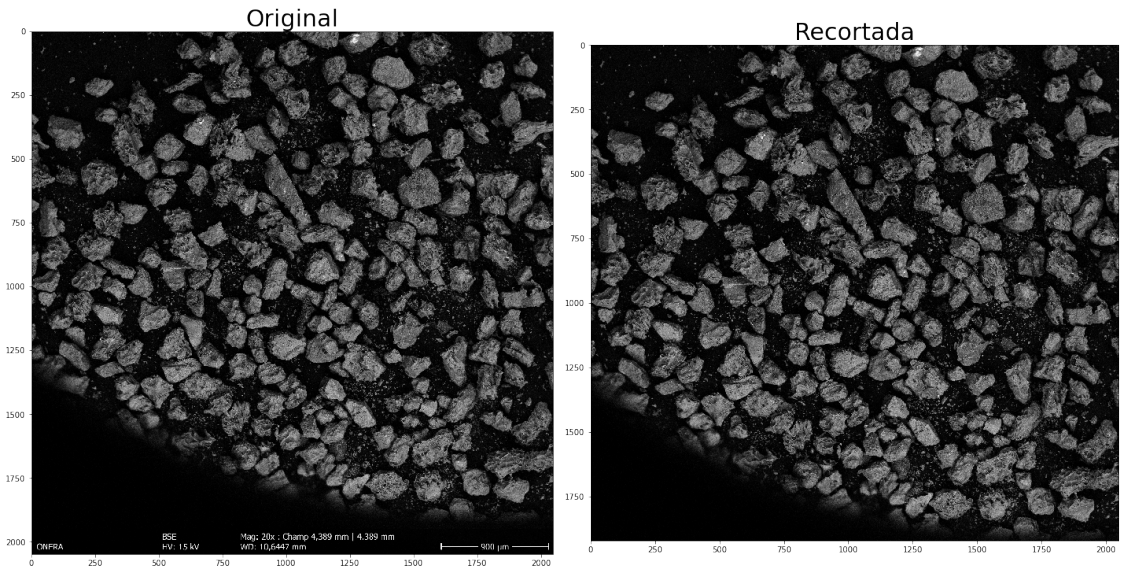
**Nota**: *OpenCV puede ejecutarse desde Jupyter pero las imágenes se verán como una ventana de windows, se añaden comando de skimage y matplotlib para poder observarlas en Jupyter sin necesidad de ejecutar el código, en caso de usar una IDE como Spider se puede trabajar sin ellos.*

[1]:
```
#Importamos módulos
import cv2
import numpy as np
import skimage
from skimage import color
import pandas as pd

#leemos la imagen, se pasa la ruta y el modo de lectura (escala de grises) como␣
↪argumentos
img = cv2.imread('C:/Users/JMCas/Downloads/dust_promete004.tif',cv2.
↪IMREAD_GRAYSCALE)

cropped = img[0:1920] # Recortamos la imagen hasta cierta altura en pixeles␣
↪para excluir la escala (area efectiva)
cropped2 = cropped.copy() # Creamos una copia pues OpenCV reescribira la imagen␣
↪al realizar contornos
```

```
#Mostramos las imagenes
cv2.namedWindow('img', cv2.WINDOW_NORMAL)
cv2.resizeWindow('img', 1200, 1000)

cv2.imshow("img", img)
cv2.waitKey(0)

cv2.imshow("img", cropped)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

[2]:
```
#### Codigo opcional

from skimage import io
import matplotlib.pyplot as plt

fig = plt.figure()
fig.set_figwidth(20)
fig.set_figheight(10)
ax = fig.add_subplot(1, 2, 1)
imgplot = io.imshow(img)
ax.set_title('Original', size = 30)
ax = fig.add_subplot(1, 2, 2)
imgplot = io.imshow(cropped)
ax.set_title('Recortada', size = 30)
```

[2]: Text(0.5, 1.0, 'Recortada')



## 3.1 Método Treshold

Es un filtro para la imagen, en el cual se define un valor del threshold, para cada pixel de la imagen se evalúa, de modo que, si es mayor que el valor colocado en el treshold, pasa por el filtro estableciéndose como 1, pero si es menor se establece en 0. Para realizar un threshold, es preciso que la imagen se encuentre en escala de grises.

La función encargada de realizar este proceso consta de 4 argumentos, de los cuales, la imagen es el primero de ellos, el segundo es el valor del threshold, el cual será usado para hacer la clasificación de los pixeles, el tercer argumento es el valor máximo, el cual se asigna a los pixeles que superen el valor asignado al threshold, finalmente, el cuarto argumento es para seleccionar el tipo de threshold que se aplicará a la imagen, existiendo:

- cv.THRESH_BINARY
- cv.THRESH_BINARY_INV
- cv.THRESH_TRUNC
- cv.THRESH_TOZERO
- cv.THRESH_TOZERO_INV

Los resultados que presenta cada tipo de threshold ante la misma entrada pueden verse en la imagen anterior con fines comparativos.

Así mismo, las siguientes gráficas representan la forma en que actúan los distintos métodos ante una misma entrada y valores del threshold. La función será llamada partiendo de la siguiente forma general, solo indicando el tipo de threshold a utilizar:

```
cv.threshold(src, thresh, maxval, type[, dst]) ->   retval, dst
```

Para los propósitos del presente proyecto se usó principalmente la función threshold binaria.

### 3.1.1 Threshold binario

se define por la siguiente función.

```
dst(x,y)={maxval if src(x,y)>thresh
               0 otherwise
```

Lo cual se traduce en establecer un valor para el pixel siempre y cuando supere el valor establecido para el threshold y establecerse en 0 en caso contrario, cabe recordar que esos valores se establecen desde el momento en que se llama a la función.

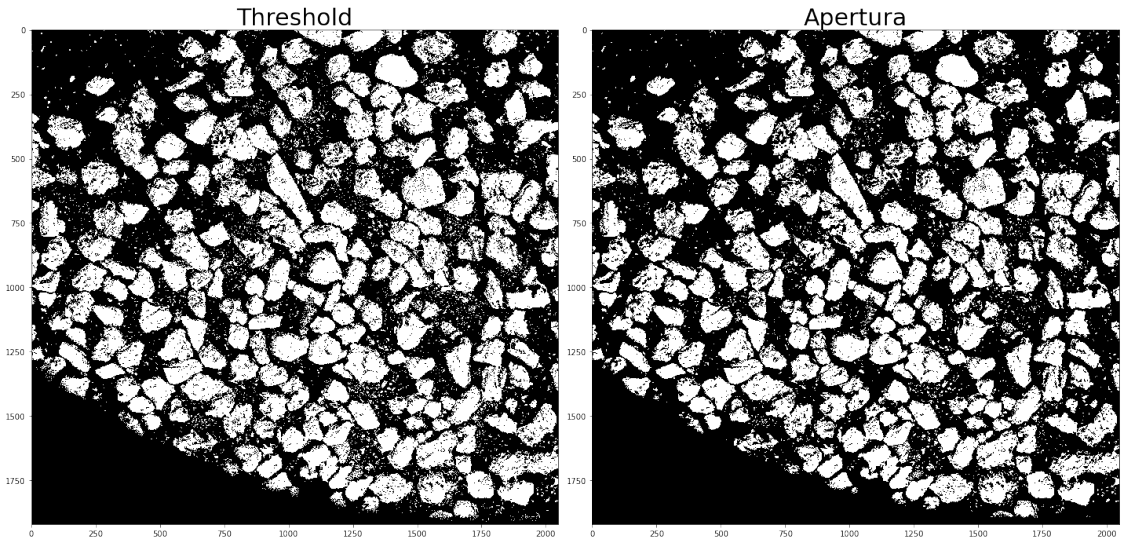
Nota: Otros métodos.

[3]:
```
#Aplicamos el método de Threshold binario, métodos adapativos y de otsu no nos␣
 ↪ayudan mucho
#ya que queremos los bordes de las particulas
ret1, thresh = cv2.threshold(cropped, 40 , 255, cv2.THRESH_BINARY)

#creamos una matriz que funciona de kernel para la operación de apertura
# Info:
# https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html
```

```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3,3))

opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 1)

#Mostramos las imagenes
cv2.namedWindow('img', cv2.WINDOW_NORMAL) #Mostramos las imagenes
cv2.resizeWindow('img', 1200, 1000)

cv2.imshow('img', thresh)
cv2.waitKey(0)

cv2.imshow('img', opening)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

[4]:
```
#### Codigo opcional

from skimage import io
import matplotlib.pyplot as plt

fig = plt.figure()
fig.set_figwidth(20)
fig.set_figheight(10)
ax = fig.add_subplot(1, 2, 1)
imgplot = io.imshow(thresh)
ax.set_title('Threshold', size = 30)
ax = fig.add_subplot(1, 2, 2)
imgplot = io.imshow(opening)
ax.set_title('Apertura', size = 30)
```

[4]:
```
Text(0.5, 1.0, 'Apertura')
```




### 3.1.2 Transformaciones morfológicas

Las transformaciones morfológicas son operaciones sencillas que tienen como base la forma de la imagen. Estas se aplican a imágenes que ya se encuentran en binario. Esta función necesita de 2 entradas, una es la imagen original y la segunda es el elemento estructurante o kernel que decide la naturaleza de la operación. Los operadores morfológicos son erosión, dilatación (siendo los básicos), apertura, cierre, gradiente, etc.

**Tipos de transformaciones morfológicas** 1.- Erosión Este elemento estructurante tiene como idea básica la misma que la erosión del suelo, siendo que el kernel erosiona los límites del objeto en primer plano, este se desliza a través de la imagen, es decir si un pixel en la imagen original (ya sea 1 o 0) se considera 1 solo si todos los pixeles debajo del kernel son 1; de lo contrario, se erosiona (se reduce a 0). Entonces, lo que sucede es que todos los píxeles cercanos al límite se descartarán dependiendo del tamaño del kernel. Entonces, el grosor o tamaño del objeto en primer plano disminuye o simplemente la región blanca disminuye en la imagen. Es útil para eliminar pequeños ruidos blancos, separar dos objetos conectados, etc.

2.- Dilatación Es justo lo opuesto a la erosión. Aquí, un elemento de píxel es '1' si al menos un píxel debajo del núcleo es '1'. Por lo tanto, aumenta la región blanca en la imagen o aumenta el tamaño del objeto en primer plano. Normalmente, en casos como la eliminación de ruido, la erosión va seguida de dilatación. Porque la erosión elimina los ruidos blancos, pero también encoge nuestro objeto. Entonces lo dilatamos. Dado que el ruido se ha ido, no volverán, pero nuestra área de objetos aumenta. También es útil para unir partes rotas de un objeto.

3.- Apertura Esta función es solo una erosión seguida de dilatación, siendo útil para eliminar el ruido.

4.- Clausura Esta función es la inversa a la apertura, siendo que primero se realiza una dilatación seguida de una erosión. Es útil para cerrar pequeños agujeros dentro de los objetos en primer plano o pequeños puntos negros en el objeto.

Nota: Otras operaciones morfológicas

### 3.1.3 Elemento estructurante

Creamos manualmente elementos de estructuración en los ejemplos anteriores con la ayuda de Numpy. Tiene forma rectangular. Pero en algunos casos, es posible que necesite granos de forma elíptica / circular. Entonces, para este propósito, OpenCV tiene una función:

`cv.getStructuringElement ()`

Simplemente pasa la forma y el tamaño del kernel, obtienes el kernel deseado.

**Tipos de kernel**

- Núcleo rectangular

```
>>> cv.getStructuringElement (cv.MORPH_RECT, (5,5))
matriz
([[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1]], dtype = uint8)
```

- Núcleo elíptico

```
>>> cv.getStructuringElement (cv.MORPH_ELLIPSE, (5,5))
matriz
([[0, 0, 1, 0, 0],
[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1],
[0, 0, 1, 0, 0]], dtype = uint8)
```

- Núcleo en forma de cruz

```
>>> cv.getStructuringElement (cv.MORPH_CROSS, (5,5))
matriz
([[0, 0, 1, 0, 0],
[0, 0, 1, 0, 0],
[1, 1, 1, 1, 1],
[0, 0, 1, 0, 0],
[0, 0, 1, 0, 0]], dtype = uint8
```

Para detectar lo mejor posible las partículas era necesario hacer uso de estas funciones que nos ofrece OpenCV, como se pudo observar en la descripción de los kernel, la erosión facilita el detectar objetos ya que elimina el ruido alrededor del objeto a observar, mientras que la dilatación aumenta el tamaño del objeto permitiendo recuperar área que se haya perdido por una erosión y que sea de utilidad, como en nuestro caso que buscamos calcular el área de las partículas. Pero como se pudo observar no es necesario realizar dos funciones diferentes, basta con usar la apertura o la clausura ya que estas funciones realizan una erosión y dilación seguidas, es por eso que nos decantamos por usar estas funciones en el código.

### 3.1.4 Floodfill

`cv.floodFill(image, mask, seedpoint, newval [loDiff, upDiff, Flags])`

Rellena un componente conectado con el color dado. La función cv::floodFill llena un componente conectado comenzando desde el punto inicial con el color especificado. La conectividad está determinada por la cercanía de color / brillo de los píxeles vecinos. El píxel en (x+y) se considera que pertenece al dominio si:

- En caso de una imagen en escala de grises y rango flotante:

$$src(x', y') - loDiff \leq src(x, y) \leq src(x', y') + upDiff$$

- En caso de una imagen en escala de grises y rango fijo:

$$src(seedpoint.x, seedpoint.y) - loDiff \leq src(x, y) \leq src(seedpoint.x, seedpoint.y) + upDiff$$

- En el caso de una imagen en color y rango flotante:

$$\begin{aligned} src(x', y')_r - loDiff_r \leq src(x, y)_r \leq src(x', y')_r + upDiff_r \\ src(x', y')_g - loDiff_g \leq src(x, y)_g \leq src(x', y')_g + upDiff_g \\ src(x', y')_b - loDiff_b \leq src(x, y)_b \leq src(x', y')_b + upDiff_b \end{aligned}$$

- En el caso de una imagen en color y rango fijo:

$$src(seedpoint.x, seedpoint.y)_r - loDiff_r \leq src(x, y)_r \leq src(seedpoint.x, seedpoint.y)_r + upDiff_r$$

$$src(seedpoint.x, seedpoint.y)_g - loDiff_g \leq src(x, y)_g \leq src(seedpoint.x, seedpoint.y)_g + upDiff_g$$

$$src(seedpoint.x, seedpoint.y)_b - loDiff_b \leq src(x, y)_b \leq src(seedpoint.x, seedpoint.y)_b + upDiff_b$$

Donde:

$$src(x', y')$$

Es el valor de uno de los píxeles vecinos que ya se sabe que pertenece al componente. Es decir, para agregarlo al componente conectado, un color / brillo del píxel debe estar lo suficientemente cerca para:

- Color / brillo de uno de sus vecinos que ya pertenecen al componente conectado en caso de rango flotante.
- Color / brillo del punto de siembra en caso de rango fijo.

Utilice estas funciones para marcar un componente conectado con el color especificado en el lugar, o crear una máscara y luego extraer el contorno, o copiar la región a otra imagen, y así sucesivamente.

**Parámetros**

1. **Image**: Entrada / salida Imagen de 1 o 3 canales, 8 bits o de punto flotante. La función lo modifica a menos que se establezca el indicador FLOODFILL_MASK_ONLY en la segunda variante de la función. Vea los detalles a continuación.
2. **Mask**: Máscara de operación que debe ser una imagen de un solo canal de 8 bits, 2 píxeles más ancha y 2 píxeles más alta que la imagen. Dado que se trata de un parámetro de entrada y de salida, debe asumir la responsabilidad de inicializarlo. El relleno de inundación no puede atravesar píxeles distintos de cero en la máscara de entrada. Por ejemplo, una salida de detector de bordes se puede utilizar como máscara para detener el llenado en los bordes. En la salida, los píxeles de la máscara correspondientes a los píxeles rellenos de la imagen se establecen en 1 o en el valor a especificado en las banderas como se describe a continuación. Además, la función llena el borde de la máscara con unos para simplificar el procesamiento interno. Por lo tanto, es posible usar la misma máscara en múltiples llamadas a la función para asegurarse de que las áreas rellenas no se superpongan.
3. **Seedpoint**: Punto de partida.
4. **NewVal**: Nuevo valor de los píxeles de dominio repintados.
5. **LoDiff**: Diferencia máxima de brillo / color más baja entre el píxel observado actualmente y uno de sus vecinos que pertenece al componente, o un píxel semilla que se agrega al componente.
6. **UpDiff**: Diferencia máxima de brillo / color superior entre el píxel observado actualmente y uno de sus vecinos que pertenece al componente, o un píxel semilla que se agrega al componente.
7. **Rect**: Parámetro de salida opcional establecido por la función al rectángulo delimitador mínimo del dominio repintado.
8. **Banderas**: Banderas de operación. Los primeros 8 bits contienen un valor de conectividad. El valor predeterminado de 4 significa que solo se consideran los cuatro píxeles vecinos más cercanos (los que comparten un borde). Un valor de conectividad de 8 significa que se considerarán los ocho píxeles vecinos más cercanos (los que comparten una esquina). Los siguientes 8 bits (8-16) contienen un valor entre 1 y 255 con el que llenar la máscara (el valor predeterminado es 1). Por ejemplo, 4 | (255 « 8) considerará 4 vecinos más cercanos y llenará la máscara con un valor de 255.

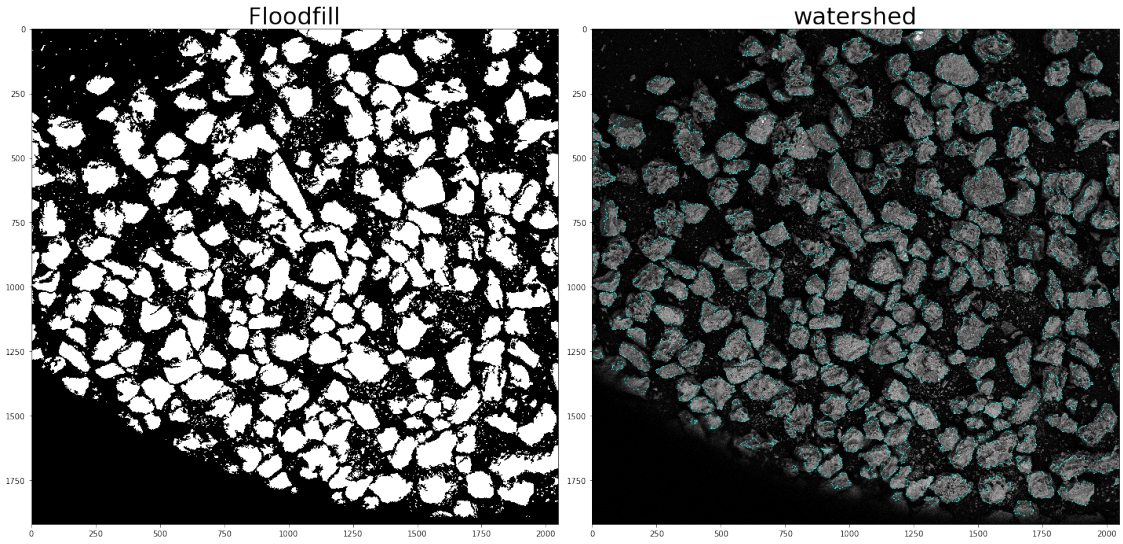
[5]:
```
# Queremos rellenar los huecos internos de las particulas
# El metodo de cierre (closing) es buena opcion pero perdemos info y precision,␣
 ↪así que optamos por el método floodfill
# https://learnopencv.com/filling-holes-in-an-image-using-opencv-python-c/

im_floodfill = opening.copy()

h,w = opening.shape[:2]
mask = np.zeros((h+2,w+2), np.uint8)

cv2.floodFill(im_floodfill, mask, (0,0), 255)

im_floodfill_inv = cv2.bitwise_not(im_floodfill)

im_out = opening | im_floodfill_inv
```

```
# método de watershed
# https://docs.opencv.org/4.0.1/d3/db4/tutorial_py_watershed.html


sure_bg = cv2.dilate(im_out ,kernel,iterations=2)

dist_transform = cv2.distanceTransform(im_out ,cv2.DIST_L2,3)

ret2, sure_fg = cv2.threshold(dist_transform,0.15*dist_transform.max(),255,0)
sure_fg = np.uint8(sure_fg)

unknown = cv2.subtract(sure_bg,sure_fg)

ret3, markers = cv2.connectedComponents(sure_fg)

markers = markers+1

markers[unknown==255] = 0

img1 = cv2.cvtColor(cropped, cv2.COLOR_GRAY2BGR)

markers = cv2.watershed(img1,markers)

img1[markers == -1] = [0,255,233]

img2 = color.label2rgb(markers, colors = ['white'],  bg_label=1)


#Mostramos las imagenes
cv2.namedWindow('img', cv2.WINDOW_NORMAL) #Mostramos las imagenes
cv2.resizeWindow('img', 1200, 1000)

cv2.imshow('img', im_out)
cv2.waitKey(0)

cv2.imshow('img', img1)
cv2.waitKey(0)

cv2.imshow('img', img2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[6]:
```
#### Codigo opcional

from skimage import io
import matplotlib.pyplot as plt
```

```
fig = plt.figure()
fig.set_figwidth(20)
fig.set_figheight(10)
ax = fig.add_subplot(1, 2, 1)
imgplot = io.imshow(im_out)
ax.set_title('Floodfill', size = 30)
ax = fig.add_subplot(1, 2, 2)
imgplot = io.imshow(img1)
ax.set_title('watershed', size = 30)
```

[6]: `Text(0.5, 1.0, 'watershed')`




### 3.1.5 Watershed

La función *watershed* realiza una segmentación de imágenes basada en marcadores utilizando el algoritmo de cuencas hidrográficas watershed.

Antes de pasar la imagen a la función, se debe delinear aproximadamente las regiones deseadas en los marcadores de imagen con índices positivos (mostrado dentro de este código). Entonces, cada región se representa como uno o más componentes conectados con los valores de píxel 1, 2, 3, etc. que se marcan usando las herramientas de contornos (findcontours y drawcontours). Los marcadores son referencias de las futuras regiones de la imagen. Todos los demás píxeles de los marcadores, cuya relación con las regiones delineadas no se conoce y deben ser definidos por el algoritmo, deben establecerse en ceros. En la salida de la función, cada píxel de los marcadores se establece en un valor de los componentes de referencia que crea un "llenado de piscina" en la imagen.

En Python tiene la siguiente estructura:

```
cv.watershed(image, markers)
```

- image: es la imagen de entrada en 8 bits
- markers: Es la salida de mapa de pixeles "llenados"

### 3.1.6 Contornos

Los contornos se pueden explicar simplemente como una curva que une todos los puntos continuos (a lo largo del límite), que tienen el mismo color o intensidad. Los contornos son una herramienta útil para el análisis de formas y la detección y reconocimiento de objetos. Las imágenes binarias tienen mejores resultados. Por lo tanto, antes de encontrar contornos, es conveniente usar el método de threshold o la detección de bordes.

Para esta tarea usamos las funciones findContours y drawContours incluidas en las funciones de dibujo de Open CV es importante recordar que las funciones trabajan encontrando las partes blancas de una imagen que contrastan con un fondo negro, por ello, la imagen debe ser filtrada con anterioridad.

En Python la funcion findContours tiene la siguiente estructura:

```
cv.findContours(imagen, modo, método)
```

**Los modos del algoritmo recuperación de contorno son:**

- cv.RETR_EXTERNAL, recupera solo los bordes externos
- cv.RETR_LIST, recupera todos los contornos sin establecer jerárquias.
- cv.RETR_CCOMP, recupera todos los contornos y los organiza en una jerarquía de dos niveles.

**Los métodos de aproximación de contornos son:**

- cv.CHAIN_APPROX_NONE, que guarda todos los puntos del contorno.
- cv.CHAIN_APPROX_SIMPLE que hace una aproximación con los puntos horizontales, verticales y diagonales de la figura
- cv.CHAIN_APPROX_TC89_L1 que aplica el método Teh-Chin

La función drawContours tiene la siguiente estructura:

```
cv.drawContours(imagen, contours, contourIdx, color)
```

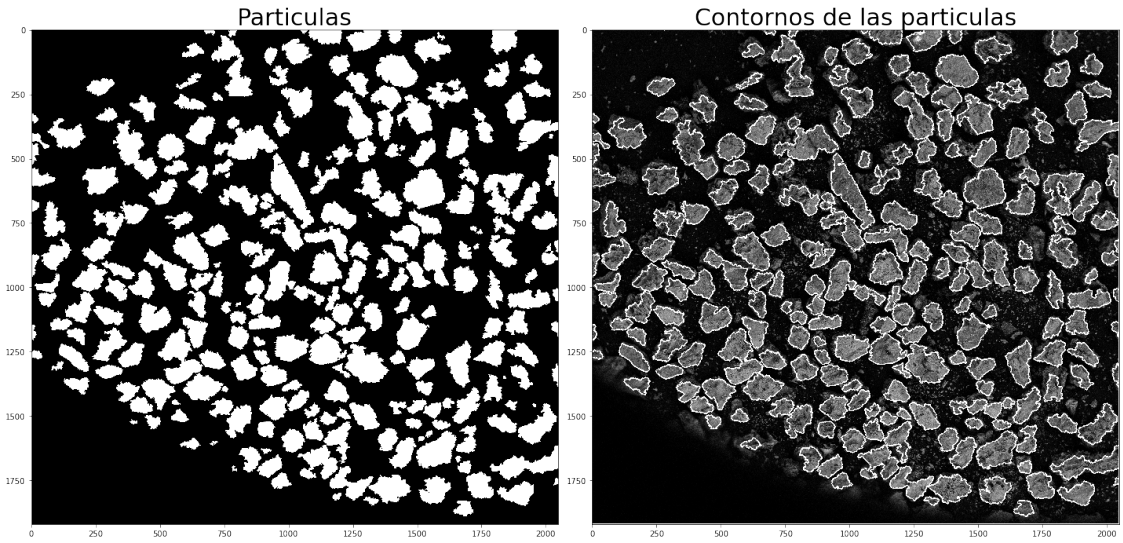
- imagen: Imagen de destino.
- contours: Todos los contornos de entrada. Cada contorno se almacena como un vector de puntos.
- contourIdx: Parámetro que indica un contorno a dibujar. Si es negativo, se dibujan todos los contornos.
- color: Color de los contornos en formato RGB o BGR.

[10]:
```
img3 = cv2.cvtColor(cropped, cv2.COLOR_GRAY2BGR)

cv_image = skimage.img_as_ubyte(img2)

buffer = cv2.cvtColor(cv_image, cv2.COLOR_BGRA2GRAY)

ret, m2 = cv2.threshold(buffer, 0, 255, cv2.THRESH_BINARY|cv2.THRESH_OTSU)
```

```
contours, hierarchy = cv2.findContours(m2,cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
#Estos objetos de tipo contorno son filtrables por su tamaño
big_contours = [c for c in contours if cv2.contourArea(c) >= 1000]

#Ordenamos los contornos
sorted_contours= sorted(big_contours , key=cv2.contourArea, reverse= True)

#Dibujamos objetos de tipo contorno sobre los dibujados anteriormente
for c in big_contours :
    cv2.drawContours(img3, c, -1, (255,255,255), 3)



#Mostramos las imagenes
cv2.namedWindow('img', cv2.WINDOW_NORMAL)
cv2.resizeWindow('img', 1200, 1000)

cv2.imshow('img', m2)
cv2.waitKey(0)

cv2.imshow('img', img3)
cv2.waitKey(0)


cv2.destroyAllWindows()
```

[13]:
```
#### Codigo opcional

from skimage import io
import matplotlib.pyplot as plt

fig = plt.figure()
fig.set_figwidth(20)
fig.set_figheight(10)
ax = fig.add_subplot(1, 2, 1)
imgplot = io.imshow(cv_image)
ax.set_title('Particulas', size = 30)
ax = fig.add_subplot(1, 2, 2)
imgplot = io.imshow(img3)
ax.set_title('Contornos de las particulas', size = 30)
```

[13]:
```
Text(0.5, 1.0, 'Contornos de las particulas')
```




### 3.1.7 Características de los contornos

1. **Momentos**: Ayudan a calcular algunas características como el centro de masa del objeto, área u objeto. A partir de los momentos calculados en la función cv.moments() pueden extraerse datos útiles tales como el centroide, el área etc.

```
M = cv.moments(cnt)
#Centroides
cx = int(M['m10']/M['m00'])
cy = int(M['m01']/M['m00'])
```

2. **Area**: O superficie:

```
area = cv.contourArea(cnt)
```

3. **Perimetro**: O longitud de arco, el segundo argumento indica si es un contorno cerrado o solo un arco:

```
perimeter = cv.arcLength(cnt,True)
```

4. **Cáscara Convexa**: Similar a la aproximación de contorno, la función cv.convexHull() revisa una curva por defectos de convexidad y los repara. En términos generales, las curvas convexas consideran los aspectos siguientes:

- Puntos: Son los contornos por los que se pasa.
- Cáscara: es la salida de la función, aunque normalmente se evita
- Sentido: Es un señalamiento de orientación, en caso de ser verdadero, la convexión se da en sentido de las agujas del reloj, en caso contrario, se ubica en dirección opuesta.
- Puntos de retorno: Su valor por defecto es verdadero, devuelve las coordenadas de los puntos de cáscara si es falso retorna los índices de los puntos de contorno correspondientes a la cáscara

```
hull = cv.convexHull(cnt)
```

### 3.1.8 Propiedades de los contornos

1. **Ratio de aspecto**: Es la relación del ancho con la altura del rectángulo que contiene al objeto.

$$RatiodeAspecto = \frac{Ancho}{Alto}$$

2. **Solidez**: Es la relación del área de contorno con su área convexa.

$$Solidez = \frac{reaContorno}{AreaCscaraConvexa}$$

3. **Eje mayor**: Segmento de recta localizado entre los vértices de la Elipse. Para el caso de un contorno con coordenadas en la imagen:

$$EjeMayor = |\frac{X_{EjeMayor} - X}{2}|$$

4. **Eje menor**: Segmento de recta perpendicular al eje mayor cuyos extremos se localizan sobre la elipse. Para el caso de un contorno con coordenadas en la imagen:

$$EjeMenor = |\frac{Y_{EjeMenor} - Y}{2}|$$

[17]:
```
#Declaramos arreglos vacios donde se guardara nuestra información
area = []
perimeter = []
aspect_ratio = []
solidity = []
major = []
minor = []

#Vamos a iterar sobre cada contorno para calcular las propiedades geometricas

for i,c in enumerate(sorted_contours):
    #Calculamos un rectangulo que se ajuste al contorno para obtener su area y␣
 ↪perimetro
    x,y,w,h = cv2.boundingRect(c)
    aspect_ratio.append(float((w)/h))
    #Guardamos en el arreglo
    area.append(cv2.contourArea(c))
    perimeter.append(cv2.arcLength(c,True))
    #Calculamos la cascara convexa
    area1=cv2.contourArea(c)
    hull=cv2.convexHull(c)
    hull_area=cv2.contourArea(hull)
    #Guardamos la solidez
    solidity.append(float((area1)/hull_area))
```

```
    #Calculamos una elipse ajustada al contorno
    (a1, a2),(d1,d2),angle=cv2.fitEllipse(c)
    #Obtenemos los parámetros y los guardamos
    long=a1-d2
    small=a2-d2
    major.append(abs(long/2))
    minor.append(abs(small/2))

    #Creamos etiquetas para enumerar las particulas en la imagen utilizando sus␣
 ↪momentos
    M= cv2.moments(c)
    cx= int(M['m10']/M['m00'])
    cy= int(M['m01']/M['m00'])
    x,y,w,h= cv2.boundingRect(c)
    #Ponemos el texto en las particulas
    cv2.putText(img3, text= str(i+1), org=(cx-10,cy-30),
            fontFace= cv2.FONT_HERSHEY_SIMPLEX, fontScale=1, color= (0,0,255),
            thickness=2, lineType=cv2.LINE_AA)
    cv2.putText(cropped2, text= str(i+1), org=(cx-10,cy-30),
            fontFace= cv2.FONT_HERSHEY_SIMPLEX, fontScale=1, color= (255␣
 ↪,255,255),
            thickness=2, lineType=cv2.LINE_AA)

#### creamos un dataframe con la ayuda de la libreria de pandas para crear␣
 ↪exportar como tabla
data = { "areas" : area ,
         "perimetros": perimeter,
         "aspect_ratio": aspect_ratio,
         "solidity": solidity,
         "Semieje mayor": major,
         "Semieje menor": minor
        }

archivo = pd.DataFrame(data)

archivo.index +=1

#Aqui se guarda como .csv
archivo.to_csv("Particulas.csv", index = True)

#Finalmente dibujamos los cuadrados de area minima:

for i in range(0,len(big_contours)):
    rect = cv2.minAreaRect(big_contours [i])
    box = cv2.boxPoints(rect)
    box = np.int0(box)
    img_result = cv2.drawContours(cropped2,[box],0,(255,0,255),2)
```

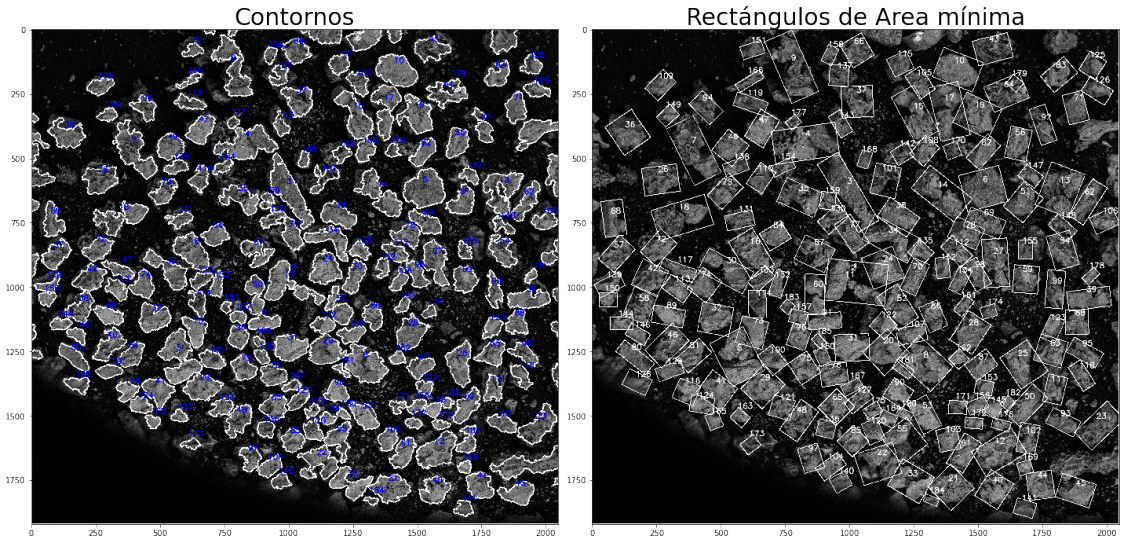
```
#Mostramos las imagenes
cv2.namedWindow('img', cv2.WINDOW_NORMAL)
cv2.resizeWindow('img', 1200, 1000)

cv2.imshow('img', img3)
cv2.waitKey(0)

cv2.imshow('img', cropped2)
cv2.waitKey(0)


cv2.destroyAllWindows()
```

[18]:
```
#DataFrame que contiene los parámetros de las partículas y es guardado como␣
↪archivo .csv
archivo
```

[18]:
```
          areas    perimetros  aspect_ratio  solidity  Semieje mayor  \
1     3928193.0   7932.000000      1.066667  1.000000     704.258789
2     3856140.0  12084.116668      1.066667  0.981658     666.754181
3       17898.0   1016.479352      0.574324  0.818400     346.485519
4       17484.0   1060.562613      1.852941  0.747355     306.273697
5       17450.0   1050.621488      0.807339  0.731442     184.217873
..          …           …            …         …              …
181      1311.0    272.190906      1.229508  0.621180     564.773716
182      1202.0    193.279218      1.090909  0.813812     786.599918
183      1146.5    184.350286      0.916667  0.803715     355.320337
184      1080.0    182.450791      1.742857  0.831729     631.885628
185      1058.5    162.894442      1.170732  0.867623     416.930359

     Semieje menor
1       736.258789
2       688.172791
3       160.564774
4       108.462143
5       534.983772
..             …
181     623.419590
182     703.291080
183     516.366907
184     885.952339
185     581.111267

[185 rows x 6 columns]
```

[15]:
```
#### Codigo opcional

from skimage import io
import matplotlib.pyplot as plt

fig = plt.figure()
fig.set_figwidth(20)
fig.set_figheight(10)
ax = fig.add_subplot(1, 2, 1)
imgplot = io.imshow(img3)
ax.set_title('Contornos enumerados', size = 30)
ax = fig.add_subplot(1, 2, 2)
imgplot = io.imshow(cropped2)
ax.set_title('Rectángulos de Area mínima', size = 30)
```

[15]:
```
Text(0.5, 1.0, 'Rectángulos de Area mínima')
```




## 4 Conclusiones

El procesamiento de imágenes tiene como objetivo mejorar el aspecto de las imágenes y hacer más evidentes en ellas ciertos detalles que se desean hacer notar. Se tuvo como antecedente del trabajo deseado, un algoritmo desarrollado por un grupo de compañeros de la facultad, basado en el lenguaje de programación Java, no obstante, debido a la caducidad de las herramientas, los algoritmos pueden caer en la obsolescencia. El lenguaje de programación Python cuenta con grandes recursos que pueden satisfacer numerosas necesidades, en esta ocasión, aplicamos estas herramientas a la creación de un algoritmo de procesamiento de imágenes. La naturaleza de las imágenes que procesamos es amorfa lo que conlleva un reto al momento de obtener datos como sus características geométricas. Sin embargo, debido al apoyo de las herramientas antes mencionadas, tales como las funciones de la biblioteca OpenCV, se pudieron obtener los datos deseados

**Bibliografía**

1. “Calibración de datos de nubes de ceniza para los volcanes Mexicanos”, Laboratorio Nacional de Observación de la Tierra, Instituto de Geografía, UNAM. Abril de 2019. Recuperado de: https://www1.cenapred.unam.mx/SUBCUENTA/6a%20SESI%C3%93N%20EXTRAORDINARIA/3.%20G