# Beyond IoT: Inteligencia sensorial y tecnología inteligente, modulo 4

## ¿Agua limpia o sucia? ¿Cómo de sucia?

Autor: Diego Hinojosa Córdova

Fecha: 23-Nov-2020

## I. Tomar imágenes simples.

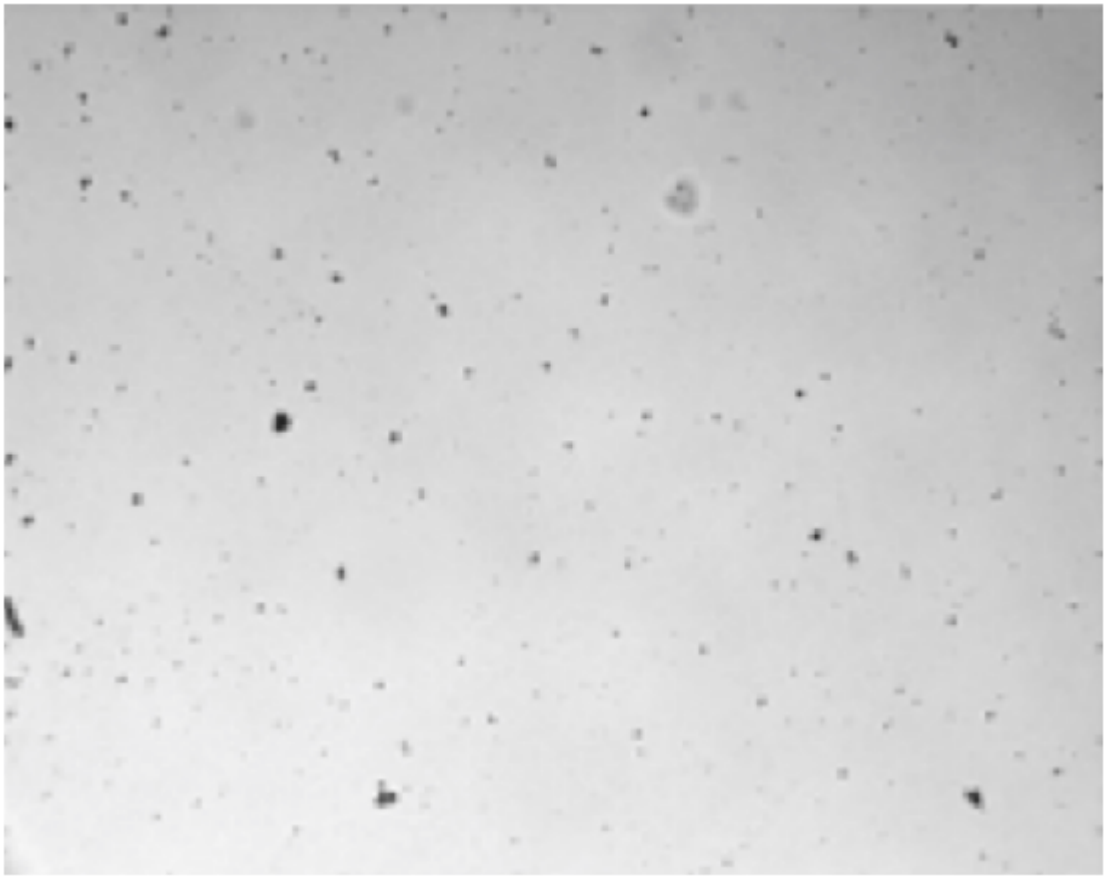
```
# If you are running from Google Colaboratory, you probably
# need to install pytorch.
!pip3 install torch torchvision  # This will take a while installing.
# You might need to do Runtime -> Restart Runtime after running this cell
# on Google Colab.
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.6/dist-packages (1.7.0+cu
Requirement already satisfied: torchvision in /usr/local/lib/python3.6/dist-packages (0
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-packages (from to
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from tor
Requirement already satisfied: dataclasses in /usr/local/lib/python3.6/dist-packages (fr
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.6/dist-packages (
```

```
import torch, matplotlib
import numpy as np
import matplotlib.pyplot as plt
import imageio
matplotlib.rc('image', cmap = 'gray')

# Lista de imágenes
img1_simple = 'https://github.com/dhinojosac/mit_beyond_iot/blob/master/mod4/M4-I-Dirty-Water
img2_simple = 'https://github.com/dhinojosac/mit_beyond_iot/blob/master/mod4/M4-I-Dirty-Water
img3_diry = 'https://github.com/dhinojosac/mit_beyond_iot/blob/master/mod4/M4-I-Dirty-Water-E
img4_life = 'https://github.com/dhinojosac/mit_beyond_iot/blob/master/mod4/M4-I-Dirty-Water-E
img5_life = 'https://github.com/dhinojosac/mit_beyond_iot/blob/master/mod4/M4-I-Dirty-Water-E

# Seleccionamos una imagen de arriba para ser procesada
img_url = img2_simple

img = imageio.imread(img_url)

# Transform into pytorch tensor
```

```
# Transform into pytorch tensor.
img = torch.tensor(img, dtype=torch.float) / 255.0

# Show the image size.
print('Image size: ', img.shape)

# Plot the image.
def display_image(img,figsize=(16,8)):
  plt.figure(figsize=(16,8)); plt.imshow(img)
  plt.grid(False);  plt.axis('off'); plt.show()

  # Plot the image augmented size
def display_bimage(img):
  plt.figure(figsize=(20,15)); plt.imshow(img)
  plt.grid(False);  plt.axis('off'); plt.show()

display_image(img)
```

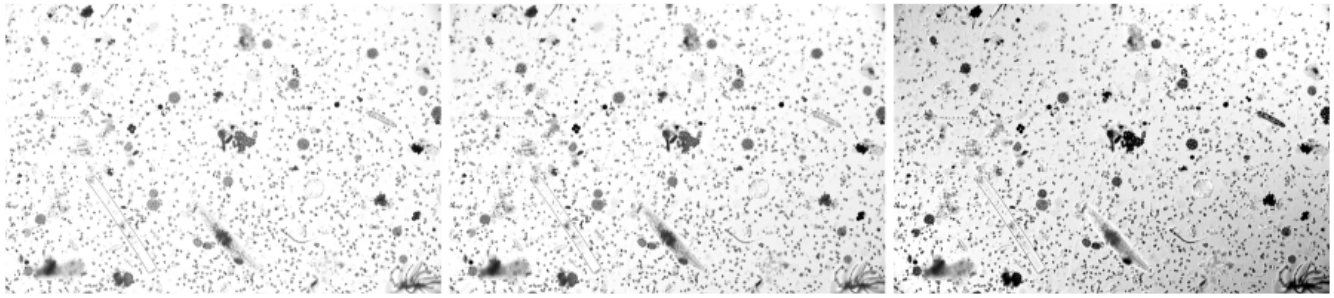
```
Image size:  torch.Size([317, 400, 4])
```



## II. Filtrar las imágenes con filtros de pasa bajos o filtros de paso alto.

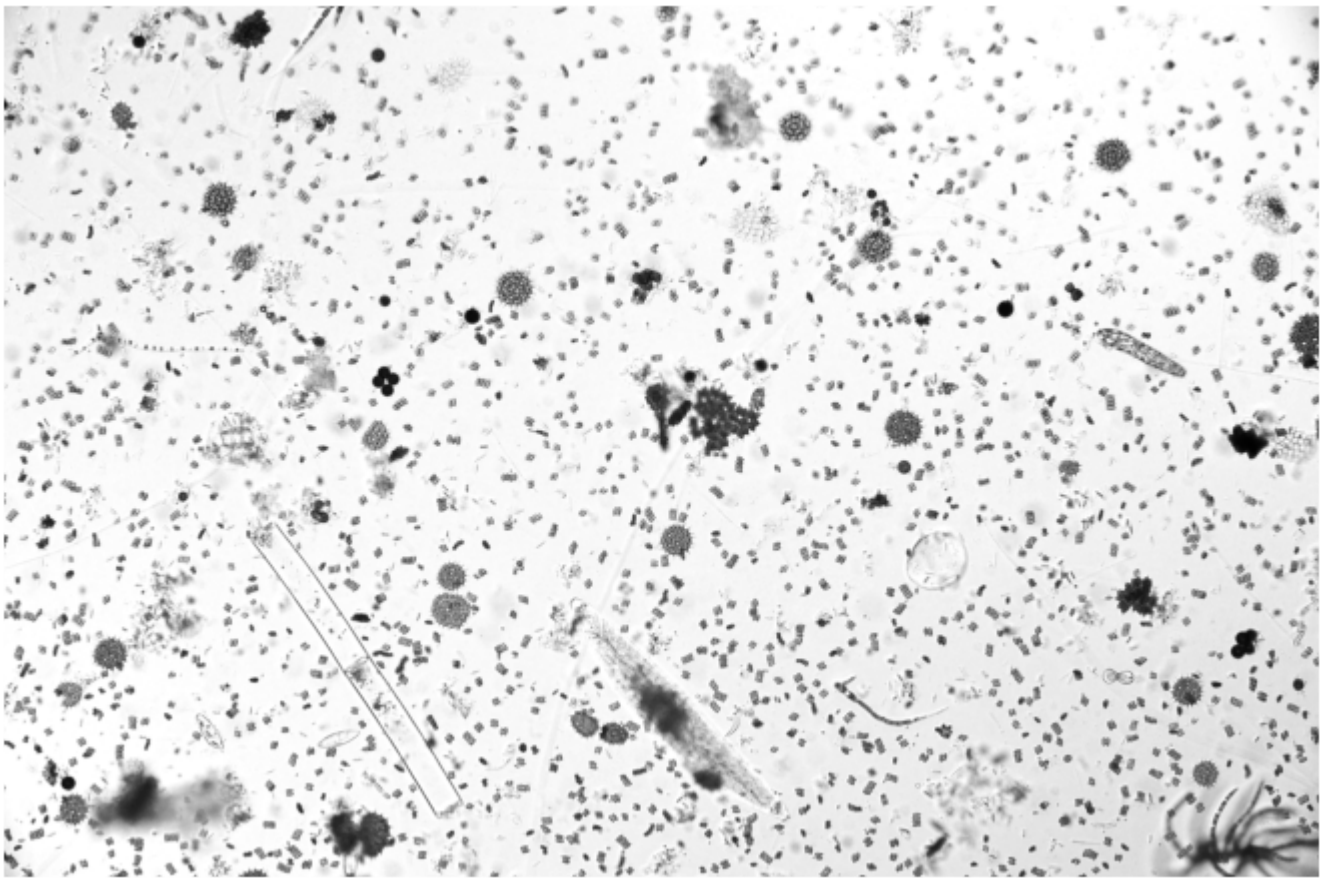
```
# Descomponemos la imagen en sus 3 canales RGB
r_image = img[:, :, 0]
g_image = img[:, :, 1]
b_image = img[:, :, 2]
```

```
# pixeles cercanos a 1 es oscuro, cercanos a 0 es claro
space = torch.ones(r_image.shape[0],100)
display_image(torch.cat((r_image,space, g_image,space, b_image), 1))
# Also, please review what torch.cat does (nothing to do with felines).
```



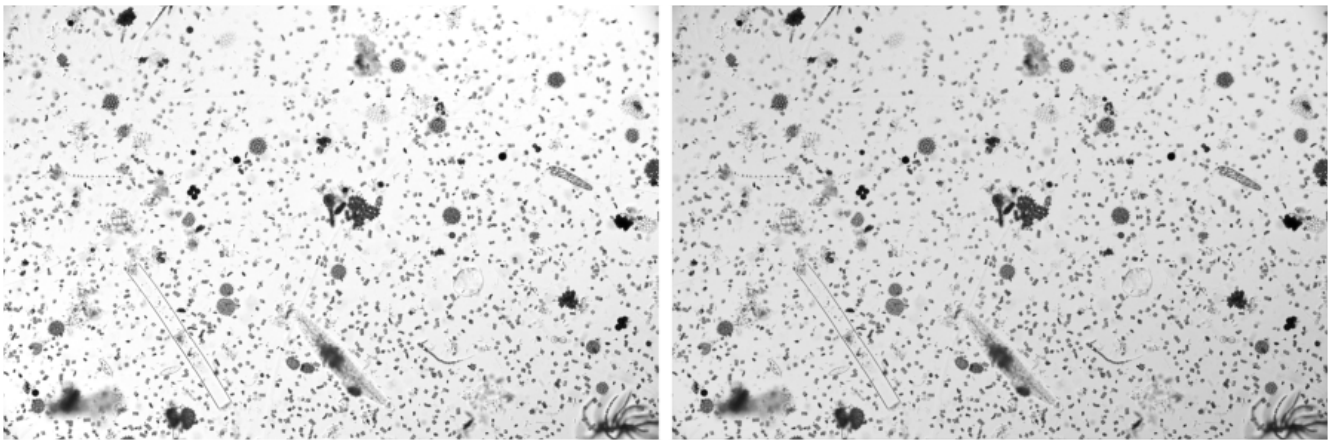
Tomamos la imagen anterior, la convertimos a escala de grises.

```
gray_img = (img[:, :, 0] + img[:, :, 1] + img[:, :, 2]) / 3.0  # convertimos escala de grises
#input_image = gray_img[500:1500, 500:1500] # cortamos la imagen
input_image = gray_img
display_image(input_image)
```



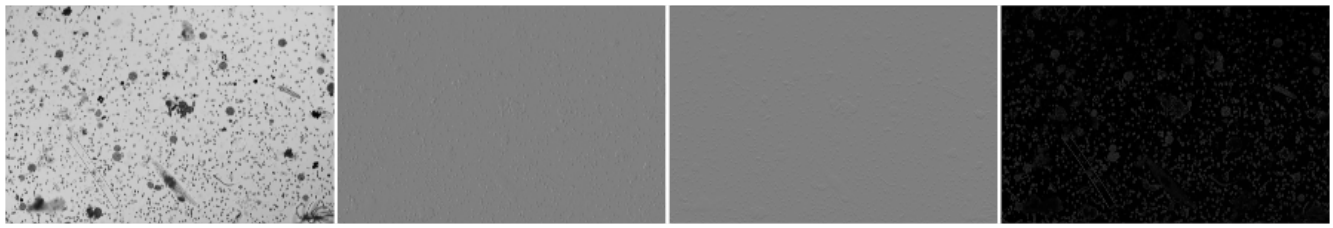
Aplicamos una convolución.

```
# kernel de tamaño 3x3 que calcula el promedio entre los pixeles que rodean al central
# esto es un filtro pasa bajos. Suaviza la imagen, reduce los cambios abruptos en los
# valores de cada pixel
kv = 1.0 / 10.0
blur_weights = torch.Tensor([[kv, kv, kv],
                             [kv, kv, kv],
                             [kv, kv, kv]])


# Se usa la función F.conv2d para realizar la convolución
def SingleChannelConvolution(input_image, kernel):
    import torch.nn.functional as F
    kernel = kernel.contiguous()
    output = F.conv2d(input_image.unsqueeze(0).unsqueeze(0),
                      kernel.unsqueeze(0).unsqueeze(0), padding = 1)
    return output.squeeze()


# Execute the convolution operation
output_image = SingleChannelConvolution(input_image, blur_weights)

space = torch.ones(input_image.shape[0],100)
display_image(torch.cat((input_image,space, output_image), 1))
```

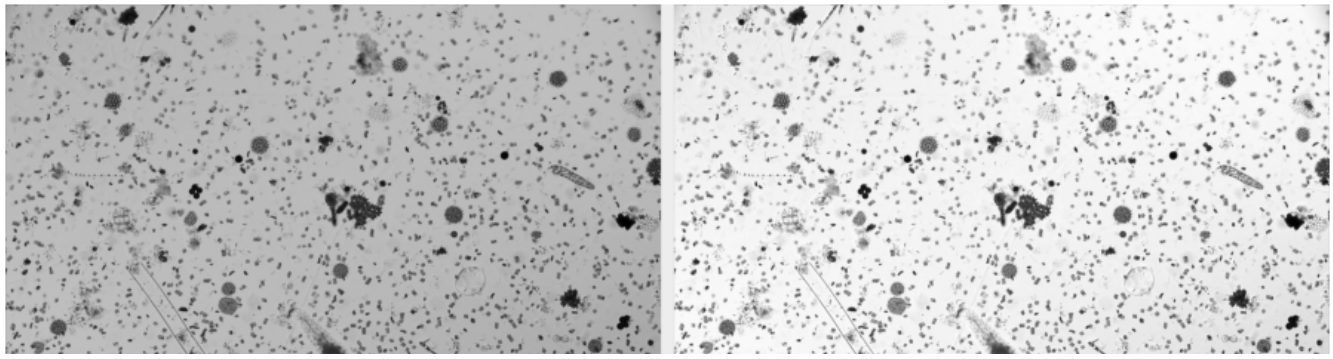


```
input_image = SingleChannelConvolution(input_image, blur_weights.clone())
input_image = SingleChannelConvolution(input_image, blur_weights.clone())

# kernel de Sobel de tamaño 3x3 para detectar bordes
weights = torch.tensor([[1.0, 0.0, -1.0],
```

```
                        [2.0, 0.0, -2.0],
                        [1.0, 0.0, -1.0]])

space = torch.ones(input_image.shape[0],100)

# Execute the convolution operation for the weights and transposed weights.
gradx = SingleChannelConvolution(input_image, weights)
grady = SingleChannelConvolution(input_image, weights.t())

# Gradient magnitude  = sqrt(gradx.^2 + grady.^2)
magnitude = torch.sqrt(gradx * gradx + grady * grady)

# Make sure the images are still between 0 and 1.
gradx = (gradx - gradx.min()).div(gradx.max() - gradx.min())
grady = (grady - grady.min()).div(grady.max() - grady.min())
magnitude = magnitude / magnitude.max()

# Display the output.
plt.figure(figsize=(16,8));
space = torch.ones(input_image.shape[0],50)
plt.imshow(torch.cat((input_image,space, gradx,space, grady,space, magnitude), 1));
plt.grid(False);plt.axis('off'); plt.show()
```

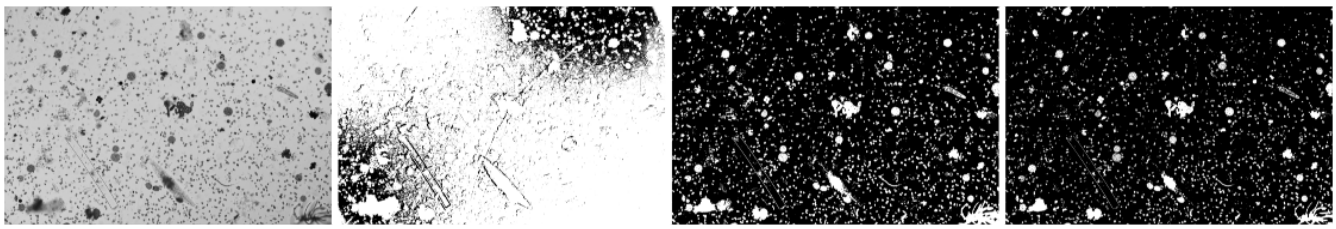


```
# Kernel 3x3 filtro pasa alto o sharpening
weights = torch.tensor([[-0.0, -0.1, 0.0],
                        [-1.0/4.0,  2, -1.0/4.0],
                        [-0.0, -0.1, -0.0]])

image_hpf = SingleChannelConvolution(input_image, weights)

space = torch.ones(input_image.shape[0],100)
display_image(torch.cat((input_image, space, image_hpf), 1))
```



## III. Conviértalas en imágenes binarias con diferentes umbrales.

```
# Binarizar imagen anterior con distintos thresholds
def Binarize(input_image, th):
  mask = input_image<th
  mask = mask.float()
  return mask

th1_im = Binarize(input_image,0.8)
th2_im = Binarize(input_image,0.6)
th3_im = Binarize(input_image,0.5)

space = torch.ones(input_image.shape[0],100)
display_bimage(torch.cat((input_image, space, th1_im, space, th2_im, space, th3_im), 1))
```

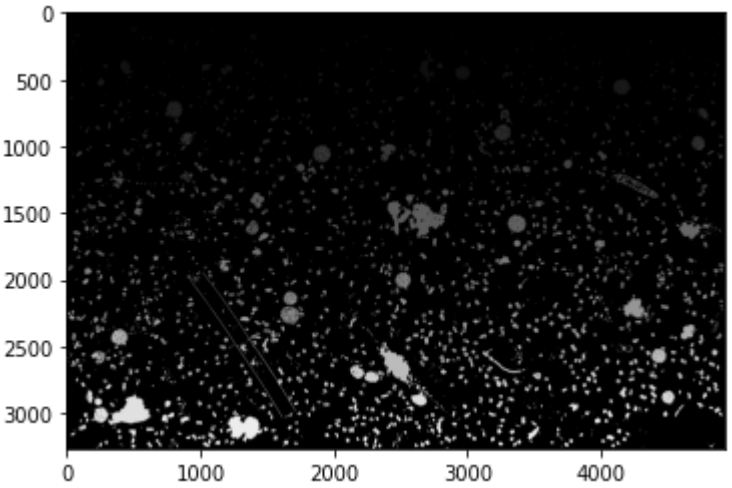


## IV. Aísle las partículas. Aplique filtrado lógico (es decir, filtros morfológicos) para limpiar la imagen y aislar, aproximadamente, cada partícula como un solo "componente conectado".

```
# Escogemos th3 que aparentemente se ve mejor
# utilizando un algoritmo de clasificación como el "connected components labeling"
```

```
# utilizando un algoritmo de clasificación como el "connected components labeling"
# podemos contar la cantidad de particulas
from scipy import ndimage

selected_image = th3_im
label_im, nb_labels = ndimage.label(selected_image)   # connected components form clusters
print("Numero de conjuntos %i" % nb_labels)
plt.imshow(label_im);  # Se puede ver como a cada cluster se le asigna un tono
                       # distinto para diferenciarlo
```

```
Numero de conjuntos 3368
```




# V. Cuente las partículas y determine sus propiedades (tamaño, forma, etc.).

1. Utilice el etiquetado de imágenes.

El algoritmo anterior se encarga de detectar cluster y luego etiquetarlos, para así poder contarlos.

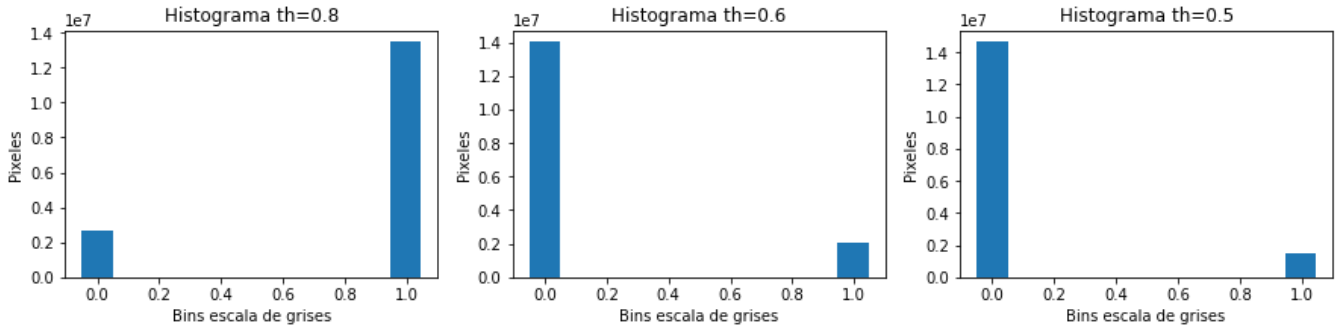
```
print("El número de partículas es:", nb_labels)
```

```
El número de partículas es: 3368
```

2. Visualice histogramas de varias propiedades o combinaciones de propiedades.

```
# Se crea histograma de la imagen seleccionada anteriormente
h1, b1 = np.histogram(th1_im, bins=256, range=(0, 1))
h2, b2 = np.histogram(th2_im, bins=256, range=(0, 1))
h3, b3 = np.histogram(th3_im, bins=256, range=(0, 1))


fig = plt.figure(figsize=(15, 3))
```

```
ax1 = fig.add_subplot(1, 3, 1)
ax2 = fig.add_subplot(1, 3, 2)
ax3 = fig.add_subplot(1, 3, 3)

ax1.bar(b1[0:-1], h1, width = 0.1)
ax2.bar(b2[0:-1], h2, width = 0.1)
ax3.bar(b3[0:-1], h3, width = 0.1)

ax1.set_title("Histograma th=0.8")
ax2.set_title("Histograma th=0.6")
ax3.set_title("Histograma th=0.5")

# recorremos todos los gráficos para agregarles etiquetas a los ejes
for ax in [ax1, ax2, ax3]:
    ax.set_ylabel("Pixeles")
    ax.set_xlabel("Bins escala de grises")

plt.show()
```




3. ¿Puede clasificar distintos tipos de partículas?

Se podría clasificar por tamaños de particulas, utilizando un algoritmo que entregue la cantidad de pixeles que componen un cluster, así poder inferir el tamaño de las particulas.

## VI. Cuente las distribuciones de las propiedades de las partículas para diferentes parámetros de procesamiento.

Se puede observar, que cambiando el threshold, los histogramas varían, cambiando la cantidad de pixeles blancos y negros, mientras más particulas, más pixeles oscuros hay. Esto puede ser un parámetro para medir la turbiedad del agua.

# VII. Evalúe su procedimiento de medición: debilidades y fortalezas.

Se puede medir la turbiedad, realizando un histograma para saber cuantos pixeles oscuros y claros hay, de esta forma saber que tan turbia está el agua.

Un problema sería que no sabemos a la distancia que estamos de las particulas o de la muestra, por lo que estaría sesgada. Lo mismo sucede si una partícula se acerca mucho al foco de la cámara, se verá más grande y afectará la medición.

En cuanto a repetibilidad, se puede tomar la posición de los cluster para poder obtener una distrubución de las partículas y saber así su varianza.

Con la precisión, se debería calibrar el sistema con muestras de agua, de tal forma de saber a posterior cuan preciso es el clasificador.

Existen otros problemas, que son la sobre posición de las particulas, creando la ilusión de ser una particula de mayor tamaño, ese es otro sesgo que tiene el sistema que debiese poder corregirse con nuevos algoritmos de procesamiento digital de imágenes.

Otro error tiene que ver con el marco que puede tener la imagen, como sucede con la imagen simple 1, esto se vería reflejado en el histograma

## Algunas Imágenes

Notar, que el procesamiento del archivo completo se puede cambiar selecionando en el inicio la imagen que se desea.