Pregunta1

November 30, 2019

[1]:

```
# Type annotations :
from typing import Tuple, List, Optional, NoReturn, Callable, Any, Dict

# Standard and OS :
import copy
import json
import glob
import os
import importlib # Required to reload a module
                 # because the Jupyter Kernel
                 # won't  really reimport by itself.
import multiprocessing as mp

# Image processing :
import cv2 as cv
import skimage
from skimage.feature import canny, peak_local_max
from skimage.util.dtype import dtype_range
from skimage.util import img_as_ubyte, img_as_float
from skimage import exposure
import skimage.morphology as morphology
from skimage.morphology import disk, skeletonize, thin, medial_axis, watershed,␣
 ↪max_tree, convex_hull_image, closing
from skimage.filters import sobel
from skimage.segmentation import felzenszwalb, slic, quickshift, watershed
from skimage.segmentation import mark_boundaries
import skimage.measure as measure
import skimage.draw as draw
from skimage.color import label2rgb
#from skimage.morphology import black_tophat, skeletonize, convex_hull_image
#from skimage.morphology import disk

from skimage.filters import rank
from skimage.measure import label, regionprops

# Numeric :
import numpy as np
import pandas as pd
from scipy import ndimage as ndi

# Visualisation :
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import seaborn as sns

# Machine-Learning :
from sklearn.cluster import KMeans

# Functional programing tools :
from functools import partial, reduce
from itertools import islice, chain, repeat
from operator import itemgetter
```

[2]:

```
# Modules defined within this repo :
import mfilt_funcs as mfs
importlib.reload(mfs)
import mfilt_funcs as mfs

import utils
importlib.reload(utils)
import utils

import forutils
importlib.reload(forutils)
from forutils import find_branch_points
```

[3]:

```
lmap = lambda x, y: list(map(x, y))
lfilter = lambda x, y: list(filter(x, y))
imread = lambda x: cv.imread(x, 0)
pad_obj = lambda x: cv.copyMakeBorder(np.float64(x.image), 10, 10, 10, 10, cv.
 ↪BORDER_CONSTANT)
pad  = lambda x: cv.copyMakeBorder(np.float64(x), 10, 10, 10, 10, cv.
 ↪BORDER_CONSTANT)
pad1 = lambda x: cv.copyMakeBorder(np.float64(x), 1, 1, 1, 1, cv.BORDER_CONSTANT)
```

[4]:

```
plt.style.use('seaborn-deep')
plt.rcParams['figure.figsize'] = (12, 8)
```

[186]:

```
#################################### Tested and functional :␣
 ↪#######################################

def chunk_pad(it, size, padval=None):
    """
        Splits a list into evenly sized chunks.
        Taken from : https://stackoverflow.com/questions/312443/
 ↪how-do-you-split-a-list-into-evenly-sized-chunks
    """
    it = chain(iter(it), repeat(padval))
    return iter(lambda: tuple(islice(it, size)), (padval,) * size)
##

def segplot(
    img: np.ndarray,
    group: skimage.measure._regionprops.RegionProperties,
    color: Optional[str] = None,
    title: Optional[str] = None
) -> NoReturn:
    """
    """
    if not color:
        color = 'red'

    fig, ax = plt.subplots(figsize=(9, 9))
    ax.imshow(img, cmap='gray')

    try:
        iter(group)
        for region in group:
            minr, minc, maxr, maxc = region.bbox
            rect = mpatches.Rectangle((minc, minr), maxc - minc, maxr - minr,
                                      fill=False, edgecolor=color, linewidth=2)
            ax.add_patch(rect)
    except:
        minr, minc, maxr, maxc = group.bbox
        rect = mpatches.Rectangle((minc, minr), maxc - minc, maxr - minr,
                                 fill=False, edgecolor=color, linewidth=2)
        ax.add_patch(rect)


    if title:
        plt.title(title)
    plt.tight_layout()
    plt.show()
##

def watershed_viz(image, distance, labels):
    """
        Constructed from the example found in :
        https://scikit-image.org/docs/dev/auto_examples/segmentation/
 ↪plot_watershed.html
    """
    fig, axes = plt.subplots(ncols=3, figsize=(9, 3), sharex=True, sharey=True)
    ax = axes.ravel()

    ax[0].imshow(image, cmap=plt.cm.gray)
    ax[0].set_title('Overlapping objects')
    ax[1].imshow(-distance, cmap=plt.cm.gray)
    ax[1].set_title('Distances')
    ax[2].imshow(labels, cmap=plt.cm.nipy_spectral)
    ax[2].set_title('Separated objects')

    for a in ax:
        a.set_axis_off()

    fig.tight_layout()
    plt.show()
##

def ez_watershed(
    image: np.ndarray,
    markers: Optional[int] = None,
    footprint: Optional[np.array] = None,
    **kw
) -> Tuple[int, int, int]:
    """
    """
    distance = ndi.distance_transform_edt(image)
    if footprint is not None:
        fp = footprint
    else:
        fp = np.ones((10, 10))

    if markers is None:
        local_maxi = peak_local_max(
            distance,
            indices=False,
            footprint=np.ones((10, 10)),
            labels=image,
            **kw
        )
        markers = ndi.label(local_maxi)[0]

    labels  = watershed(-distance, markers, mask=image)

    return markers, distance, labels
##

def auto_segment(
    img: np.ndarray,
    groups: int = 2,
    skew: Optional[float] = None,
    nonzero: bool = False,
    verbose: bool = False,
    save_file: Optional[str] = None,
    figsize: Optional[Tuple[int]] = (12, 8)
) -> np.ndarray:
    """
        Segment (by thresholding)
    """

    assert type(groups) is int, f"type(groups) == '{type(groups)}', should be␣
 ↪int."

    #Create the destination image from the image passed to the function, casting␣
 ↪it when needed.
    _floats = [np.float, np.float16, np.float32, np.float64, np.float128]
    if img.dtype in _floats:
        dst: np.ndarray = copy.deepcopy(img)
    else:
        dst: np.ndarray = copy.deepcopy(np.float64(img) / 255)

    # We perform K-Means clustering analysis :
    _intensities = img.flatten()
    _show_intensities = _intensities.copy()
    if nonzero:
        _intensities = _intensities[_intensities.nonzero()]
    _kmeans = KMeans(n_clusters=groups, random_state=0, verbose=verbose).
 ↪fit(_intensities.reshape(-1, 1))
    _centers = pd.core.frame.DataFrame({
        "means": chain.from_iterable(_kmeans.cluster_centers_)
    })
    _centers = _centers.sort_values(by=['means'])

    # We obtain our threshold values as pairwise means between cluster centers.
    _centers['k'] = _centers.rolling(2).mean()

    # If we desire to skew the thresholding process, we modify the K series :
    if skew is not None:
        _centers['k'] = _centers['k'].apply(lambda x: x + skew)

    # Create the values that will fill the image, according to the thresholds.
    _fill_vals = np.linspace(0, 1, groups, dtype=np.float64)

    # Fill the image with trehshold values.
    ks = [0] + _centers['k'].dropna().tolist()
    for j in range(len(ks) - 1):
        _mask = np.nonzero( (img > ks[j]) & (img < ks[j+1]) )
        dst[ _mask ] = _fill_vals[j]
    _mask = np.nonzero( img > ks[-1] )
    dst[ _mask ] = _fill_vals[-1]

    if verbose:
        fig = plt.figure(figsize = figsize)
        if skew is not None:
           print(f"\n\n Each one of the K's was skewed by a value of {skew}\n\n")
        lmap(lambda x: plt.axvline(x, color='r'), _centers.k.dropna())
        lmap(lambda x: plt.axvline(x, color='g'), _centers.means)
        _ = sns.distplot(_show_intensities, kde=False)
        fig2 = plt.figure(figsize = figsize)
        fig2.add_subplot(1, 2, 1)
        plt.imshow(img, cmap = 'gray')
        plt.title('Original')
        fig2.add_subplot(1, 2, 2)
        plt.imshow(dst, cmap = 'gray')
        plt.title(f"Threshold ({groups} groups)")

    return dst
##

def ref_region(
    img: np.ndarray,
    selem: Any = disk(5),
    sigma: int = 3,
    opening_se: np.ndarray = np.ones((10, 10)),
    closing_se: np.ndarray = np.ones((5, 5)),
    verbose: bool = False
):
    """
    """

    # Perform histogram equalisation :
    _img_eq = rank.equalize(img, selem=selem)

    # Perform edge detection :
    _edges = canny(_img_eq, sigma=3)
    _filled = ndi.binary_fill_holes(_edges)

    # Morphological processing :
    _eroded = utils.closing(
        utils.opening(np.float64(_filled), opening_se), closing_se
    )
    if verbose:
        utils.side_by_side(img, _img_eq, title1="Original", title2="Histogram␣
↪Equalised")
        #plt.title('Lol')
        utils.side_by_side(_img_eq, _filled, title1="Histogram Equalised",␣
↪title2="Canny Edge Detection + Filled image")
        #plt.title('Lal')
        utils.side_by_side(_filled, _eroded, title1="Canny Edge Detection +␣
↪Filled image", title2="Opening, closing")
        #plt.title('Lel')

    return _eroded
##

def subdivide_hose(
    img: np.ndarray,
    n: int = 2,
    contiguous: bool = False,
    disksize: Optional[float] = None
) -> List[np.ndarray]:
    """
        Subdivide a hose into n chunks, automatically.
    """

    _edges = canny(img)
    _label_image = label(_edges, return_num=False)
    _objs = regionprops(_label_image)

    _smallest = reduce(lambda x, y: x if x.area < y.area else y, _objs)
    _largest  = reduce(lambda x, y: x if x.area > y.area else y, _objs)

    if contiguous:
        # Sort according to columns.
        _short = np.array(sorted(_smallest.coords, key=itemgetter(1)))
        _long  = np.array(sorted(_largest.coords,  key=itemgetter(1)))
    else:
        _short = _smallest.coords
        _long  = _largest.coords

    _small_chunks = np.array_split(_short, n)
    _large_chunks = np.array_split(_long, n)

    # Create n subdivision masks :
    _masked = [np.zeros_like(img, dtype=img.dtype) for i in range(n)]

    for i in range(len(_masked)):
        for _coord in _small_chunks[i]:
            _masked[i][tuple(_coord)] = 1
        for _coord in _large_chunks[i]:
            _masked[i][tuple(_coord)] = 1

    disksize = disksize if disksize is not None else 13

    return [ cv.bitwise_and(np.uint8(img), np.uint8(morphology.dilation(_mask,␣
↪disk(disksize)))) for _mask in _masked ]
##


def plot_label_image_regions(img: np.ndarray, title: Optional[str] = None) ->␣
↪NoReturn:
    """
    """
    # label image regions
    label_image = label(img)
    image_label_overlay = label2rgb(label_image, image=img)
    font = cv.FONT_HERSHEY_SIMPLEX
    color = (255, 0, 0)
    letrero = 1
    thickness = 1

    fig, ax = plt.subplots(figsize=(10, 6))
    #ax.imshow(image_label_overlay)

    for region in regionprops(label_image):
        # draw rectangle around segmented labels
        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)
        y0, x0 = region.centroid
        y0 = int(y0)
        x0 = int(x0)
        org = (x0, y0)
        image_label_overlay = cv.putText(image_label_overlay, str(letrero), org,␣
↪font, 1, color, thickness, cv.LINE_AA)
        letrero += 1

    ax.imshow(image_label_overlay)
    ax.set_axis_off()
    plt.tight_layout()
    if title is not None:
        plt.title(title)
    plt.show()
    plt.close()
```

```python
##

def get_label_image_regions(img: np.ndarray) -> NoReturn:
    """
    """
    # label image regions
    label_image = label(img)
    image_label_overlay = label2rgb(label_image, image=img)
    font = cv.FONT_HERSHEY_SIMPLEX
    color = (255, 0, 0)
    letrero = 1
    thickness = 1

    fig, ax = plt.subplots(figsize=(10, 6))
    #ax.imshow(image_label_overlay)

    for region in regionprops(label_image):
        # draw rectangle around segmented labels
        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)
        y0, x0 = region.centroid
        y0 = int(y0)
        x0 = int(x0)
        org = (x0, y0)
        image_label_overlay = cv.putText(image_label_overlay, str(letrero), org, font, 1, color, thickness, cv.LINE_AA)
        letrero += 1

    return image_label_overlay
##

#################################### Experimental : #######################################

def my_thinning(img: np.ndarray, se: np.ndarray) -> np.ndarray:
    """
    """

    return  np.bitwise_xor(img, ndi.binary_hit_or_miss(img, se))


def prune(img: np.ndarray, n: int = 1):
    """
        This function DOES NOT WORK !
        HOW IS A DON'T CARE ELEMENT IMPLEMENTED IN PYTHON ?
        :(
    """
    # Construct all of the structuring elements needed from clockwise rotations.
    clockwise_rotations = lambda y: [y] + list(map(lambda x: np.rot90(y, x),␣
 ↪reversed(range(1, 3+1))))
    _b1 = np.array([[1, 0, 0],[1, 1, 0],[1, 0, 0]], dtype=img.dtype)
    _b2 = np.array([[1, 0, 0],[0, 1, 0],[0, 0, 0]], dtype=img.dtype)
    B = reduce(
        lambda x, y: x + y,
        lmap(clockwise_rotations, [_b1, _b2])
    )
    H = np.ones((3, 3))

    # Thinning, by all of the structuring elements :

    X1 = reduce(thinning, B, img)
    while n > 1:
        X1 = reduce(thinning, B, X1)
        n -= 1

    return X1

    # Hit or miss stage :
    #X2 = reduce(ndi.binary_hit_or_miss, B, X1)

    #return X2

    # Dilation stage :
    #X3 = cv.dilate(X2, H)
    # Hit-or-miss
    #ndi.binary_hit_or_miss
##
```

[6]:
```
ls images/
```

```
Triangulos.PNG    altoflujo.png     bajo2flujo.png    triangulos2.jpg
alto2flujo.png    bajo1flujo.png    bajo3flujo.png
```

[7]:
```
cwd  = os.path.abspath('.')
path = os.path.join(cwd, 'images')
pattern = os.path.join(path, '*flujo.png')
files = glob.glob(pattern)
files
```

[7]:
```
['/Users/gml/Documents/IX/imagenes/ProyectoAsignadoImagenes/images/altoflujo.png
',
 '/Users/gml/Documents/IX/imagenes/ProyectoAsignadoImagenes/images/bajo1flujo.pn
g',
 '/Users/gml/Documents/IX/imagenes/ProyectoAsignadoImagenes/images/bajo2flujo.pn
g',
 '/Users/gml/Documents/IX/imagenes/ProyectoAsignadoImagenes/images/bajo3flujo.pn
g',
 '/Users/gml/Documents/IX/imagenes/ProyectoAsignadoImagenes/images/alto2flujo.pn
g']
```

Todas nuestras imágenes de interés contienen la cadena de caracteres ‘flujo.png’.

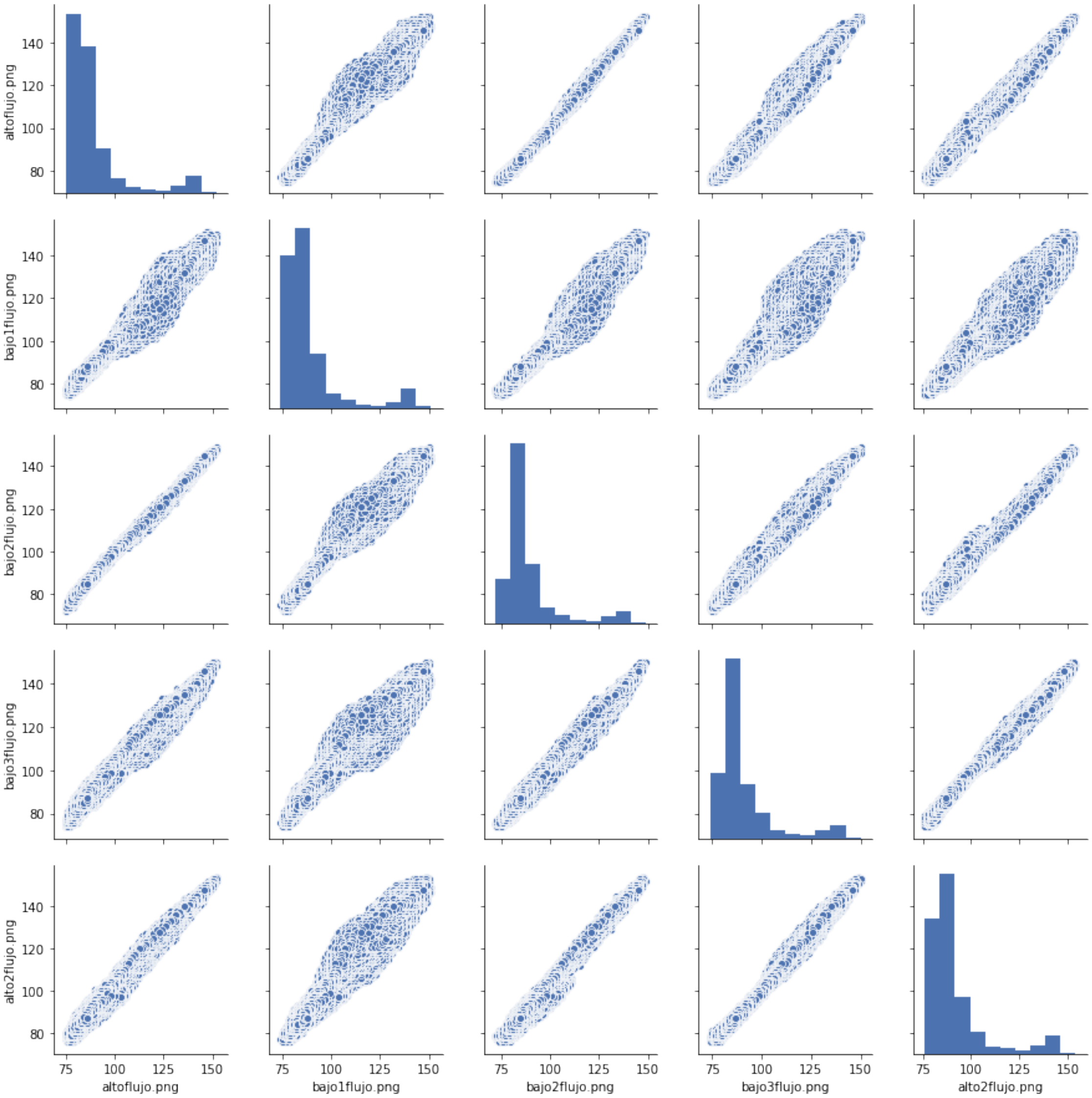
```
[8]: llaves = lmap(lambda x: os.path.split(x)[-1], files)
```

```
[9]: mangueras = {
         f"{nombre}": imread(file) for file, nombre in zip(files, llaves)
     }
```

```
[10]: intensities = pd.core.frame.DataFrame({
          key: mangueras[key].flatten() for key in mangueras.keys()
      })
```

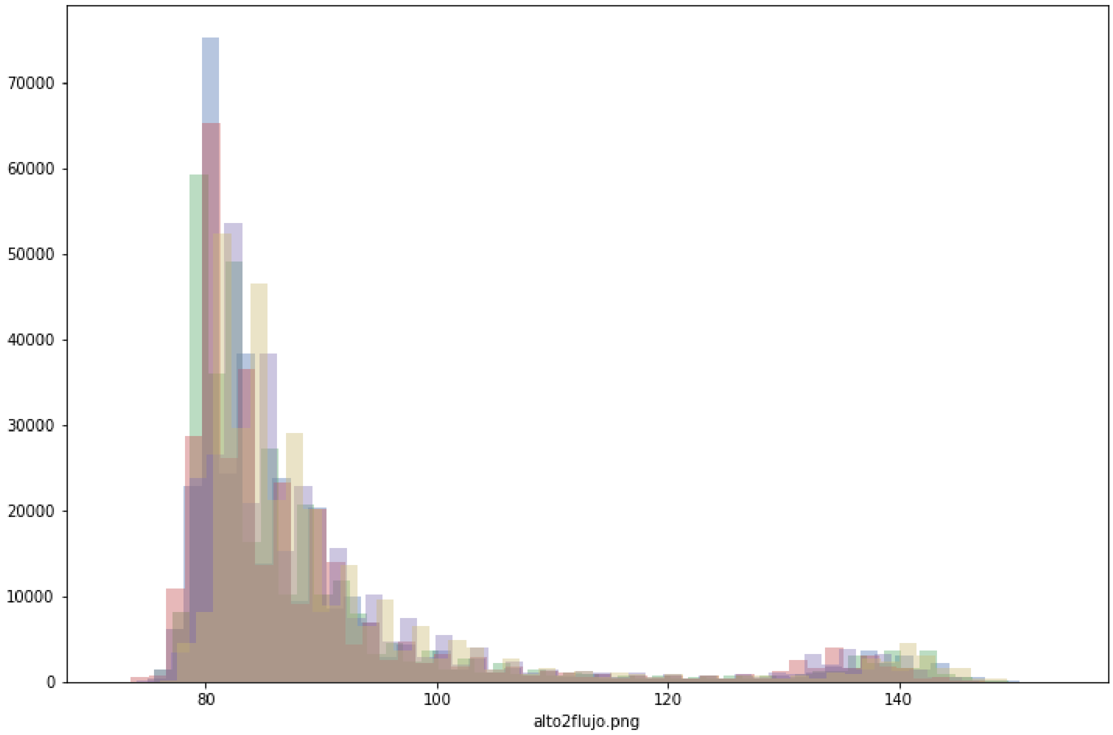
```
[9]: # SUPER SLOW !
     # Do not run !
     sns.pairplot(intensities)
```

```
[9]: <seaborn.axisgrid.PairGrid at 0x1a212a9290>
```



Podemos observar una gran correlación entre las intensidades de todas las imágenes.

[11]:
```
for i in intensities:
    sns.distplot(intensities[i],  kde=False)
```



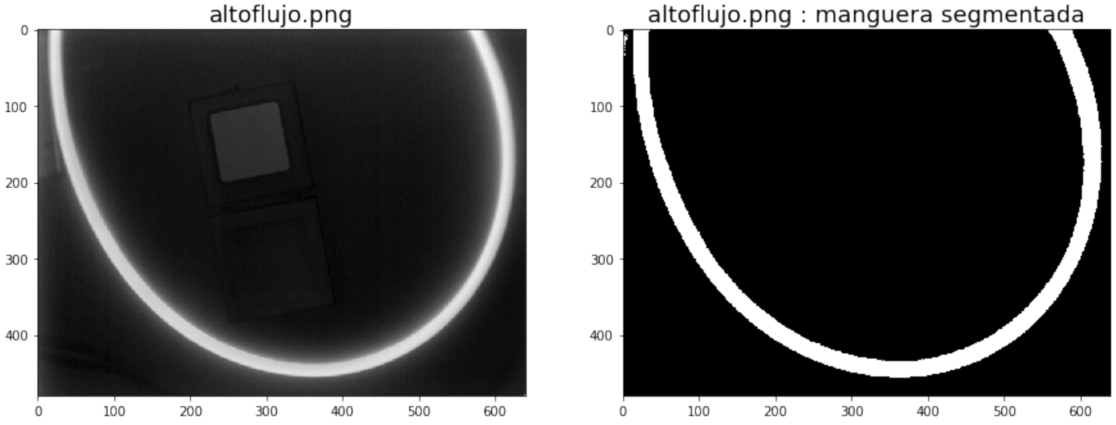
Nótese lo similares que son las distribuciones de las intensidades, independientemente de la intensidad del flujo.

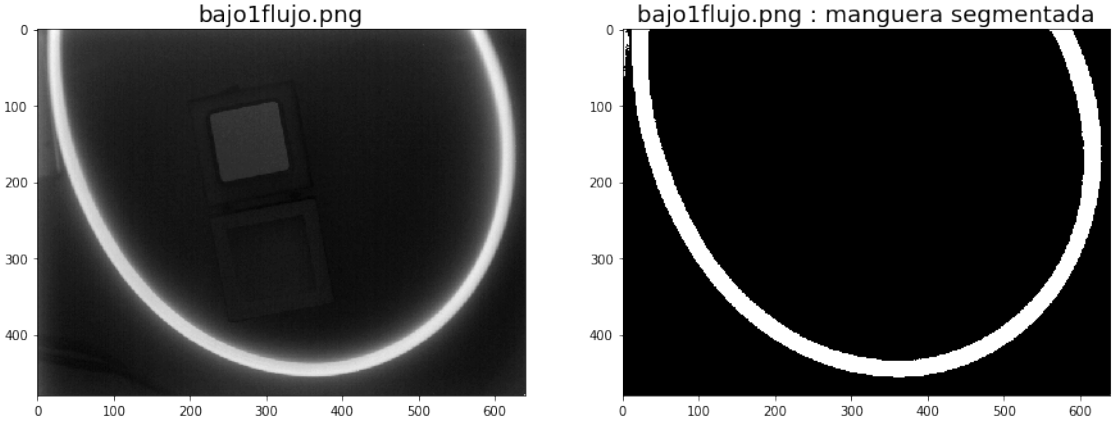
[12]:
```
mangueras_segmentadas = {
    key: auto_segment(mangueras[key], verbose=False, groups=2, skew=None) for␣
 ↪key in mangueras.keys()
}
```

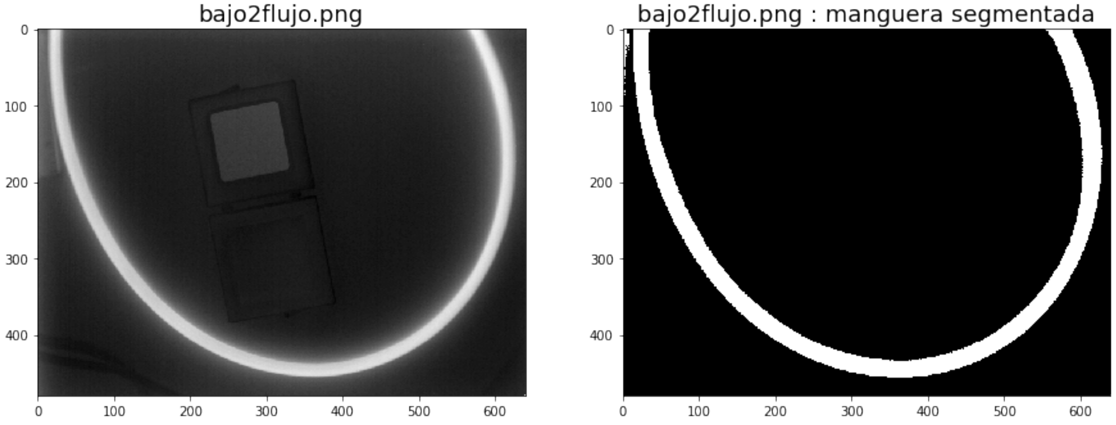
Aquí segmentamos automáticamente la región de la manguera, gracias al gran contraste que existe entre éste nuestro ente de interés y el fondo (muy claro el primero, oscuro el segundo).

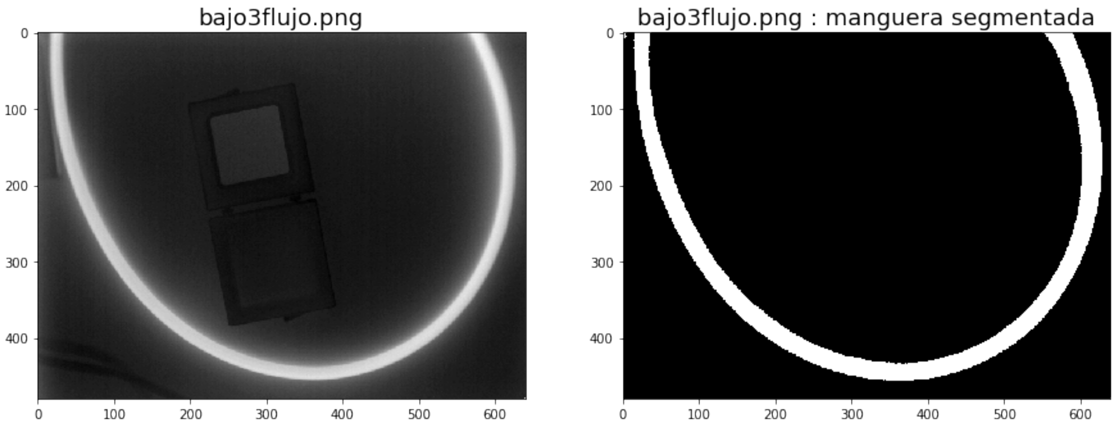
Usamos la función que diseñamos : `auto_segment()`

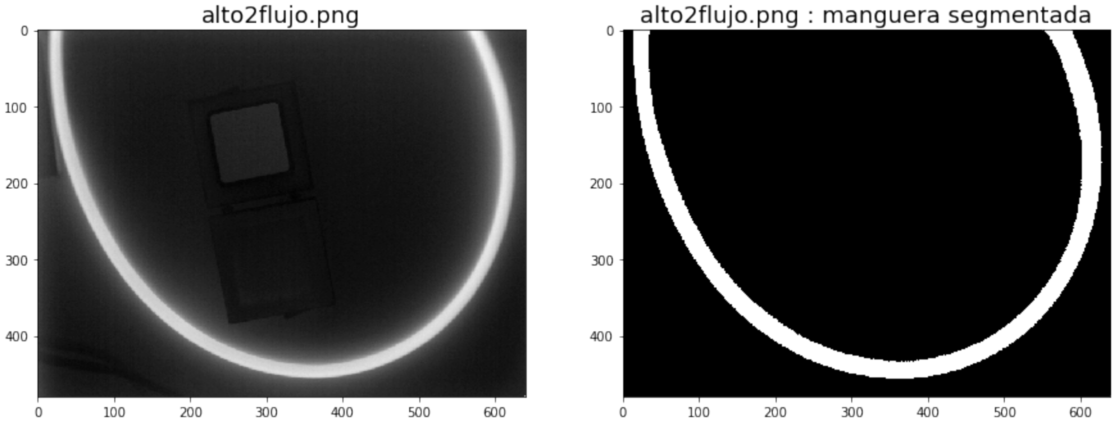
[13]:
```
for nombre in mangueras.keys():
    utils.side_by_side(
        mangueras[nombre], mangueras_segmentadas[nombre],
        title1=nombre, title2=f"{nombre} : manguera segmentada"
    )
```


altoflujo.png
altoflujo.png : manguera segmentada
0
100
200
300
400
0
100
200
300
400
500
600


bajo1flujo.png
bajo1flujo.png : manguera segmentada
0
100
200
300
400
0
100
200
300
400
500
600


bajo2flujo.png
bajo2flujo.png : manguera segmentada
0
100
200
300
400
0
100
200
300
400
500
600





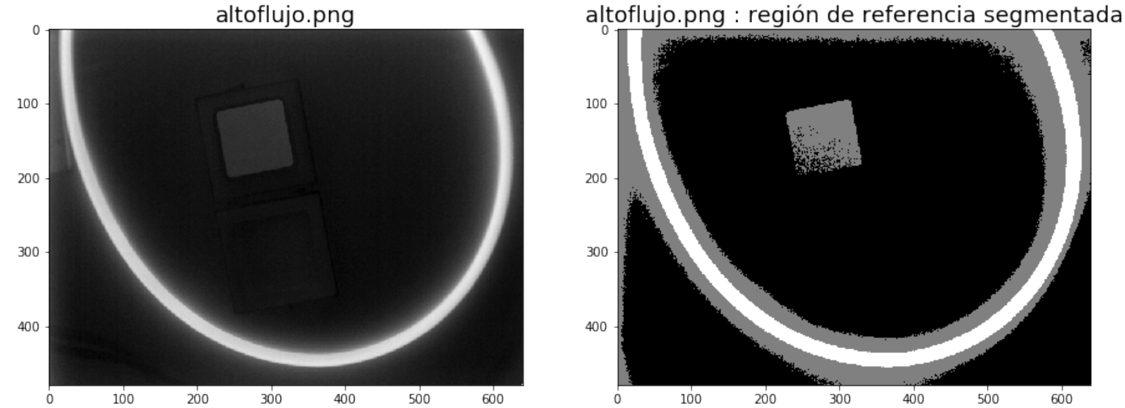
Aquí podemos observar las imágenes con su respectiva máscara de segmentación.

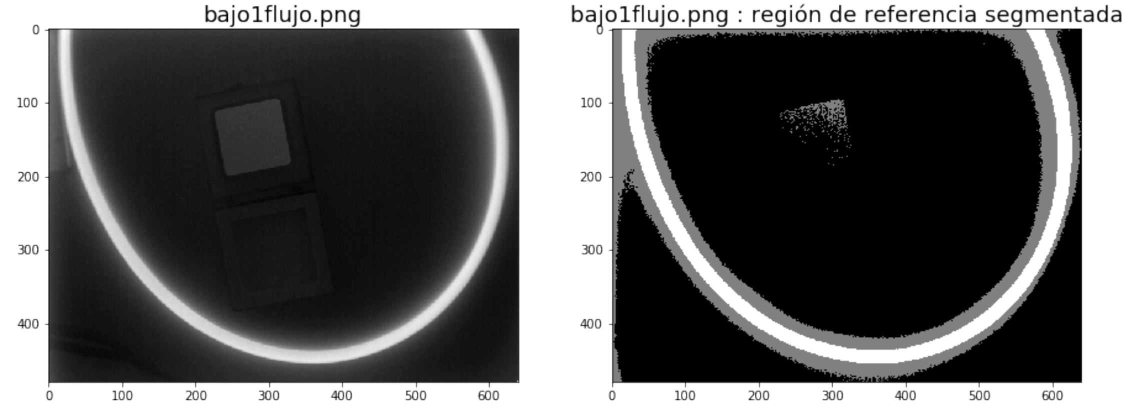
[14]:

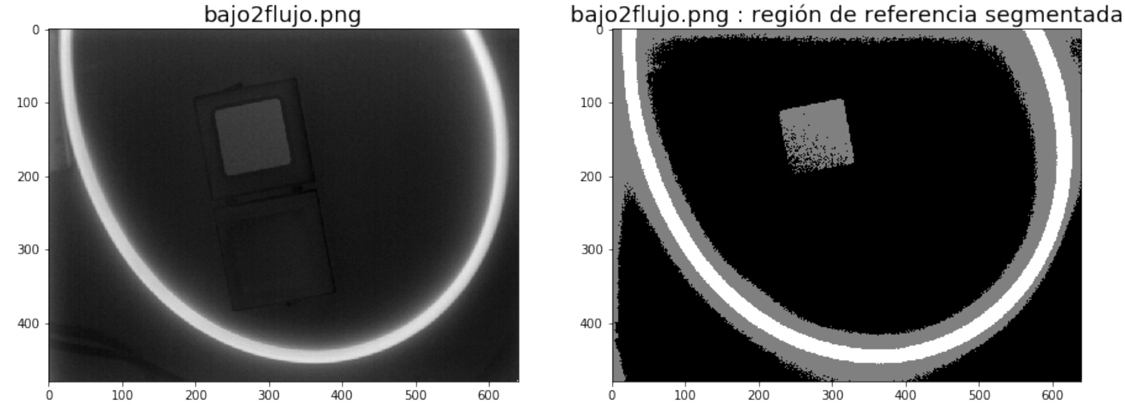
```python
region_ref1 = {
    key: auto_segment(mangueras[key], groups=3) for key in mangueras.keys()
}
```

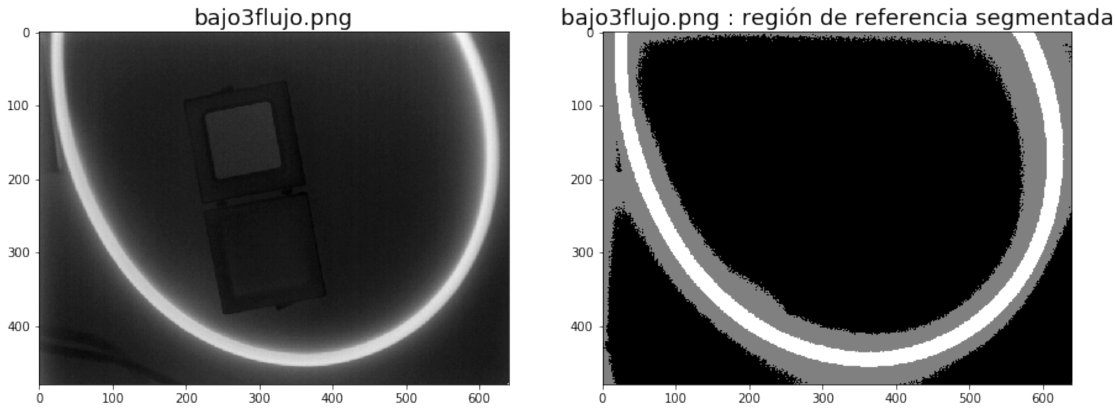
[15]:

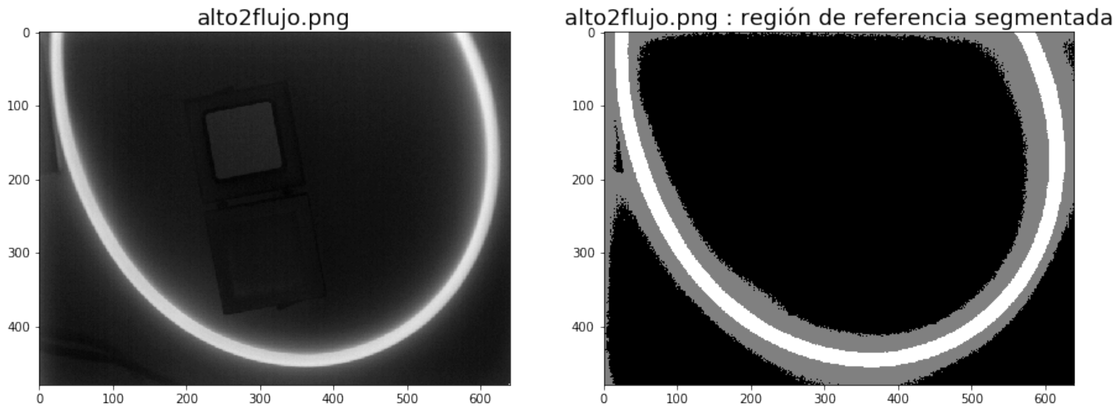
```python
for nombre in mangueras.keys():
    utils.side_by_side(
        mangueras[nombre], region_ref1[nombre],
        title1=nombre, title2=f"{nombre} : región de referencia segmentada"
    )
```


altoflujo.png
altoflujo.png : región de referencia segmentada
0
100
200
300
400
0
100
200
300
400
500
600


bajo1flujo.png
bajo1flujo.png : región de referencia segmentada
0
100
200
300
400
0
100
200
300
400
500
600


bajo2flujo.png
bajo2flujo.png : región de referencia segmentada
0
100
200
300
400
0
100
200
300
400
500
600





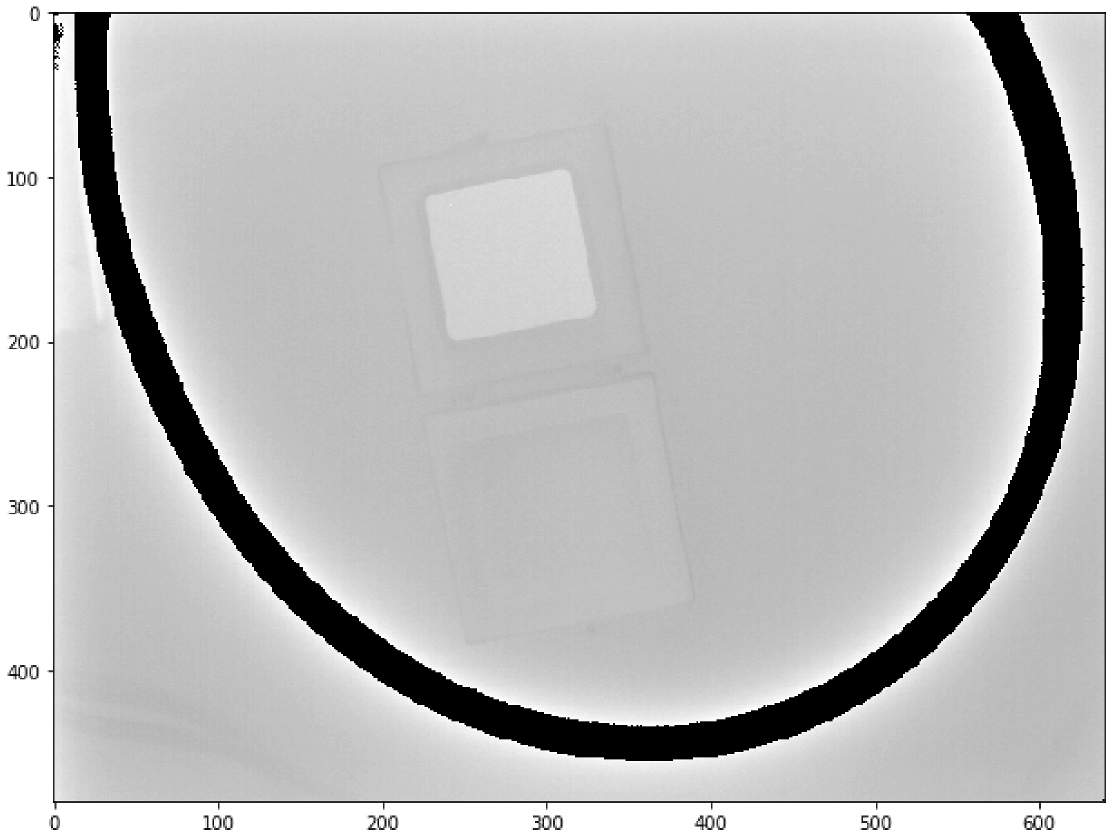
Aquí podemos observar que la referencia es más difícil de segmentar en función de las intensidades.

La función fue llamada indicando que se buscaba una imagen trinaria `auto_seg(img, groups=3)` Se esperaba que esto permitiese segmentar la **región referencia** ya que ésta muestra una intensidad mayor a la del fondo pero menor a la de la manguera.

Tal vez quitando la región de la manguera (la de mayor intensidad) sea más fácil segmentar automáticamente la **región referencia**.

[16]:
```
sin_manguera = {
    key: mangueras[key] * np.uint8(1.0 - mangueras_segmentadas[key])
    for key in mangueras_segmentadas.keys()
}
plt.imshow(sin_manguera[llaves[0]], cmap='gray')
```

[16]:
```
<matplotlib.image.AxesImage at 0x1c28376f10>
```

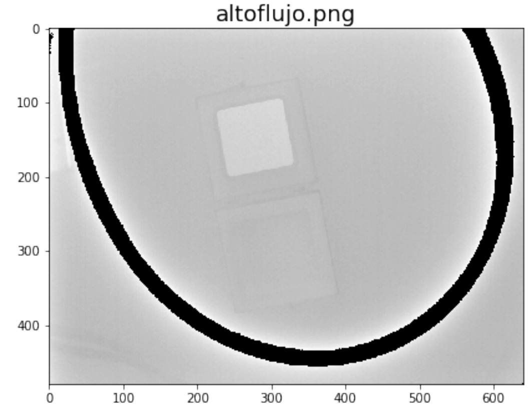


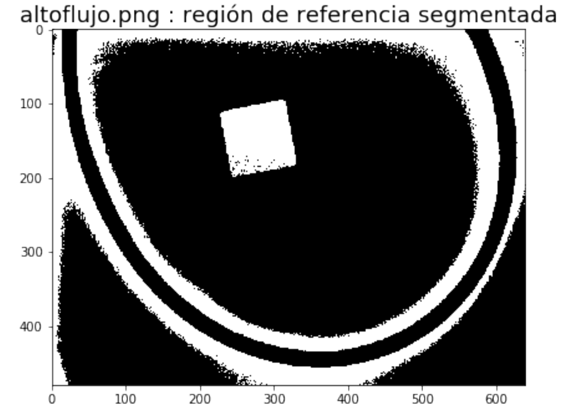
Nótese que la imagen muestra en negro la región que antes mostraba la mayor intensidad.

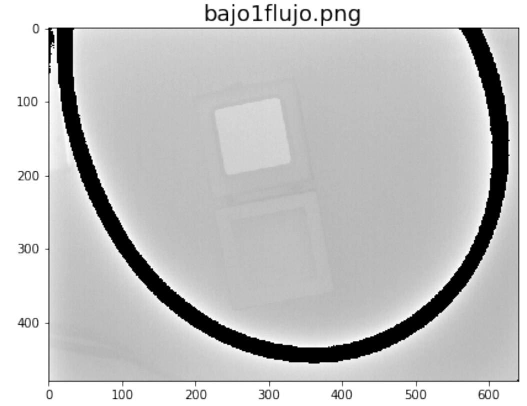
[17]:
```
sin_manguera = {
    key: mangueras[key] * np.uint8(1.0 - mangueras_segmentadas[key])
    for key in mangueras_segmentadas.keys()
}
```

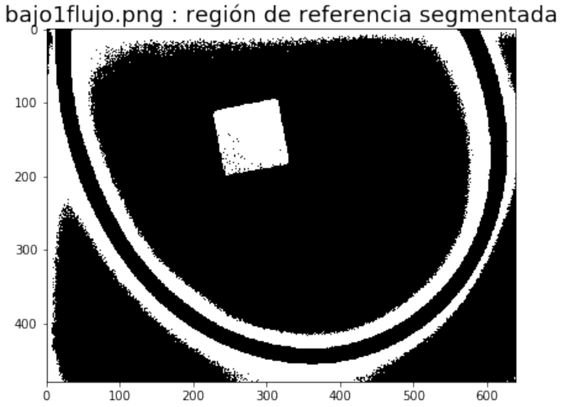
[18]:
```
region_ref2 = {
    key: auto_segment(sin_manguera[key], groups=2, nonzero=True) for key in␣
 ↪sin_manguera.keys()
}
```

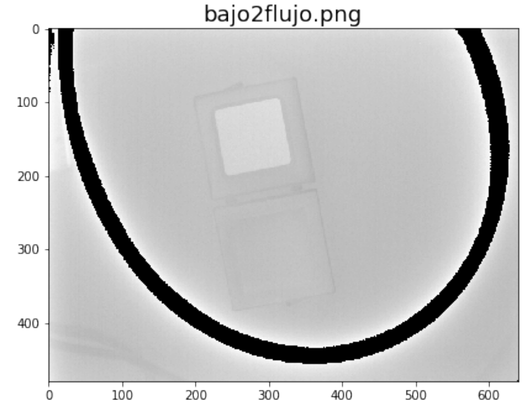
[19]:
```
for nombre in sin_manguera.keys():
    utils.side_by_side(
        sin_manguera[nombre], region_ref2[nombre],
        title1=nombre, title2=f"{nombre} : región de referencia segmentada"
    )
```

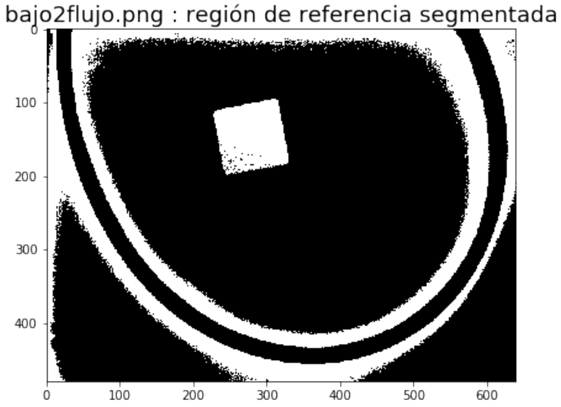

altoflujo.png
0
100
200
300
400
0
100
200
300
400
500
600

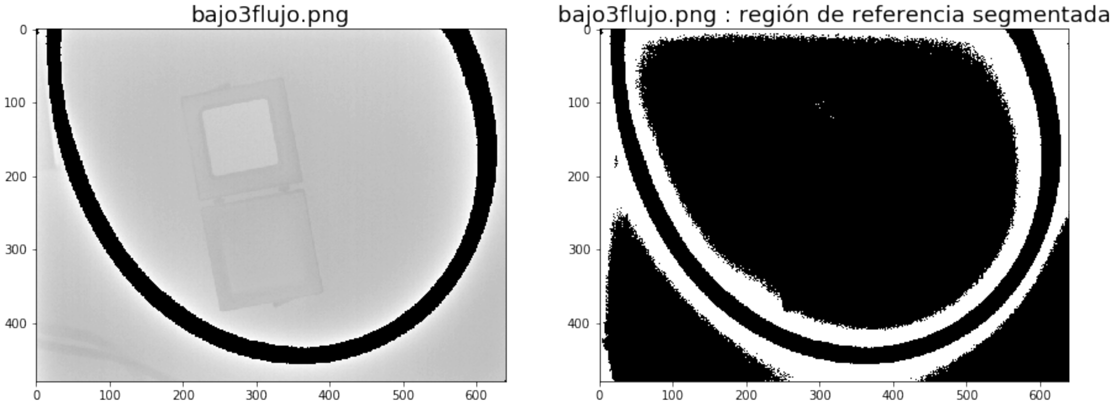

altoflujo.png : región de referencia segmentada
0
100
200
300
400
0
100
200
300
400
500
600

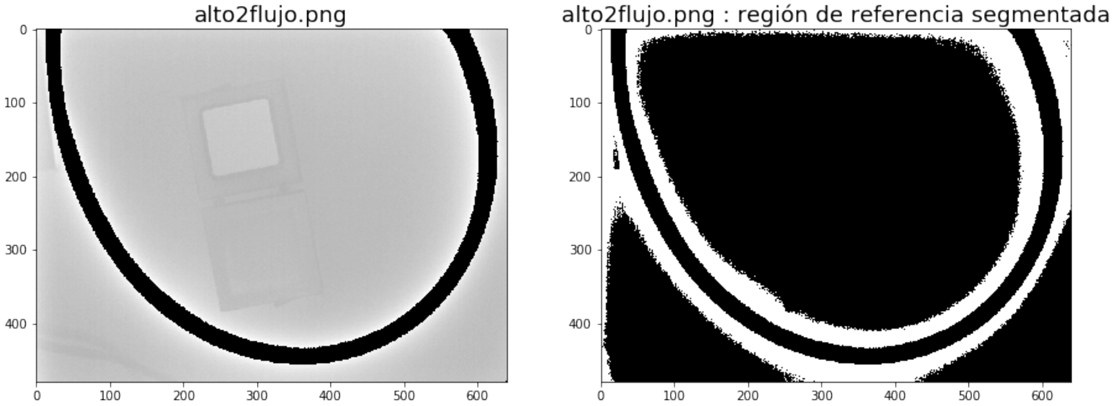

bajo1flujo.png
0
100
200
300
400
0
100
200
300
400
500
600


bajo1flujo.png : región de referencia segmentada
0
100
200
300
400
0
100
200
300
400
500
600


bajo2flujo.png
0
100
200
300
400
0
100
200
300
400
500
600


bajo2flujo.png : región de referencia segmentada
0
100
200
300
400
0
100
200
300
400
500
600





Aún teniendo la región de la manguera oscurecida, la función `auto_seg()` no permite segmentar la **región referencia** de forma automática. Esto podría atribuirse a que la forma del histograma de las ***imágenes con la manguera oscurecida*** sigue mostrando dos cúmulos principales como se muestra a continuación.

Sin embargo, debe notarse que la funcción `auto_seg(.., nonzero=True)` fue llamada con el parámetro `nonzero=True`, lo que hace que la funcón ignore las entradas que valen 0 al momento de calcular los centros de los grupos.

Si se desea una visualización más detallada del funcionamiento de este parámetro, se recomienda correr este código, en dos celdas por separado para observar el efecto del parámetro `nonzero` :

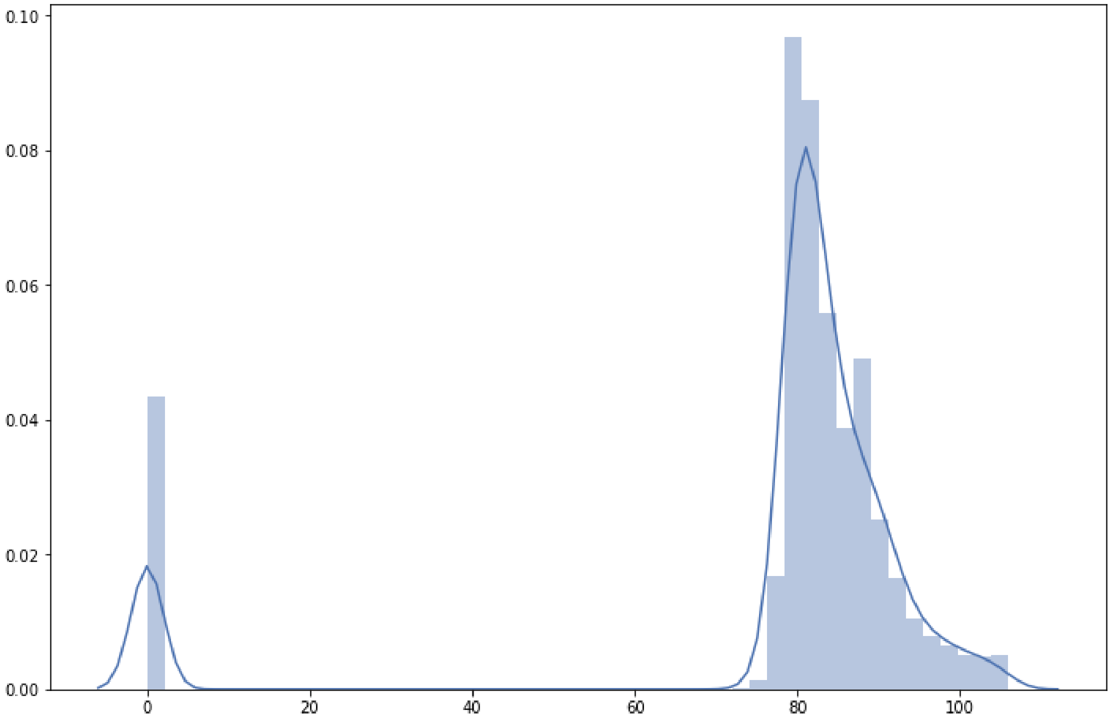
```
region_ref2 = {
    key: auto_segment(sin_manguera[key], groups=2, nonzero=True, verbose=True) for key in sin_r
}
```

por

```
region_ref2 = {
    key: auto_segment(sin_manguera[key], groups=2, nonzero=False, verbose=True) for key in sin_
}
```

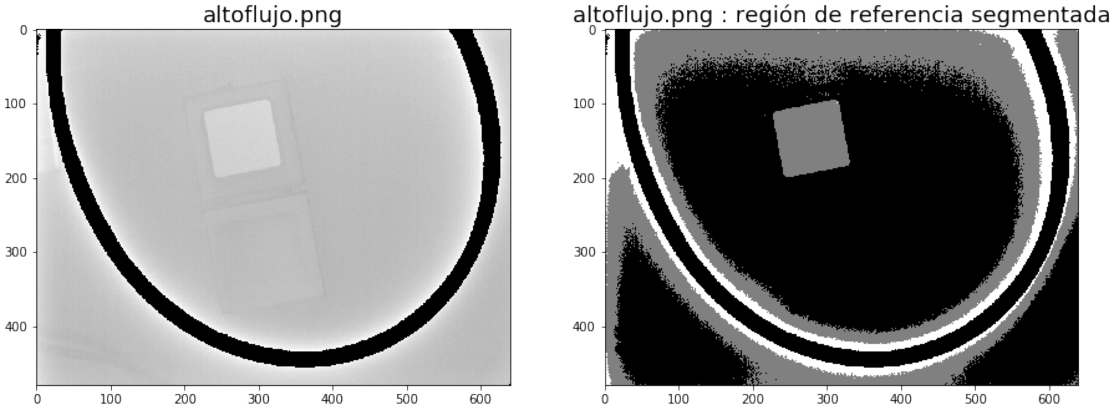
[20]:
```
sns.distplot(sin_manguera[llaves[2]].flatten())
```

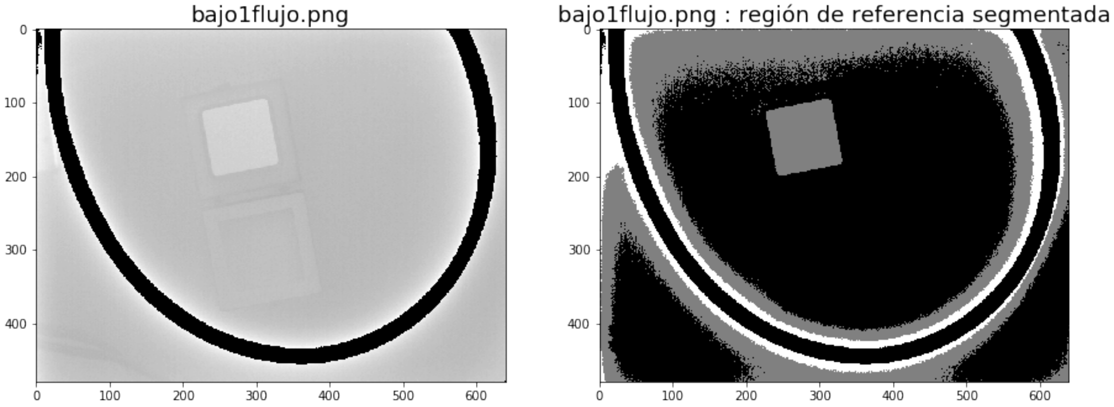
[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c2457f990>
```

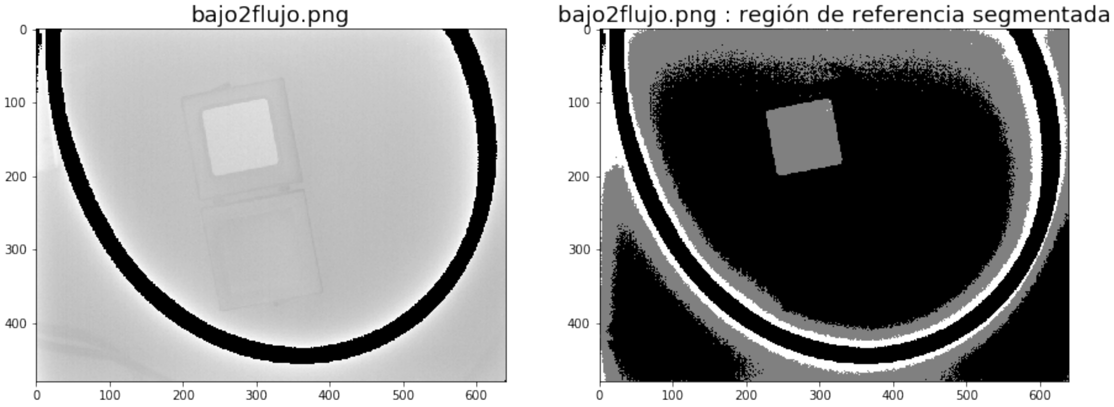


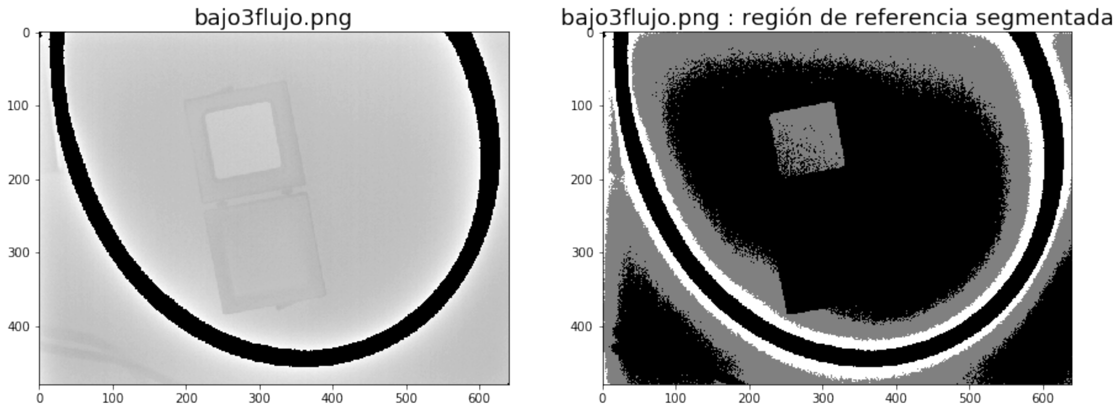
[21]:
```
region_ref3 = {
    key: auto_segment(sin_manguera[key], groups=3, nonzero=True) for key in␣
 ↪sin_manguera.keys()
}
```

[22]:
```
for nombre in sin_manguera.keys():
    utils.side_by_side(
        sin_manguera[nombre], region_ref3[nombre],
        title1=nombre, title2=f"{nombre} : región de referencia segmentada"
    )
```

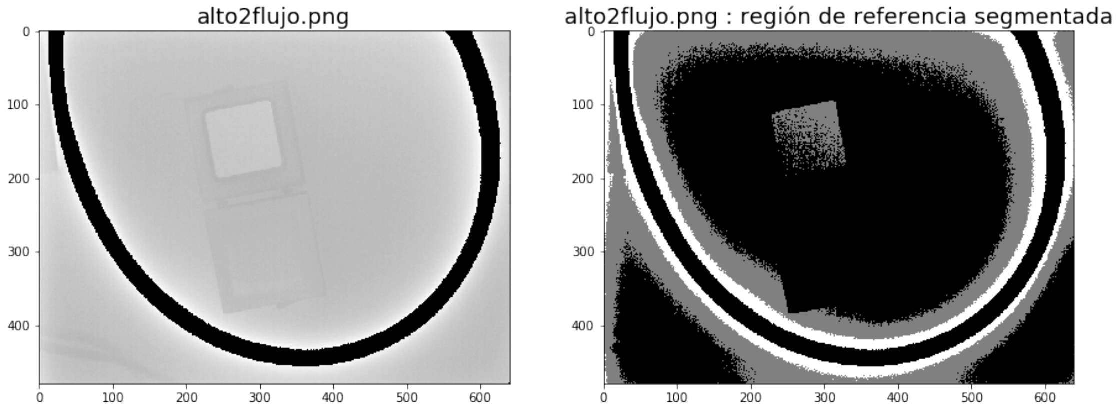

altoflujo.png
altoflujo.png : región de referencia segmentada
0
100
200
300
400
0
100
200
300
400
500
600


bajo1flujo.png
bajo1flujo.png : región de referencia segmentada
0
100
200
300
400
0
100
200
300
400
500
600


bajo2flujo.png
bajo2flujo.png : región de referencia segmentada
0
100
200
300
400
0
100
200
300
400
500
600


bajo3flujo.png

bajo3flujo.png : región de referencia segmentada


alto2flujo.png

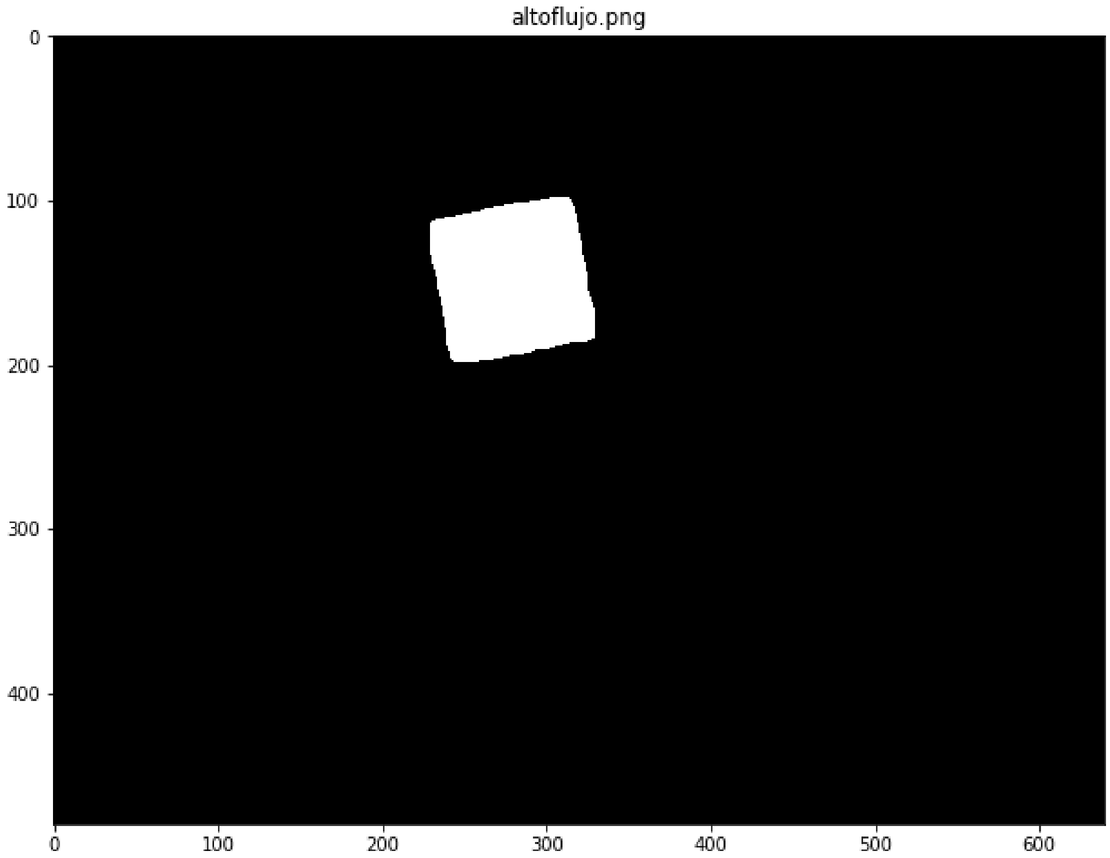
alto2flujo.png : región de referencia segmentada

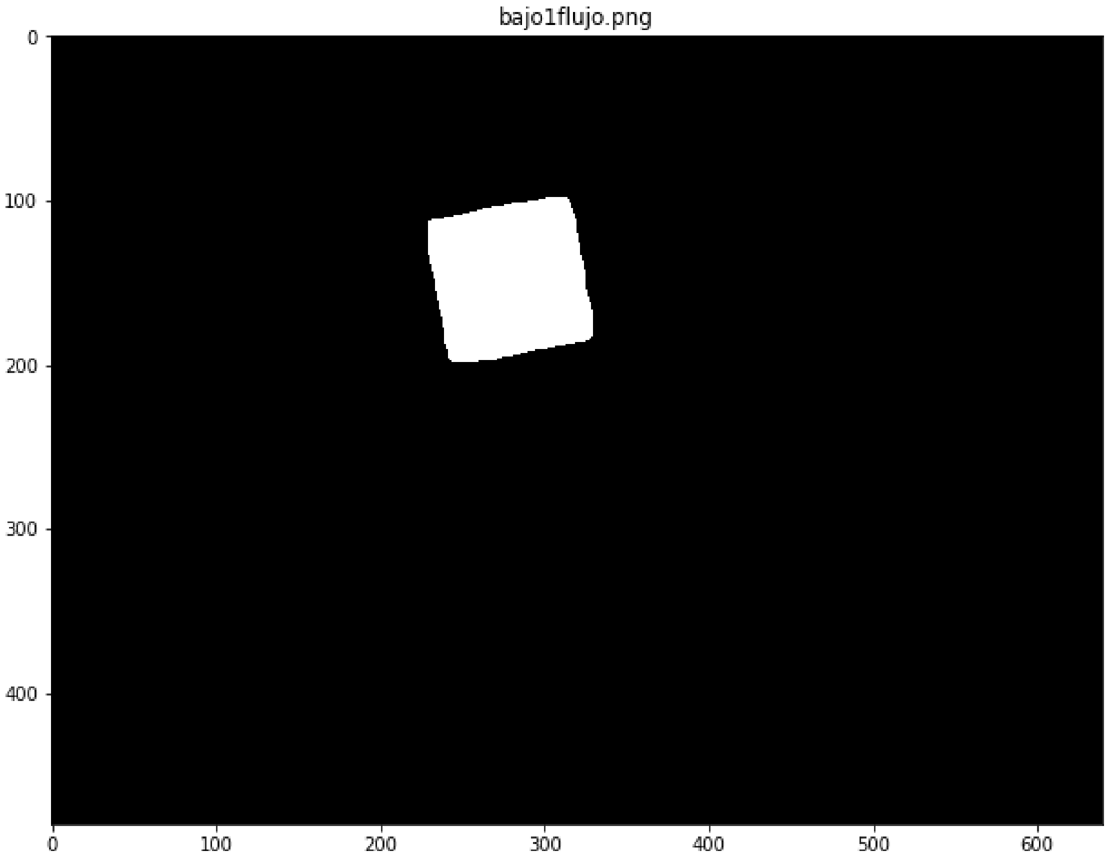
[23]:
```
edges = canny(mangueras[llaves[0]] /255.)
fill_coins = ndi.binary_fill_holes(edges)
```

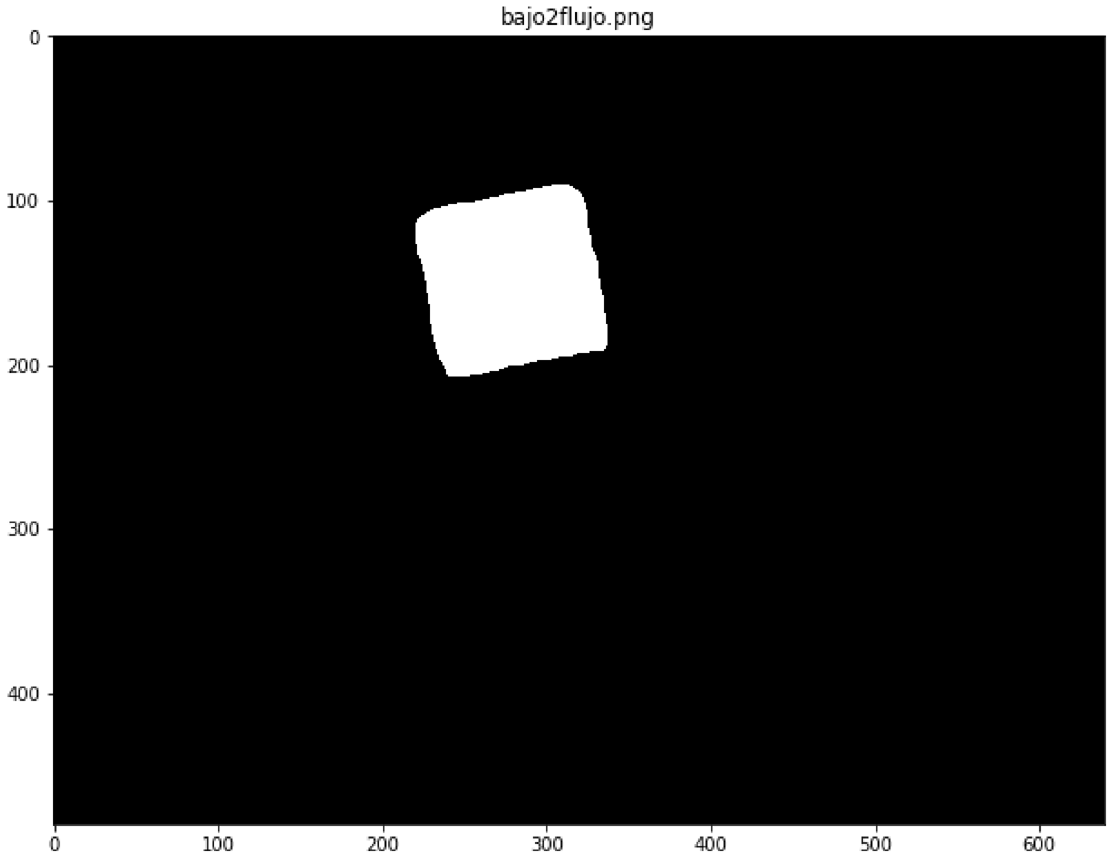
[24]:
```
verbose = False

if verbose:
    for img1 in mangueras.values():
        ref_region(img1, verbose=True)
```

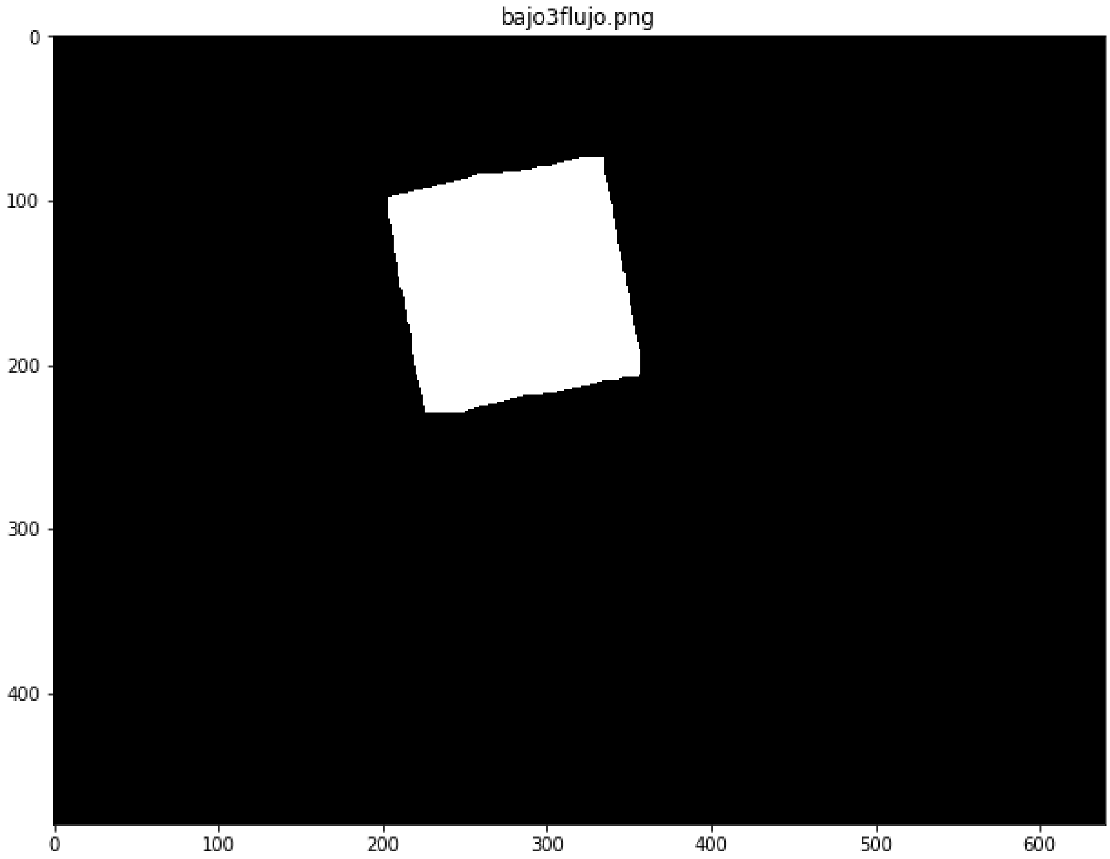
[25]:
```
region_ref4 = {
    key: ref_region(mangueras[key]) for key in mangueras.keys()
}
```

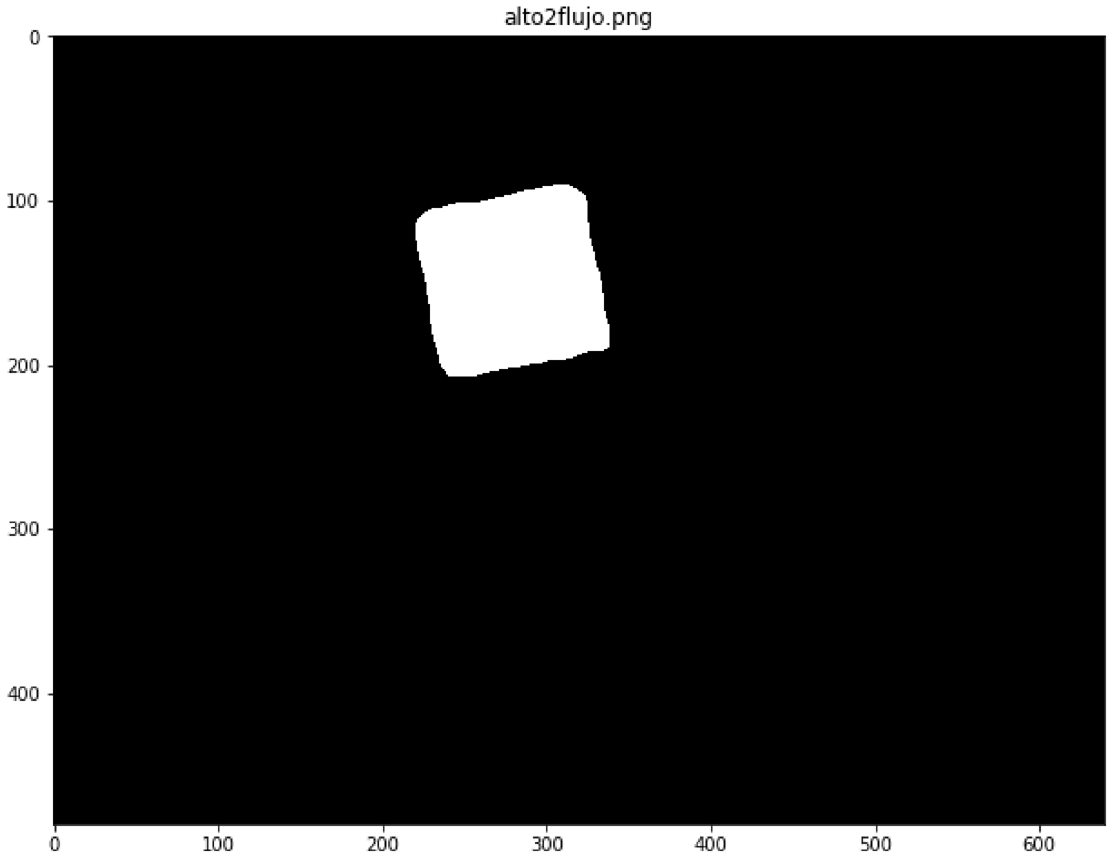
[26]:
```
for nombre, imagen in zip(region_ref4.keys(), region_ref4.values()):
    plt.figure()
    plt.imshow(np.uint8(imagen), cmap="gray")
    plt.title(nombre)
```


altoflujo.png
0
100
200
300
400
0
100
200
300
400
500
600


bajo1flujo.png
0
100
200
300
400
0
100
200
300
400
500
600


bajo2flujo.png
0
100
200
300
400
0
100
200
300
400
500
600

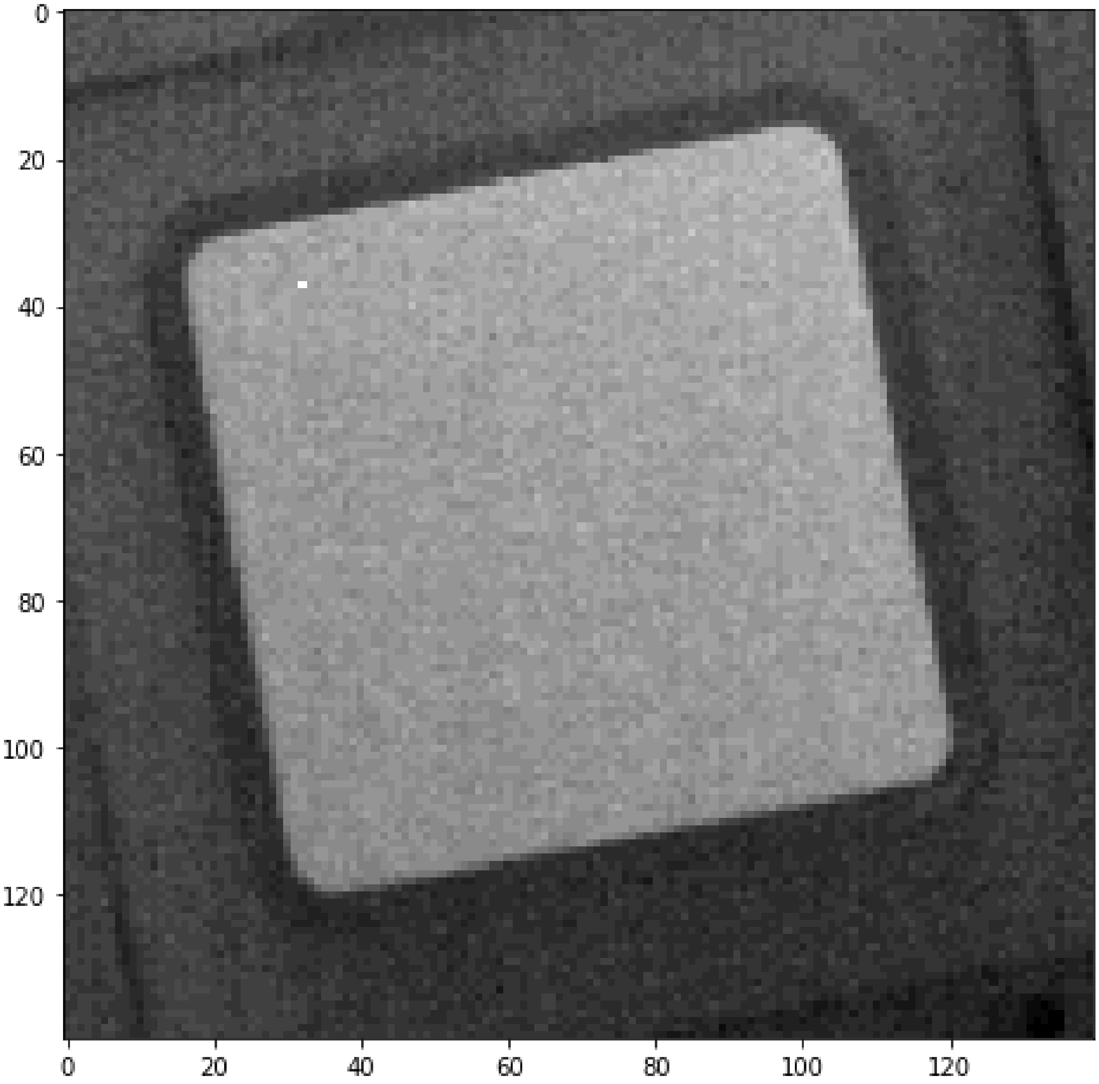

bajo3flujo.png
0
100
200
300
400
0
100
200
300
400
500
600

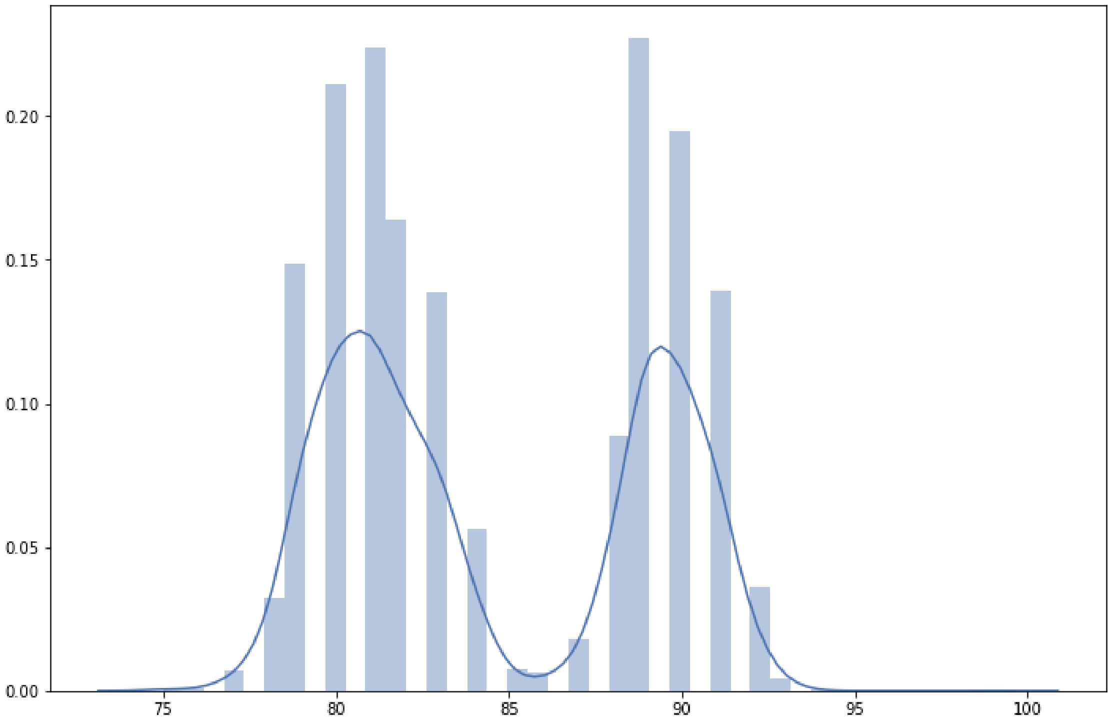


[27]:
```
segmented_ref_reg = {
    key: mangueras[key] * region_ref4[key] for key in llaves
}
```

[28]:
```
_tmp = copy.deepcopy(mangueras[llaves[0]][80:220, 210:350])
#_tmp[ _tmp < 85] = 0
#_tmp *= np.uint8( auto_segment(_tmp) * 255 )
plt.imshow(_tmp, cmap='gray')
plt.figure()
sns.distplot(_tmp.flatten()[_tmp.flatten().nonzero()])
```

[28]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c2843e810>
```


0
20
40
60
80
100
120
0
20
40
60
80
100
120

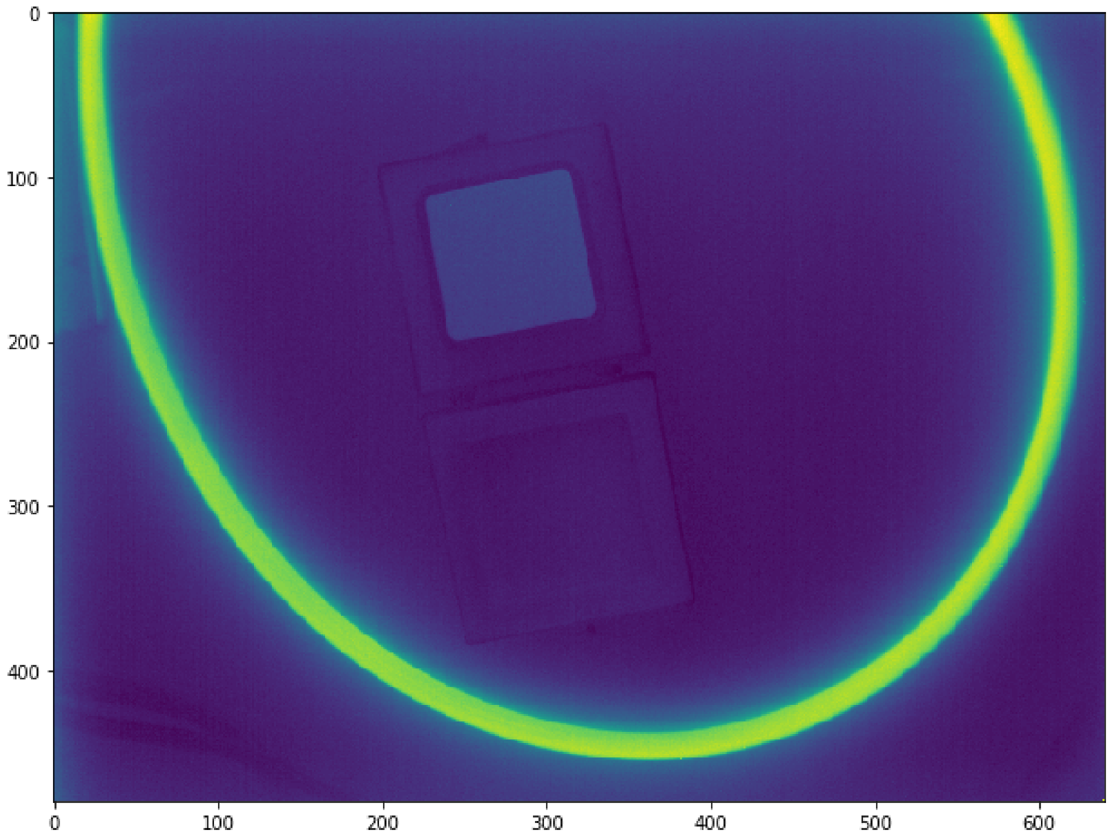


[29]:

```
plt.imshow(mangueras[llaves[0]])
```

[29]:

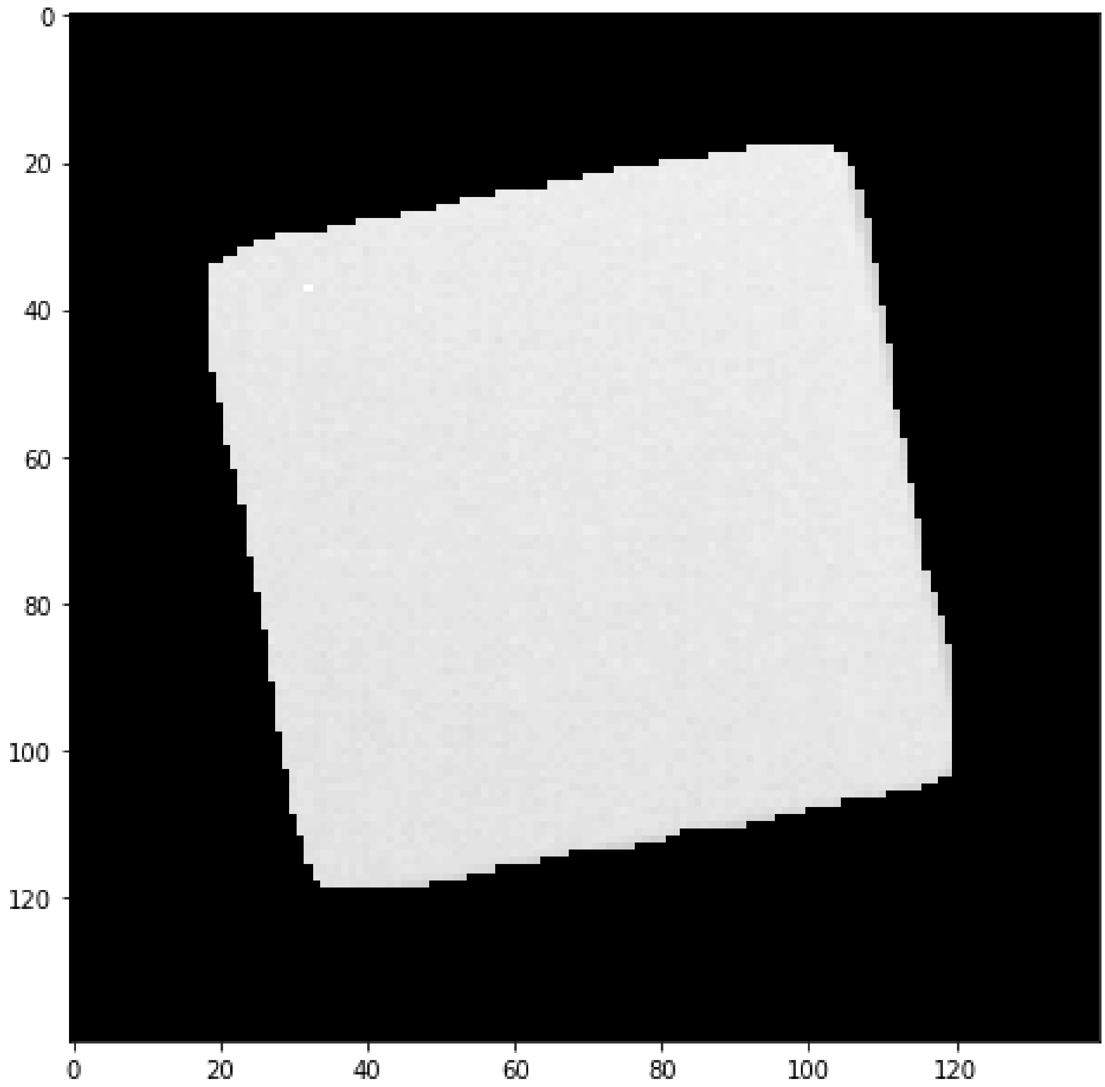
```
<matplotlib.image.AxesImage at 0x1c25f2c5d0>
```

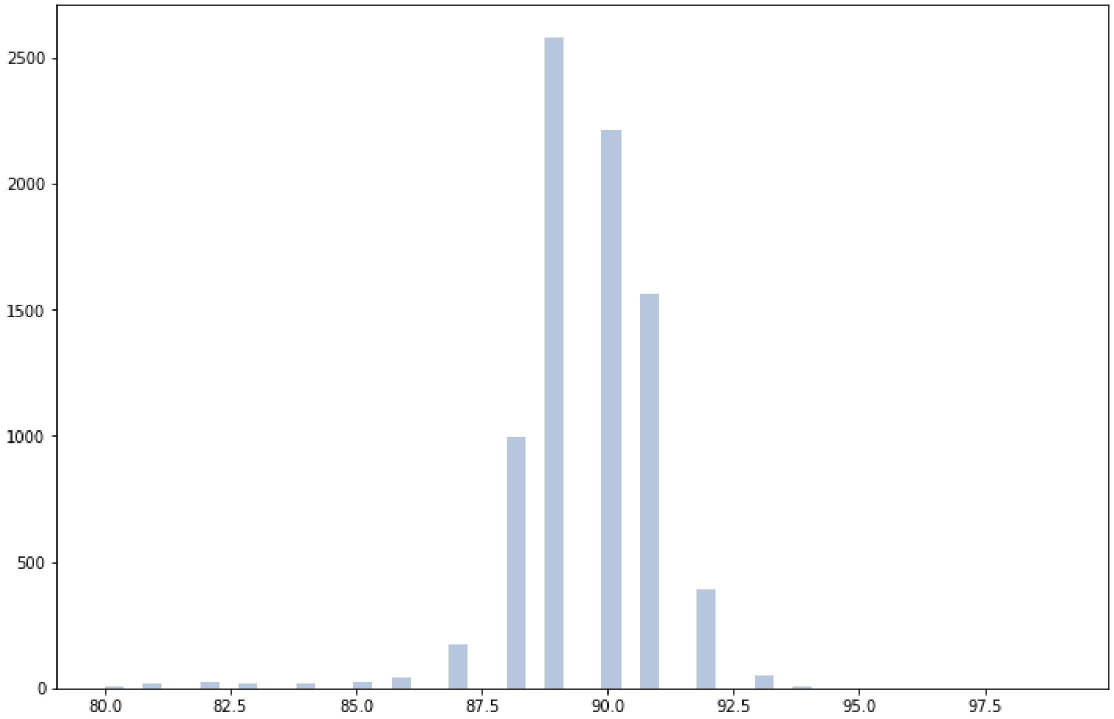


[30]:
```
#_tmp = mangueras[llaves[0]][90:210, 200:350]
#_tmp = auto_segment(_tmp)
#plt.imshow(_tmp, cmap='gray')
#plt.figure()
#sns.distplot(mangueras[llaves[0]][_tmp.nonzero()].flatten())
```

[31]:
```
_tmp = copy.deepcopy(segmented_ref_reg[llaves[0]][80:220, 210:350])
plt.imshow(_tmp, cmap='gray')
plt.figure()
sns.distplot(_tmp[ _tmp != 0].flatten(), kde=False)
```

[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c283ce310>
```


0
20
40
60
80
100
120
0
20
40
60
80
100
120



```
[32]: # Esto servía, pero ya no :
      """
      region_info = pd.core.frame.DataFrame({
          f"{key.replace('.png', '')} ": value[ value != 0 ].flatten() for key, value␣
       ↪in segmented_ref_reg.items()
      })
      region_info.describe()
      """
```

```
[32]: '\nregion_info = pd.core.frame.DataFrame({\n    f"{key.replace(\'.png\', \'\')}
      ": value[ value != 0 ].flatten() for key, value in segmented_ref_reg.items()
      \n})\nregion_info.describe()\n'
```

```
[137]: region_info_list = list(map(
           lambda x, y: pd.core.series.Series(x[ x != 0].flatten(), name=y),␣
        ↪segmented_ref_reg.values(), segmented_ref_reg.keys()
       ))
       region_info = pd.concat(region_info_list, axis=1)
```

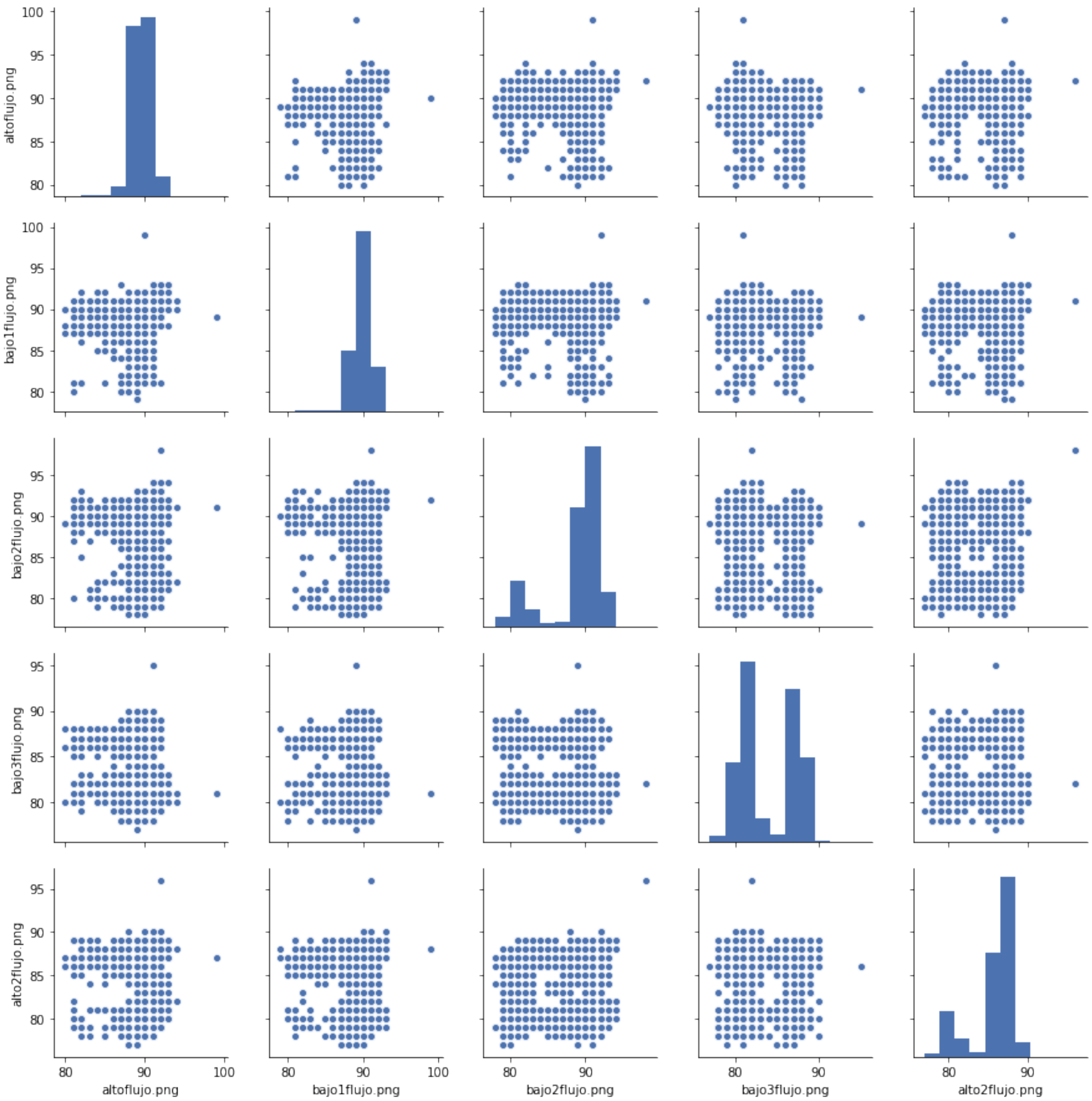
```
[138]: region_info.describe()
```

```
[138]:        altoflujo.png  bajo1flujo.png  bajo2flujo.png  bajo3flujo.png  \
       count    8111.000000     8144.000000    10736.000000    18376.000000
       mean       89.573172       89.296537       87.485190       82.928766
```

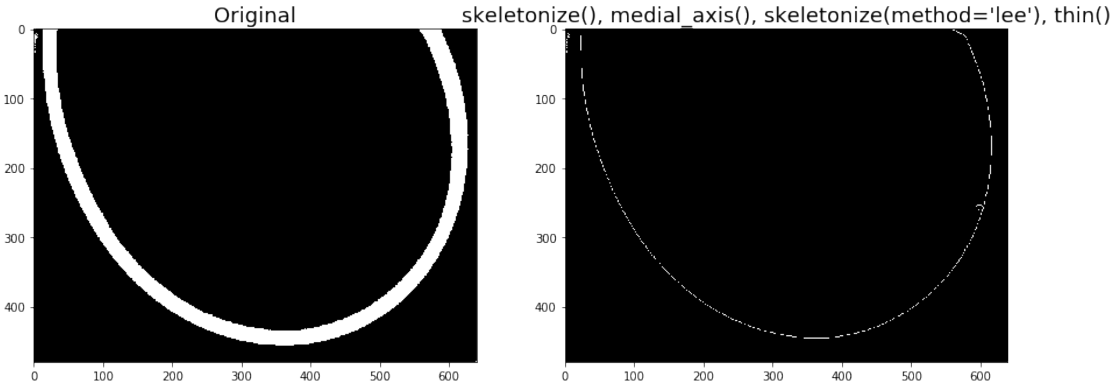
```
std         1.410561       1.432713       4.179704       3.440137
min        80.000000      79.000000      77.000000      76.000000
25%        89.000000      89.000000      87.000000      80.000000
50%        90.000000      89.000000      89.000000      82.000000
75%        90.000000      90.000000      90.000000      86.000000
max        99.000000      99.000000      98.000000      95.000000

       alto2flujo.png
count    10839.000000
mean        84.963927
std          3.245888
min         76.000000
25%         84.000000
50%         86.000000
75%         87.000000
max         96.000000
```

```
[35]: # Relatively slow, avoid running :
      sns.pairplot(region_info.dropna())
```

```
[35]: <seaborn.axisgrid.PairGrid at 0x1c25357e10>
```



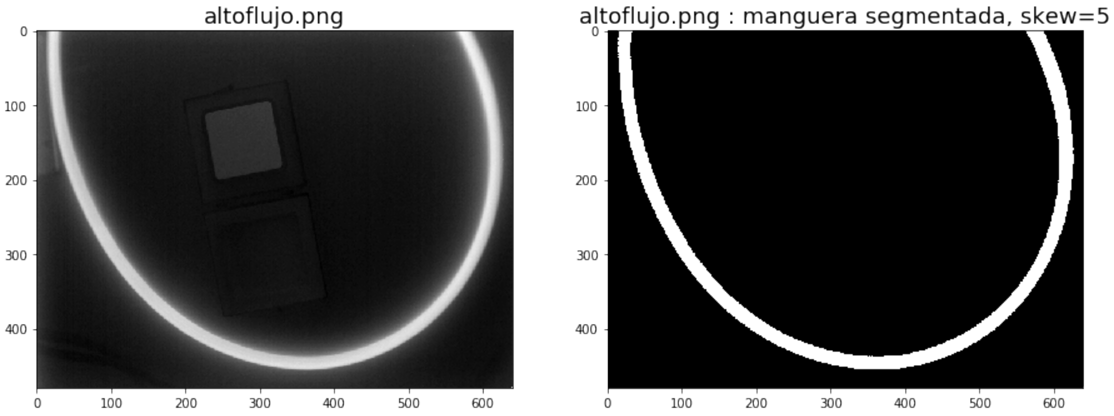
```
[181]: #find_branch_points(_hola).sum()
       _tmp = mangueras_segmentadas[llaves[0]]
       sk, ma, skl, th = skeletonize(_tmp), medial_axis(_tmp), skeletonize(_tmp,␣
        ↪method='lee'), thin(_tmp)
       la_buena = reduce(cv.bitwise_xor, lmap(np.uint8, [sk, skl, ma, th]))
       utils.side_by_side(_tmp, la_buena, title1="Original", title2="skeletonize(),␣
        ↪medial_axis(), skeletonize(method='lee'), thin()")
```


Original

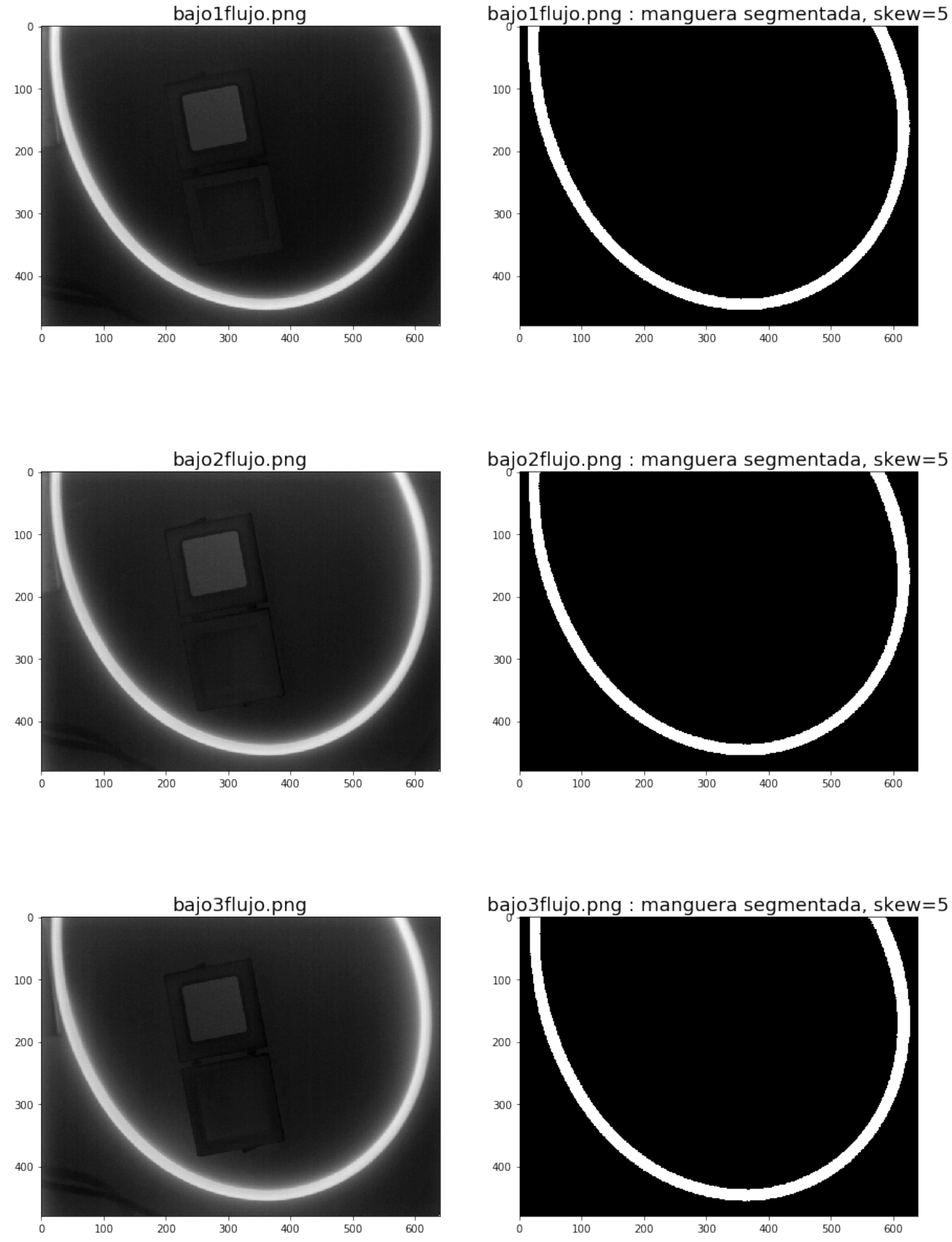
skeletonize(), medial_axis(), skeletonize(method='lee'), thin()

```
[178]: mangueras_segmentadas_amano = {
           key: auto_segment(mangueras[key], verbose=False, groups=2, skew=10) for key␣
        ↪in mangueras.keys()
       }
```

```
[179]: for nombre in mangueras_segmentadas_amano.keys():
           utils.side_by_side(
               mangueras[nombre], mangueras_segmentadas_amano[nombre],
               title1=nombre, title2=f"{nombre} : manguera segmentada, skew={5}"
           )
```


altoflujo.png

altoflujo.png : manguera segmentada, skew=5



bajo1flujo.png

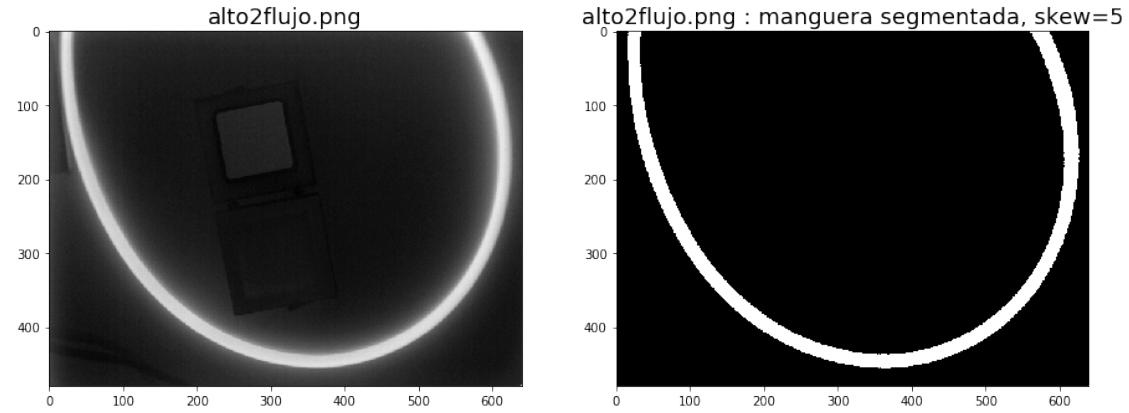
bajo1flujo.png : manguera segmentada, skew=5

bajo2flujo.png

bajo2flujo.png : manguera segmentada, skew=5

bajo3flujo.png

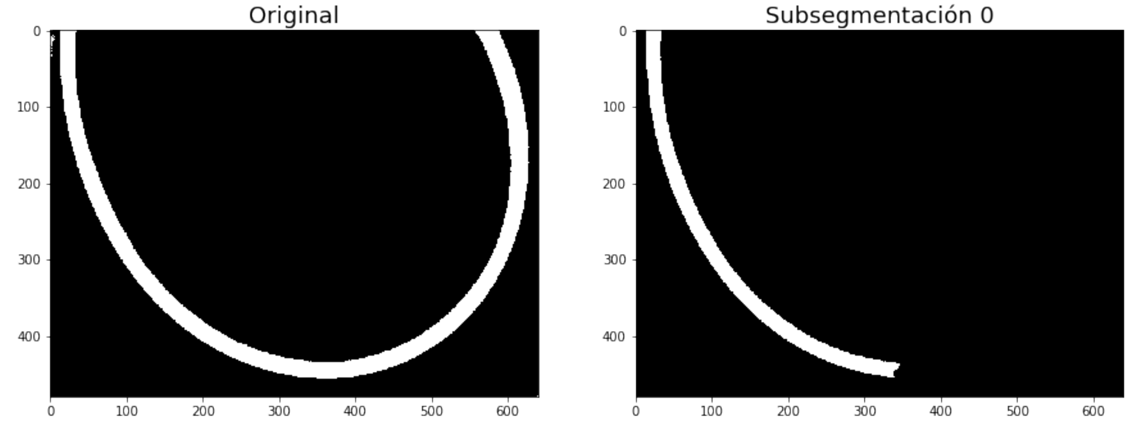
bajo3flujo.png : manguera segmentada, skew=5

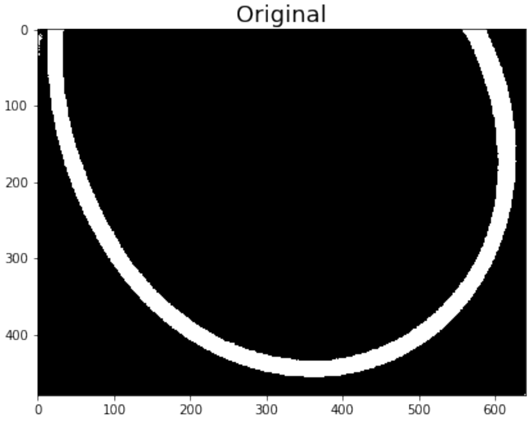


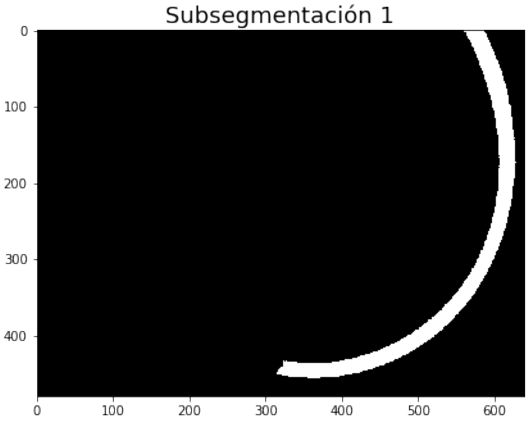
```
[69]: mascaras_agregadas_2 = {
          key: reduce(cv.bitwise_xor, subdivide_hose(mangueras_segmentadas_amano[key],␣
       ↪2, contiguous=True)) for key in llaves
      }
```

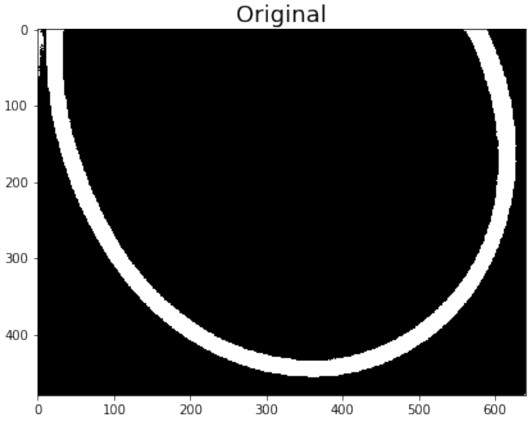
```
[70]: mascaras_subdivididas_2 = {
          key: subdivide_hose(mangueras_segmentadas_amano[key], 2, contiguous=True)␣
       ↪for key in llaves
      }
```

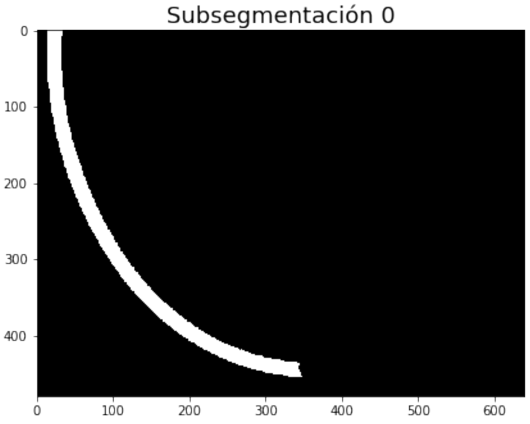
```
[182]: for nom, masc in mascaras_subdivididas_2.items():
           for i, mas in enumerate(masc):
               utils.side_by_side(mangueras_segmentadas[nom], mas, title1="Original",␣
        ↪title2=f"Subsegmentación {i}")
```

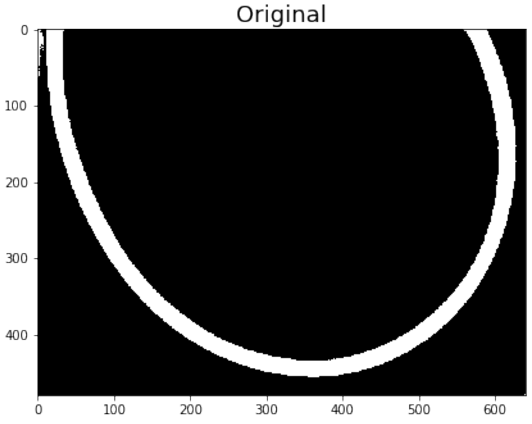


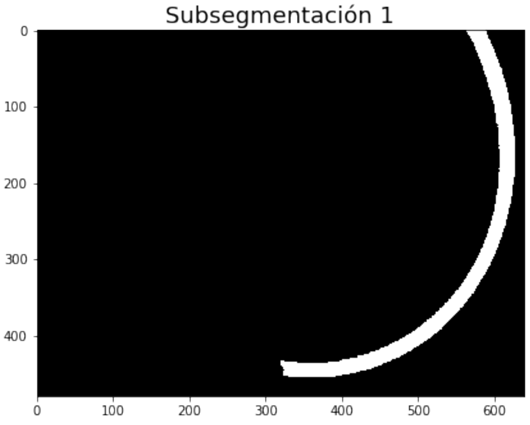

Original

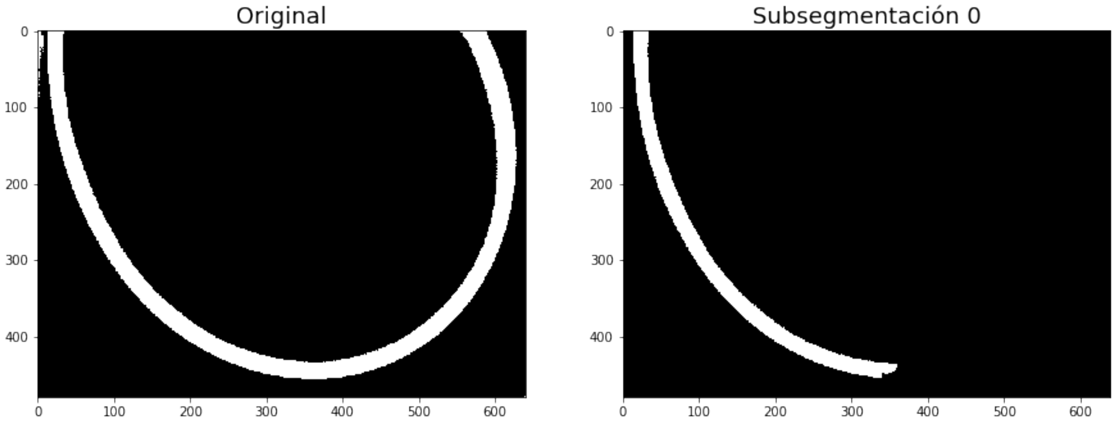

Subsegmentación 1

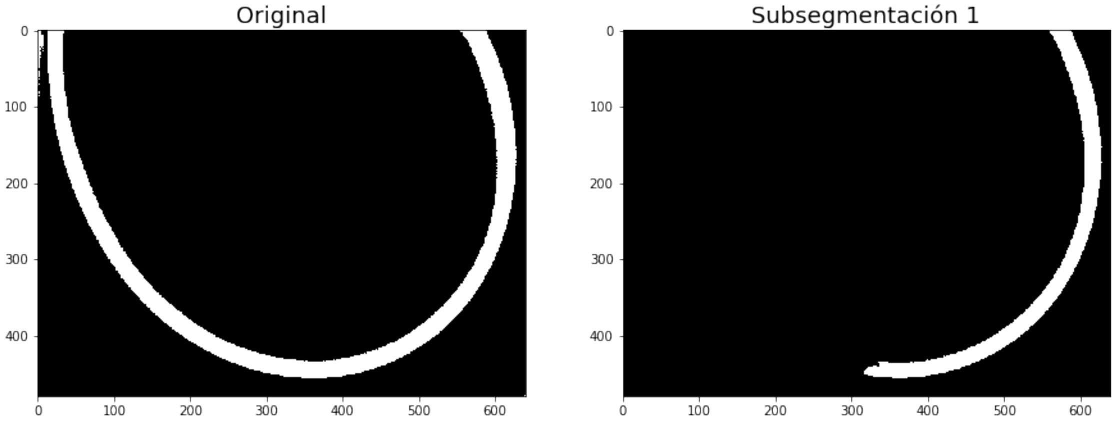

Original

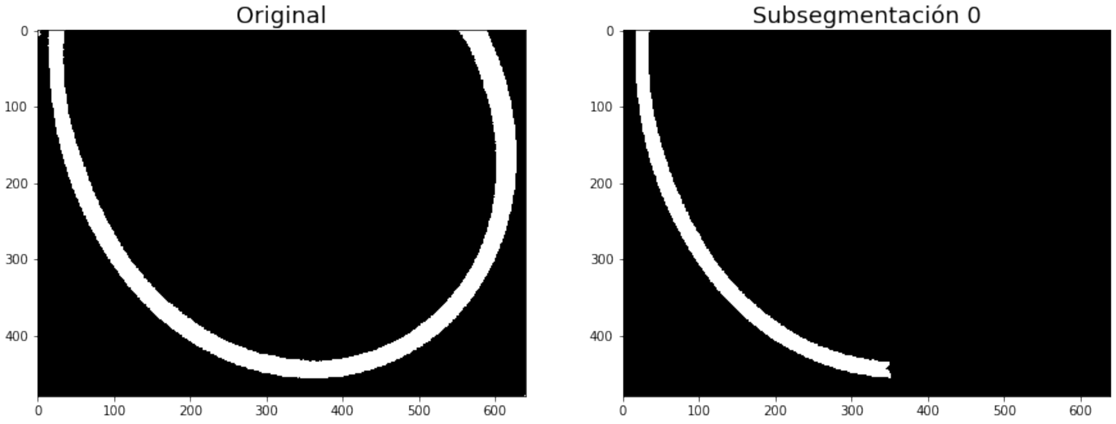

Subsegmentación 0

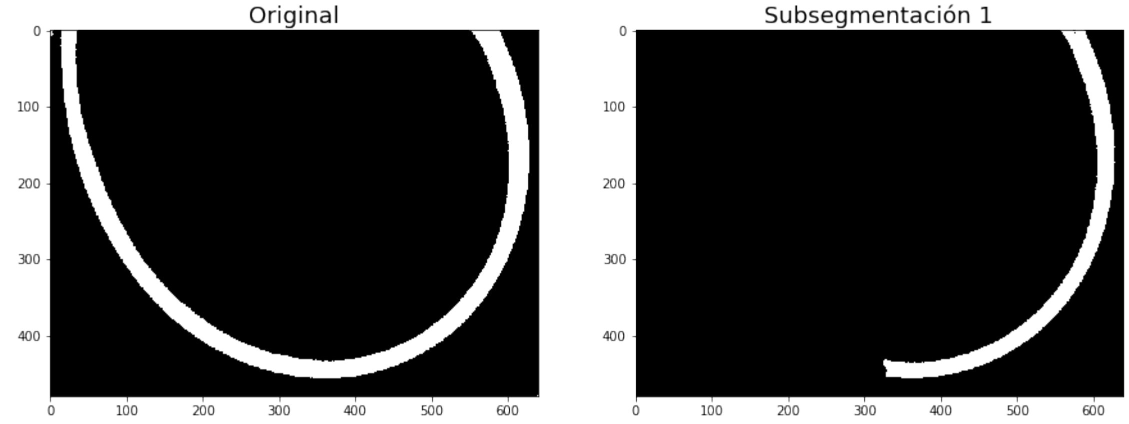

Original

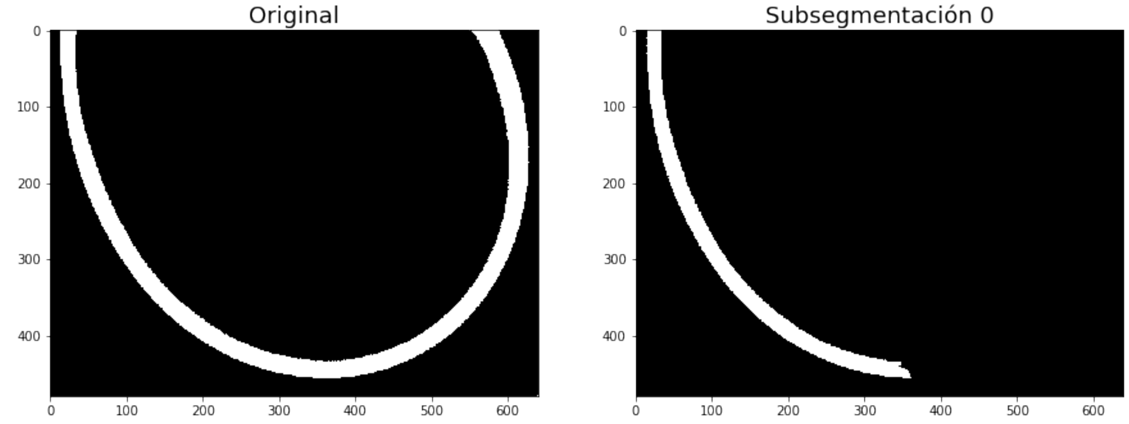

Subsegmentación 1

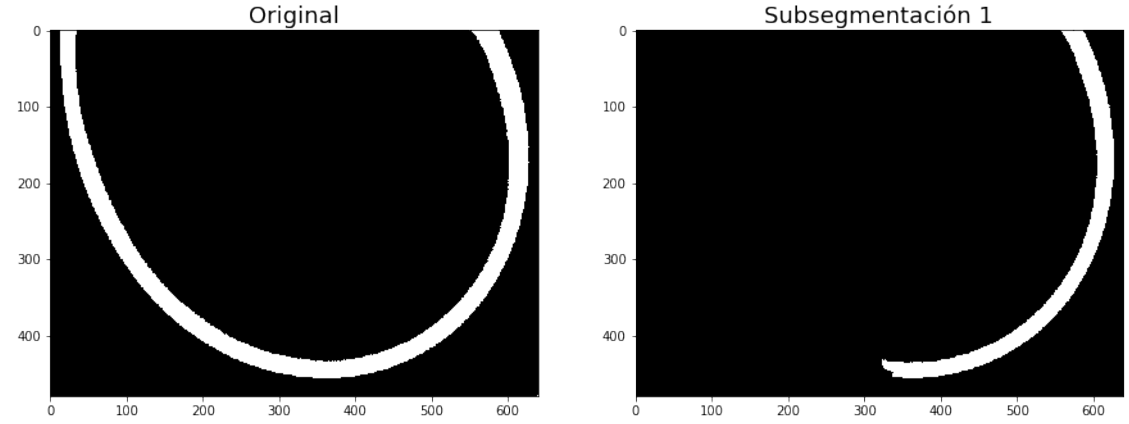

Original
Subsegmentación 0
0
100
200
300
400
0
100
200
300
400
500
600


Original
Subsegmentación 1
0
100
200
300
400
0
100
200
300
400
500
600


Original
Subsegmentación 0
0
100
200
300
400
0
100
200
300
400
500
600







[72]:
```
_plot = False
```

```
if _plot:
    for llave in llaves:
        plt.figure()
        plt.imshow(mascaras_agregadas_2[llave], cmap='gray')
```

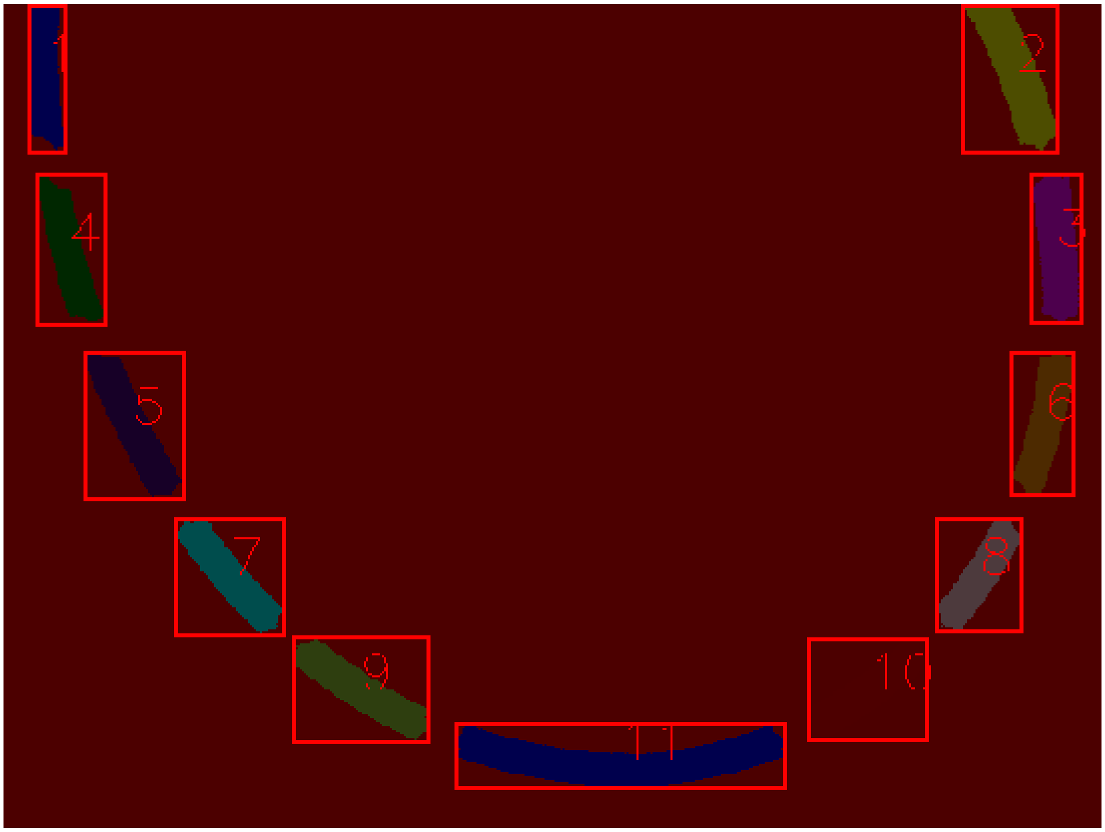
```
[73]: mascaras_agregadas_6 = {
          key: reduce(cv.bitwise_xor, subdivide_hose(mangueras_segmentadas_amano[key],␣
       ↪6, contiguous=False)) for key in llaves
      }
```

```
[74]: mascaras_subdivididas_6 = {
          key: subdivide_hose(mangueras_segmentadas_amano[key], 6, contiguous=False)␣
       ↪for key in llaves
      }
```

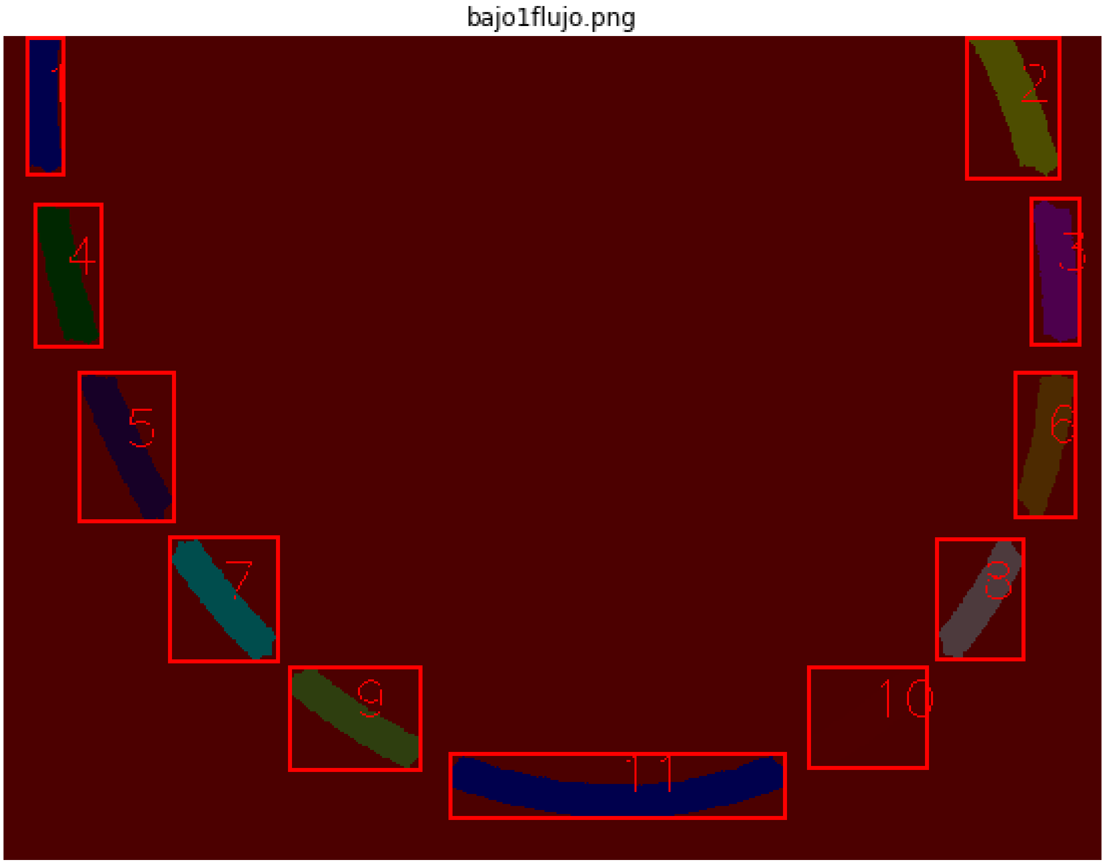
```
[187]: #plot_label_image_regions(mascaras_agregadas_2[llaves[3]])
       for llave in llaves:
           plot_label_image_regions(mascaras_agregadas_6[llave], title=llave)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

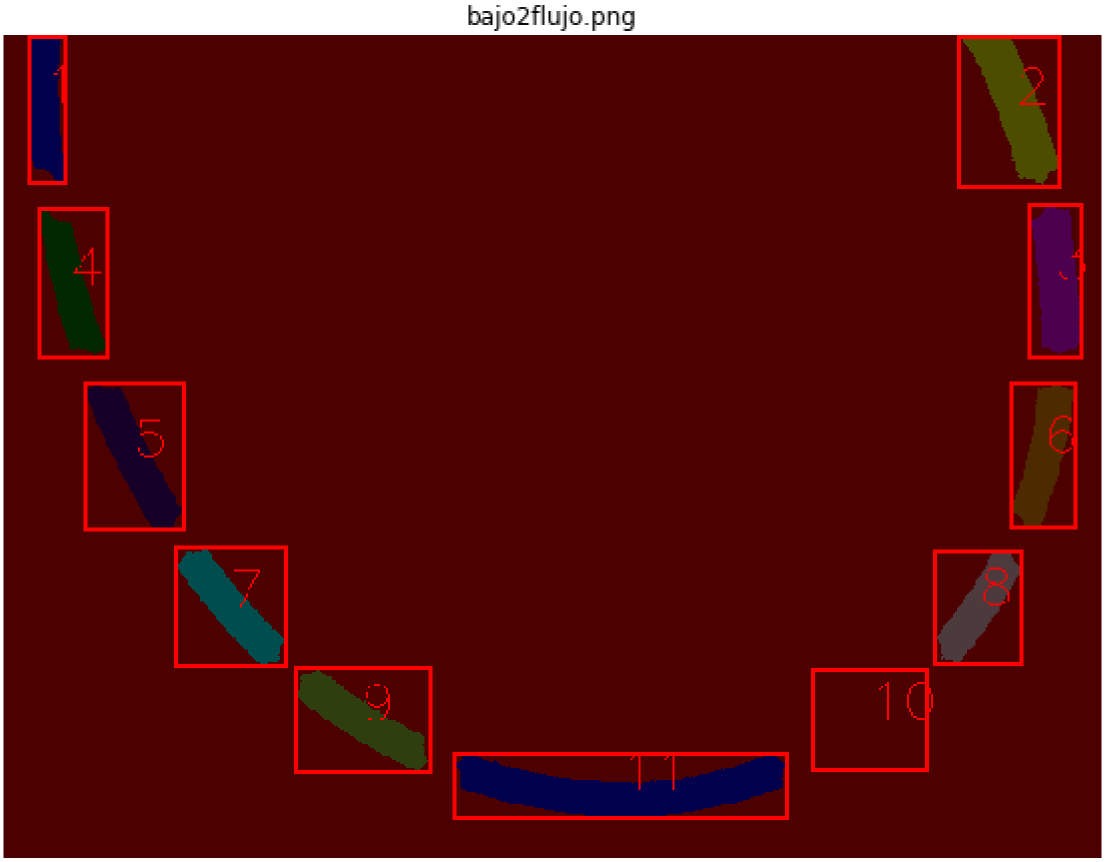
altoflujo.png



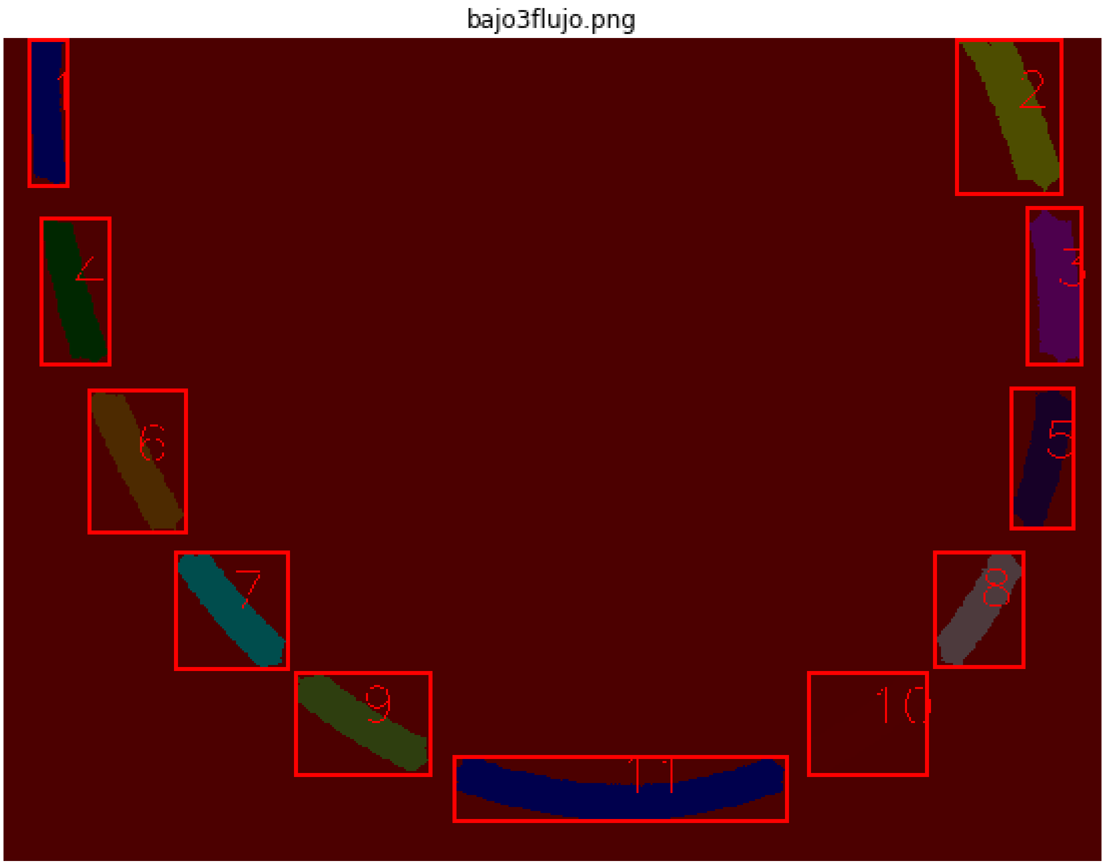
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



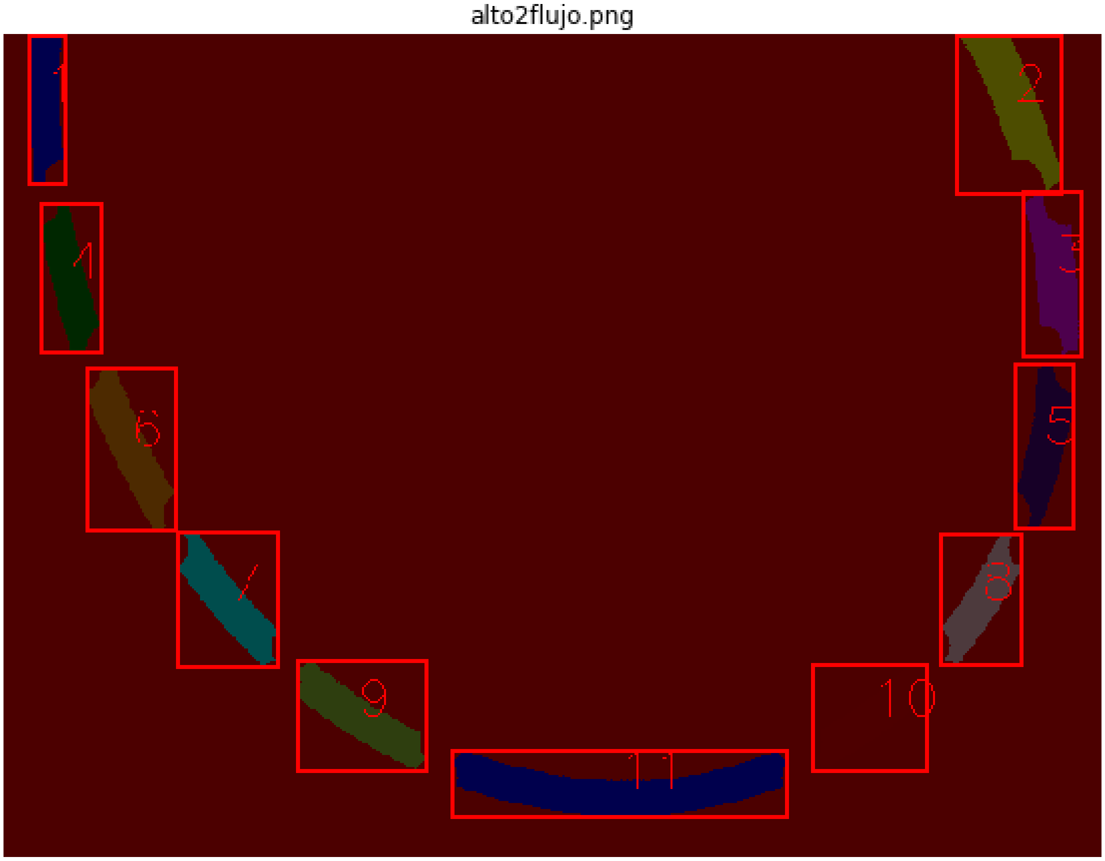
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



[76]:
```
mangueras_subdivididas_2 = {
    key: mangueras[key] * mascaras_subdivididas_2[key] for key in llaves
}
```

[84]:
```
mangueras_agregadas_2 = {
    key: mangueras[key] * mascaras_agregadas_2[key] for key in llaves
}
```

[85]:
```
mangueras_agregadas_6 = {
    key: mangueras[key] * mascaras_agregadas_6[key] for key in llaves
}
```

[86]:
```
#plt.imshow(mangueras_subdivididas_2[llaves[0]][1])
```

[87]:
```
#plt.imshow(mangueras_subdivididas_2[llaves[0]], cmap='gray')
```

[90]:
```
mangueras_subdivididas_2[llaves[0]][0]
```

```
[90]: array([[0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             …,
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0]], dtype=uint8)
```

```
[100]:
```

```
[162]: def gen_tabla_subdivididas(y: Dict[str, List[np.ndarray]]) -> pd.core.frame.
        ↪DataFrame:
           """
               Generate dianostic tables.
           """

           return pd.core.frame.DataFrame({
               "N. Region": [i for i in range(1, len(y)+1)],
               "Num. pixeles": [len(x[ x != 0]) for x in y ],
               "Intensidad media": [ x[ x != 0].mean() for x in y ],
               "Varianza": [ x[ x != 0].std()**2 for x in y ],
               "Desv std": [ x[ x != 0].std() for x in y ],
              "Error std": [ x[ x != 0].std()**2 / len(x[ x != 0 ].flatten()) for x in y]
           })
```

```
[154]: y = mangueras_subdivididas_2[llaves[0]]
       [type(x) for x in y]
```

```
[154]: [numpy.ndarray, numpy.ndarray]
```

```
[163]: tabla_2_secciones = {
           key: gen_tabla_subdivididas(value) for key, value in␣
        ↪mangueras_subdivididas_2.items()
       }
```

```
[164]: tabla_6_secciones = {
           key: gen_tabla_subdivididas(value) for key, value in␣
        ↪mangueras_subdivididas_6.items()
       }
```

```
[165]: tabla_6_secciones = {
           key: gen_tabla_subdivididas(value) for key, value in␣
        ↪mangueras_subdivididas_6.items()
       }
```

```
[192]: k = llaves[0]
       print(k)
       tabla_6_secciones[k]
```

```
altoflujo.png
```

```
[192]:   N. Region  Num. pixeles  Intensidad media   Varianza  Desv std  Error std
       0         1          4446        135.325011  98.784831  9.939056   0.022219
       1         2          5139        133.464098  75.417032  8.684298   0.014675
       2         3          5142        133.780630  65.269263  8.078939   0.012693
       3         4          4603        134.711275  59.072189  7.685843   0.012833
       4         5          4997        135.295377  61.796282  7.861061   0.012367
       5         6          4618        135.325249  61.070480  7.814760   0.013224
```

```
[193]: k = llaves[1]
       print(k)
       tabla_6_secciones[k]
```

```
bajo1flujo.png
```

```
[193]:   N. Region  Num. pixeles  Intensidad media   Varianza  Desv std  Error std
       0         1          4211        135.252909  94.696664  9.731221   0.022488
       1         2          4994        133.903484  73.682315  8.583840   0.014754
       2         3          5095        134.155054  63.911189  7.994447   0.012544
       3         4          4704        134.813776  57.357327  7.573462   0.012193
       4         5          4931        135.631312  58.898342  7.674526   0.011945
       5         6          4669        135.963804  60.021607  7.747361   0.012855
```

```
[194]: k = llaves[2]
       print(k)
       tabla_6_secciones[k]
```

```
bajo2flujo.png
```

```
[194]:   N. Region  Num. pixeles  Intensidad media   Varianza  Desv std  Error std
       0         1          4624        132.048875  93.308165  9.659615   0.020179
       1         2          5250        130.448571  72.216498  8.498029   0.013756
       2         3          5239        130.508303  60.959608  7.807663   0.011636
       3         4          4707        131.105800  56.307481  7.503831   0.011962
       4         5          5053        131.859489  60.152036  7.755774   0.011904
       5         6          4782        132.046006  60.474671  7.776546   0.012646
```

```
[195]: k = llaves[3]
       print(k)
       tabla_6_secciones[k]
```

```
bajo3flujo.png
```

[195]:

```
   N. Region  Num. pixeles  Intensidad media   Varianza  Desv std  Error std
0          1          4709        132.803355  84.243769  9.178440   0.017890
1          2          5426        131.480464  66.476305  8.153300   0.012251
2          3          5269        131.306510  56.614156  7.524238   0.010745
3          4          4861        131.712816  53.033757  7.282428   0.010910
4          5          5172        132.368329  55.829337  7.471903   0.010795
5          6          4882        132.752560  57.013333  7.550717   0.011678
```

[196]:

```
k = llaves[4]
print(k)
tabla_6_secciones[k]
```

```
alto2flujo.png
```

[196]:

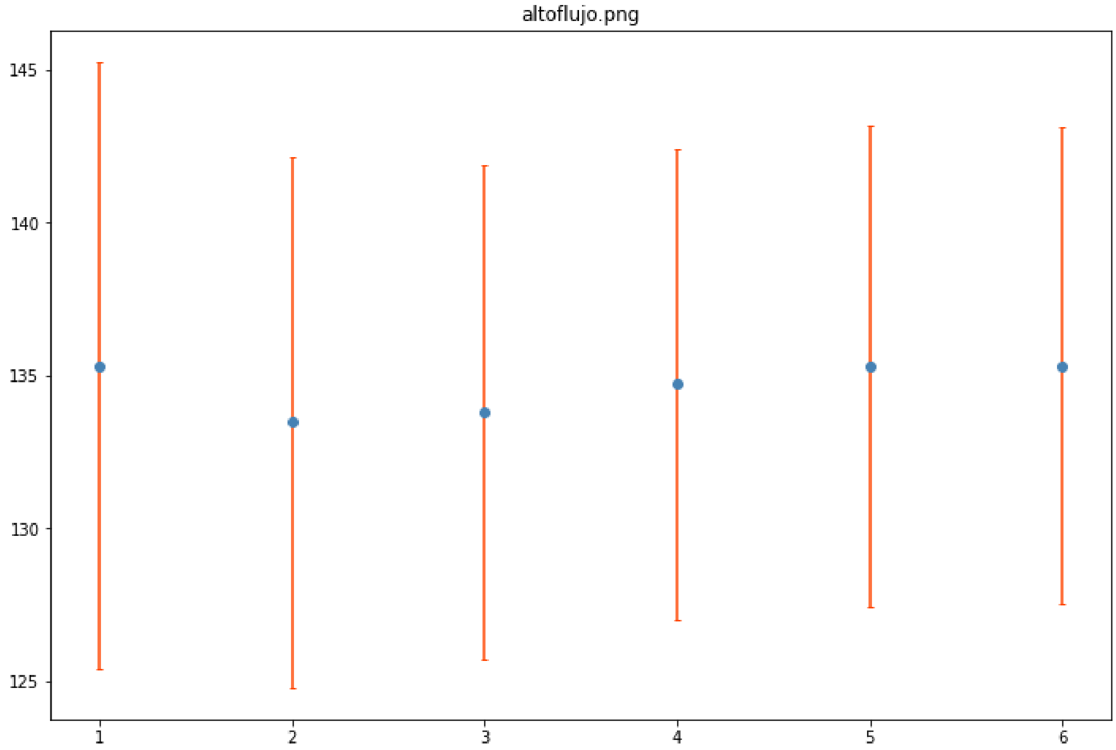
```
   N. Region  Num. pixeles  Intensidad media   Varianza  Desv std  Error std
0          1          4624        136.776384  96.180532  9.807167   0.020800
1          2          5246        135.435570  77.471545  8.801792   0.014768
2          3          5529        135.479834  69.078676  8.311358   0.012494
3          4          5003        136.034579  64.633823  8.039516   0.012919
4          5          5230        136.551243  66.384659  8.147678   0.012693
5          6          4964        136.613417  66.705308  8.167332   0.013438
```

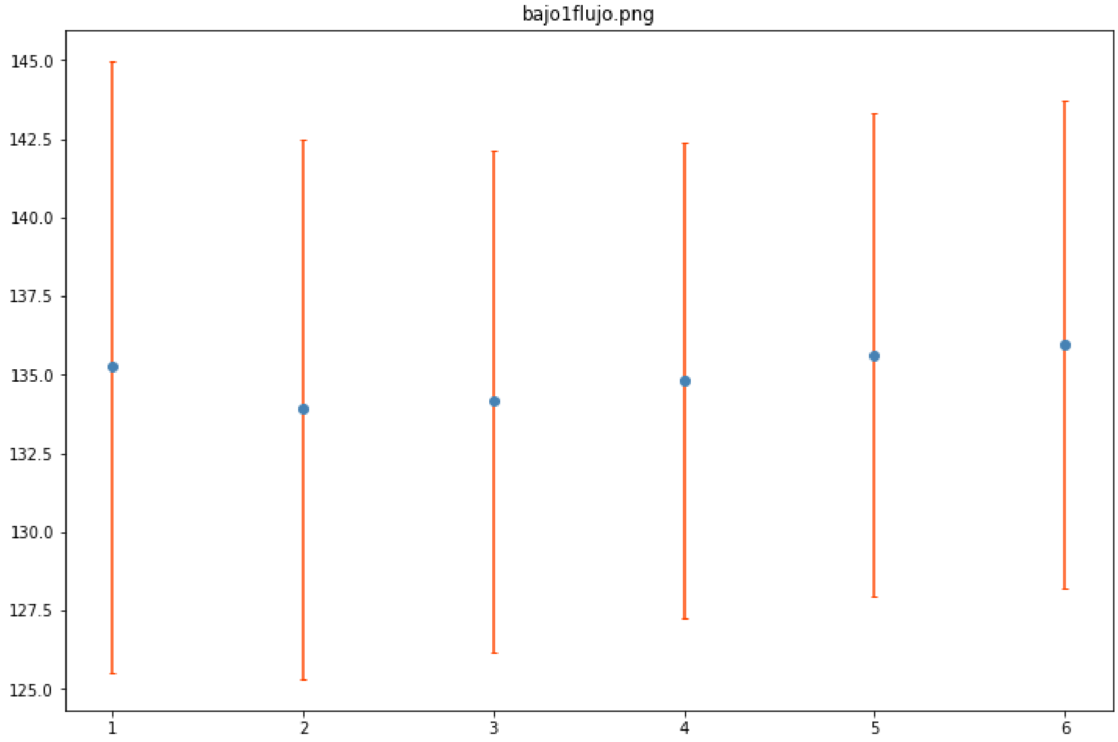
[ ]:

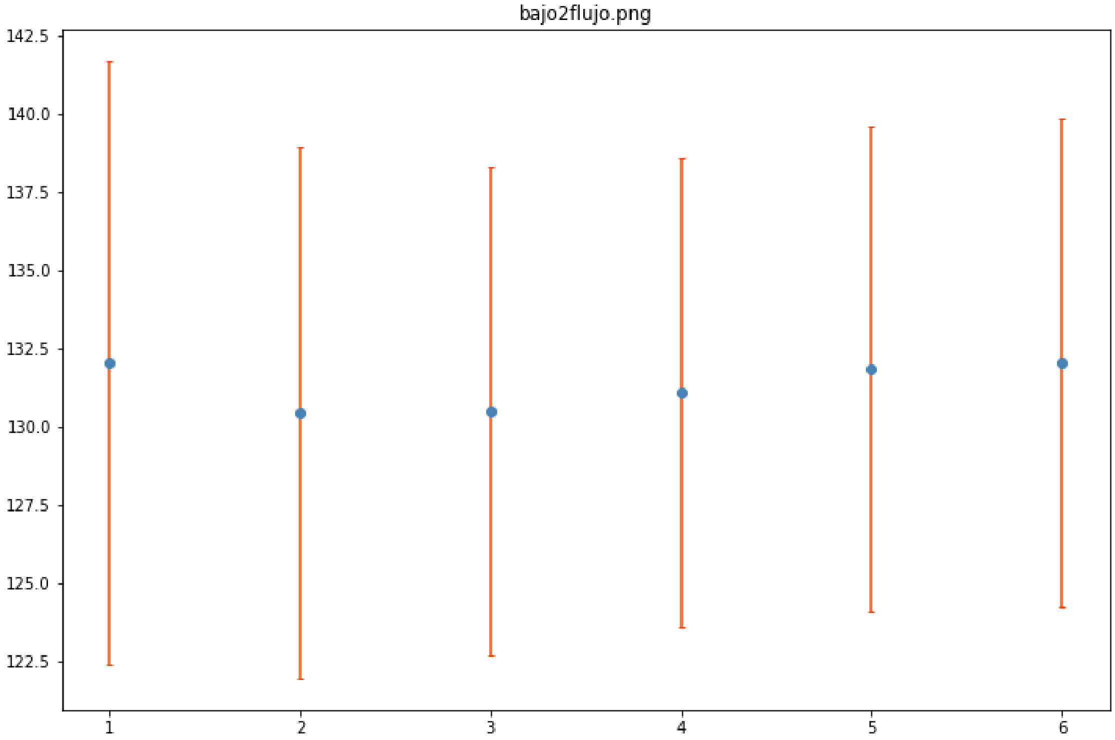
```

```

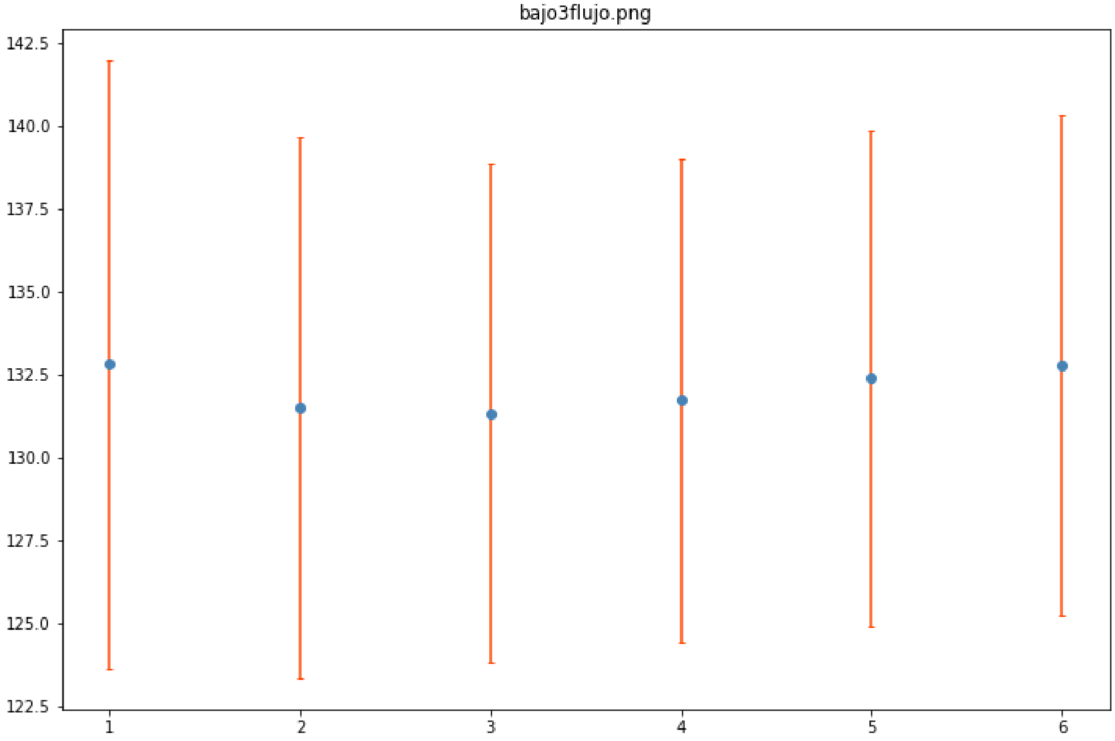
[175]:

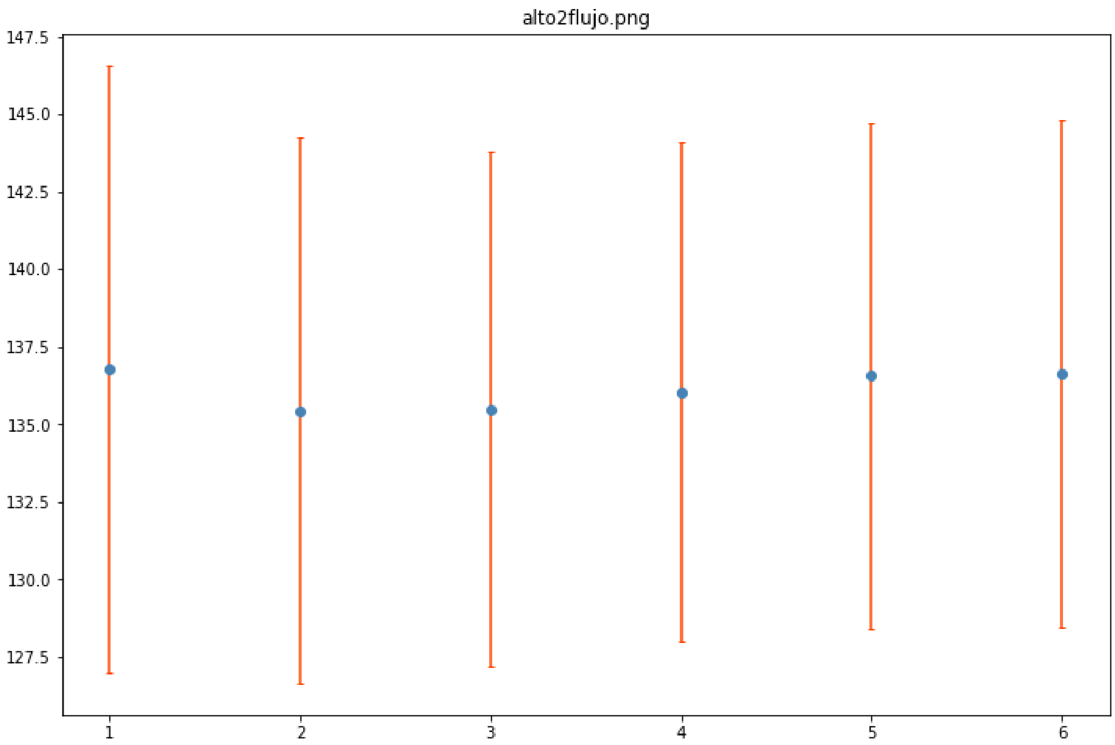
```
for nombre, dtf in tabla_6_secciones.items():
    plt.figure()
    plt.errorbar(x=dtf["N. Region"], y=dtf['Intensidad media'], yerr=dtf['Desv␣
 ↪std'],
                 fmt='o', ecolor='orangered', color='steelblue', capsize=2)
    plt.title(nombre)
```


altoflujo.png
145
140
135
130
125
1
2
3
4
5
6


bajo1flujo.png
145.0
142.5
140.0
137.5
135.0
132.5
130.0
127.5
125.0
1
2
3
4
5
6


bajo2flujo.png
142.5
140.0
137.5
135.0
132.5
130.0
127.5
125.0
122.5
1
2
3
4
5
6

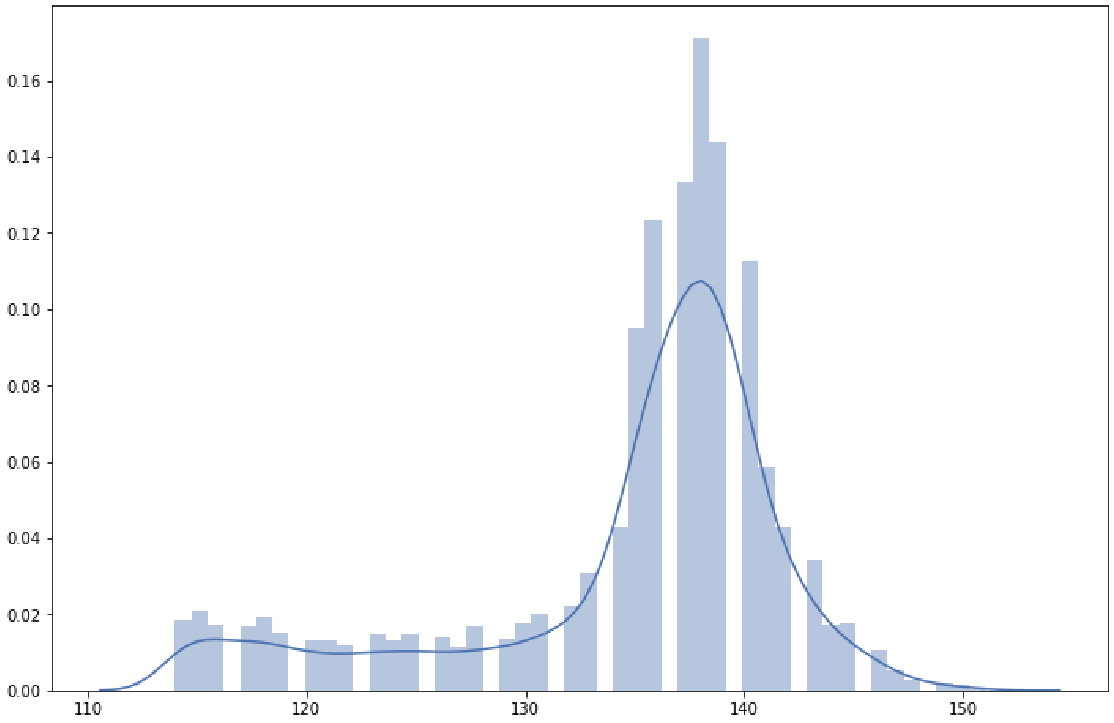

bajo3flujo.png
142.5
140.0
137.5
135.0
132.5
130.0
127.5
125.0
122.5
1
2
3
4
5
6



```
[119]: _sec1  = mangueras_subdivididas_2[llaves[1]][0]
       _data1 = _sec1[ _sec1 != 0].flatten()
       print(_data1.mean(), _data1.std()**2)
       sns.distplot(_data1)
```

```
134.84886564898372 55.51515284161077
```

```
[119]: <matplotlib.axes._subplots.AxesSubplot at 0x1c32d16c90>
```



[68]:
```
_plot = False

if _plot:
    for llave in llaves:
        plt.figure()
        plt.imshow(mangueras_subdivididas_6[llave], cmap='gray')
```

[126]:
```
norm_values: Dict[str, float] = {}
for key, image in segmented_ref_reg.items():
     norm_values.update({
         key: image[ image != 0 ].mean()
     })
```

[128]:
```
mangueras_normalizadas = {
    key: mangueras[key] / norm_values[key] for key in llaves
}
```

[133]:
```
segmented_normalised_ref_reg = {
    key: mangueras_normalizadas[key] * region_ref4[key] for key in llaves
}
```

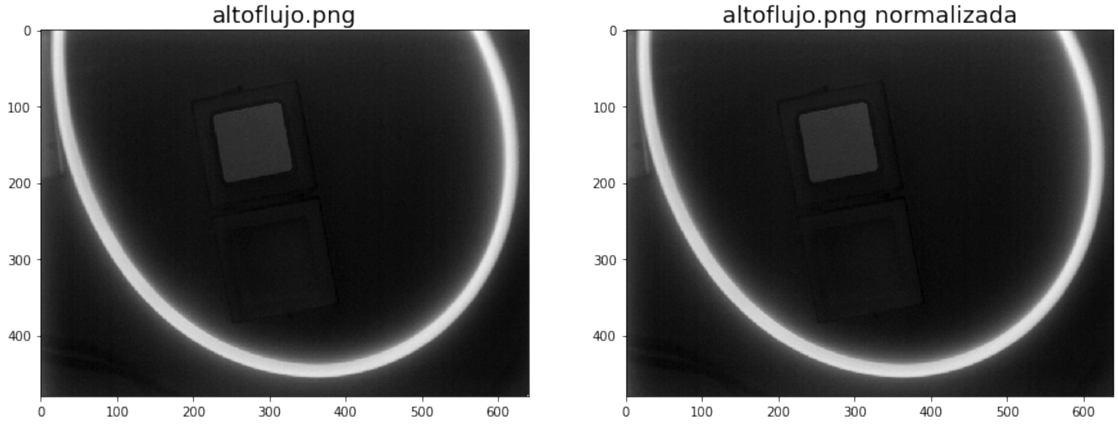
[136]:
```
norm_region_info_list = list(map(
    lambda x, y: pd.core.series.Series(x[ x != 0].flatten(), name=y),␣
 ↪segmented_normalised_ref_reg.values(), segmented_normalised_ref_reg.keys()
))
region_info = pd.concat(norm_region_info_list, axis=1)
region_info.describe()
```

[136]:

```
       altoflujo.png  bajo1flujo.png  bajo2flujo.png  bajo3flujo.png  \
count    8111.000000     8144.000000    10736.000000    18376.000000
mean        1.000000        1.000000        1.000000        1.000000
std         0.015748        0.016044        0.047776        0.041483
min         0.893125        0.884693        0.880149        0.916449
25%         0.993601        0.996679        0.994454        0.964683
50%         1.004765        0.996679        1.017315        0.988800
75%         1.004765        1.007878        1.028746        1.037035
max         1.105242        1.108666        1.120190        1.145561

       alto2flujo.png
count    10839.000000
mean         1.000000
std          0.038203
min          0.894497
25%          0.988655
50%          1.012194
75%          1.023964
max          1.129891
```

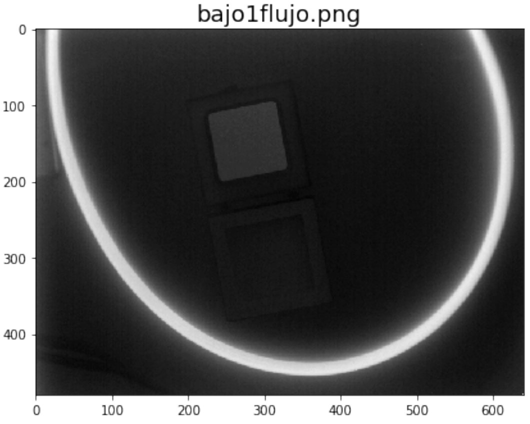
[199]:

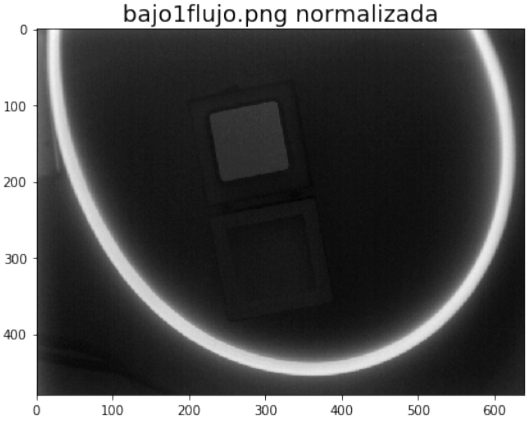
```
for llave in llaves:
    #plt.figure()
    utils.side_by_side(
        mangueras[llave], mangueras_normalizadas[llave],
        title1=llave, title2=f"{llave} normalizada"
    )
```

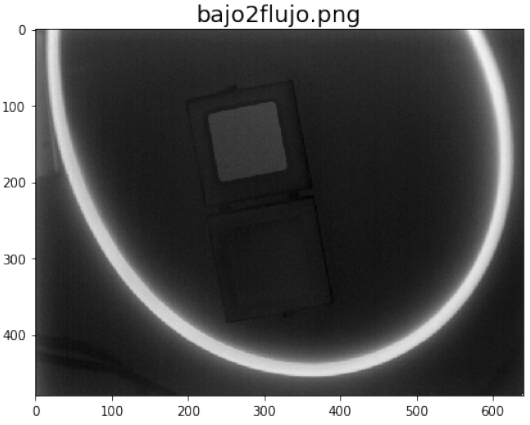


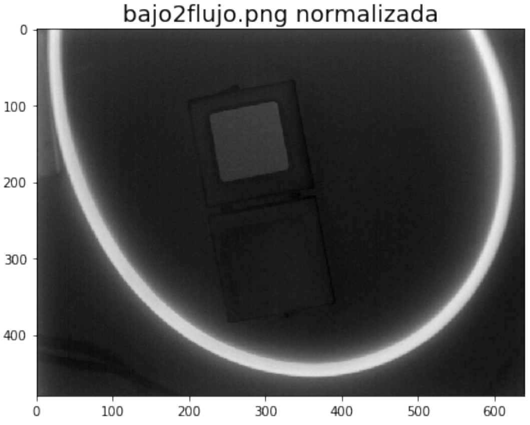
altoflujo.png

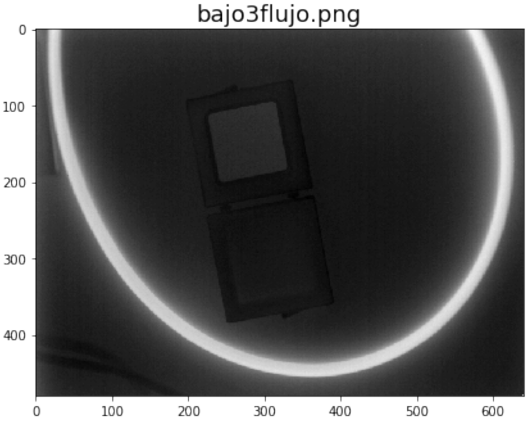
altoflujo.png normalizada

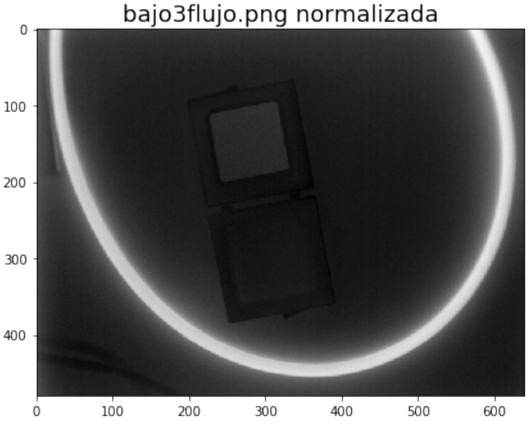

bajo1flujo.png

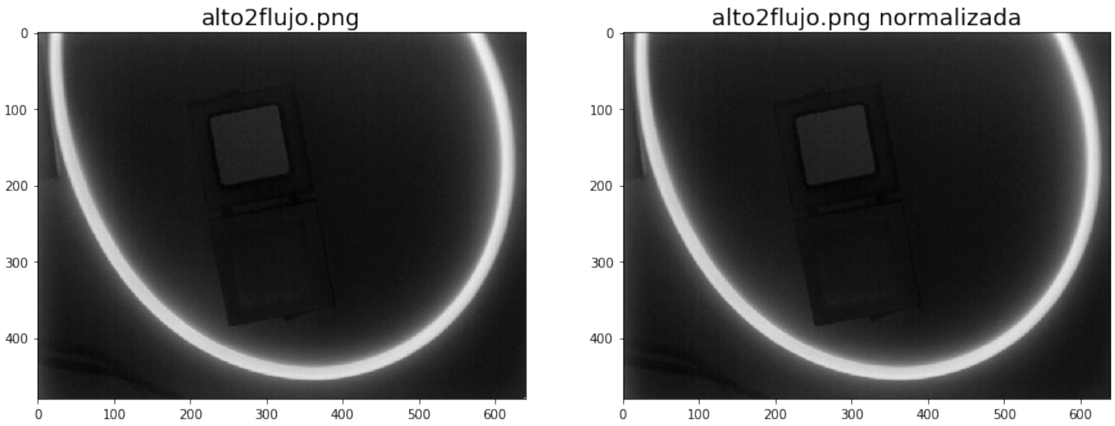

bajo1flujo.png normalizada


bajo2flujo.png

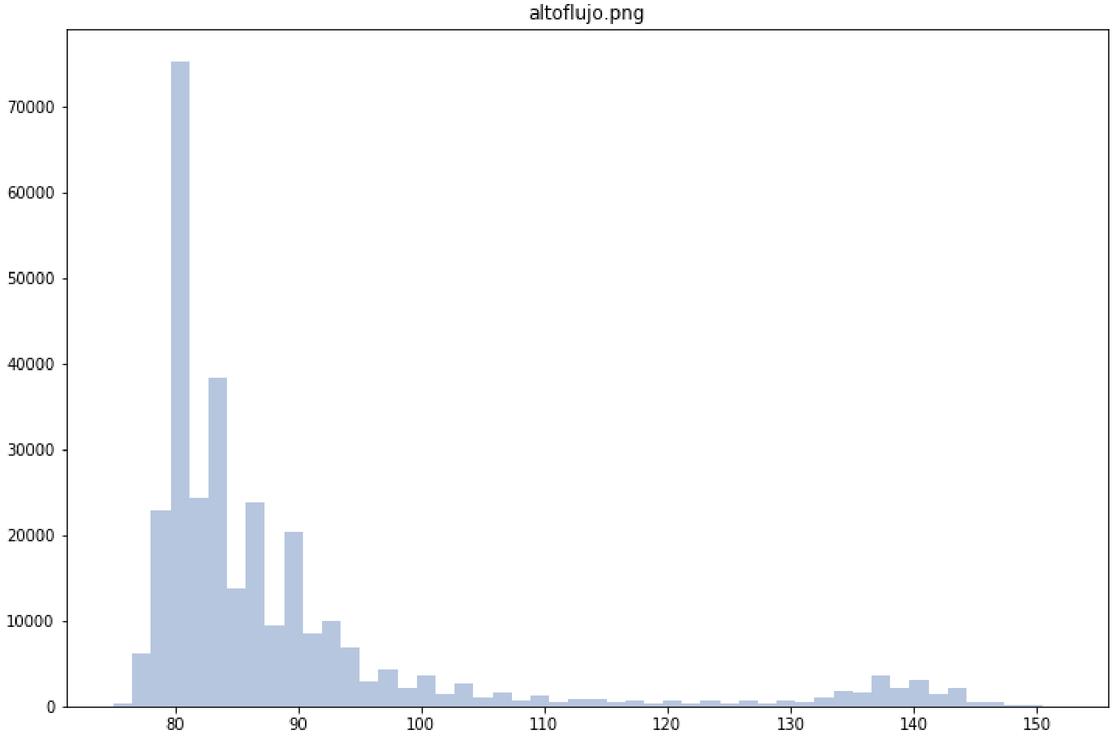

bajo2flujo.png normalizada

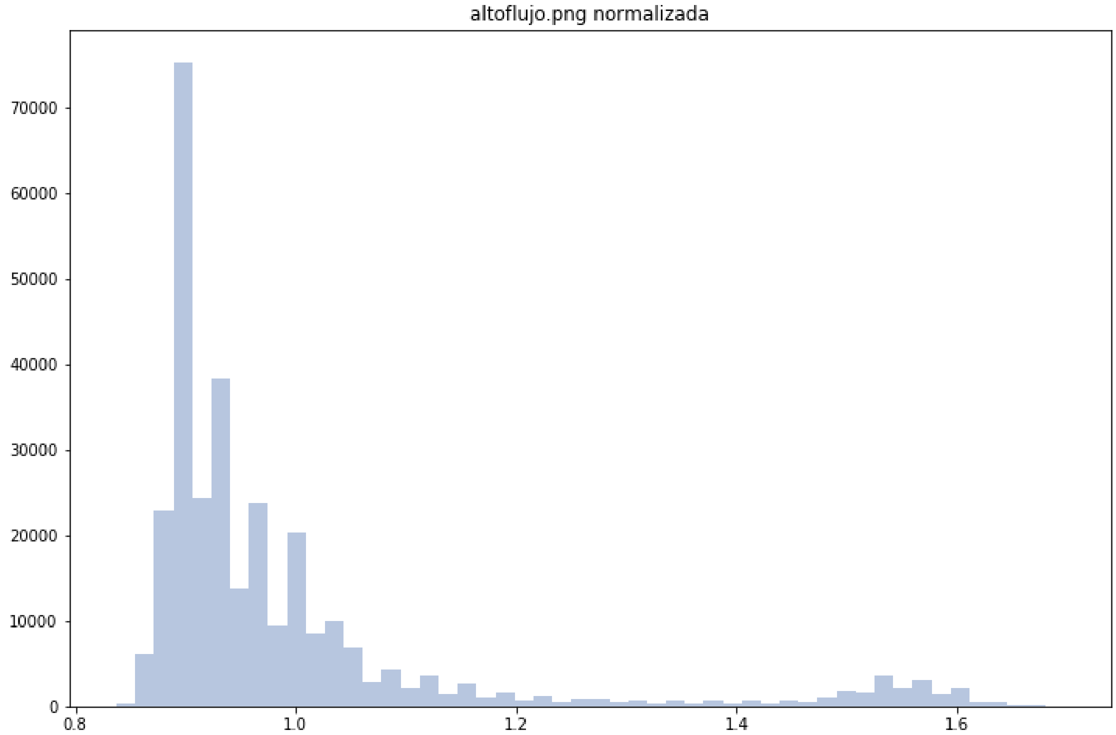

bajo3flujo.png

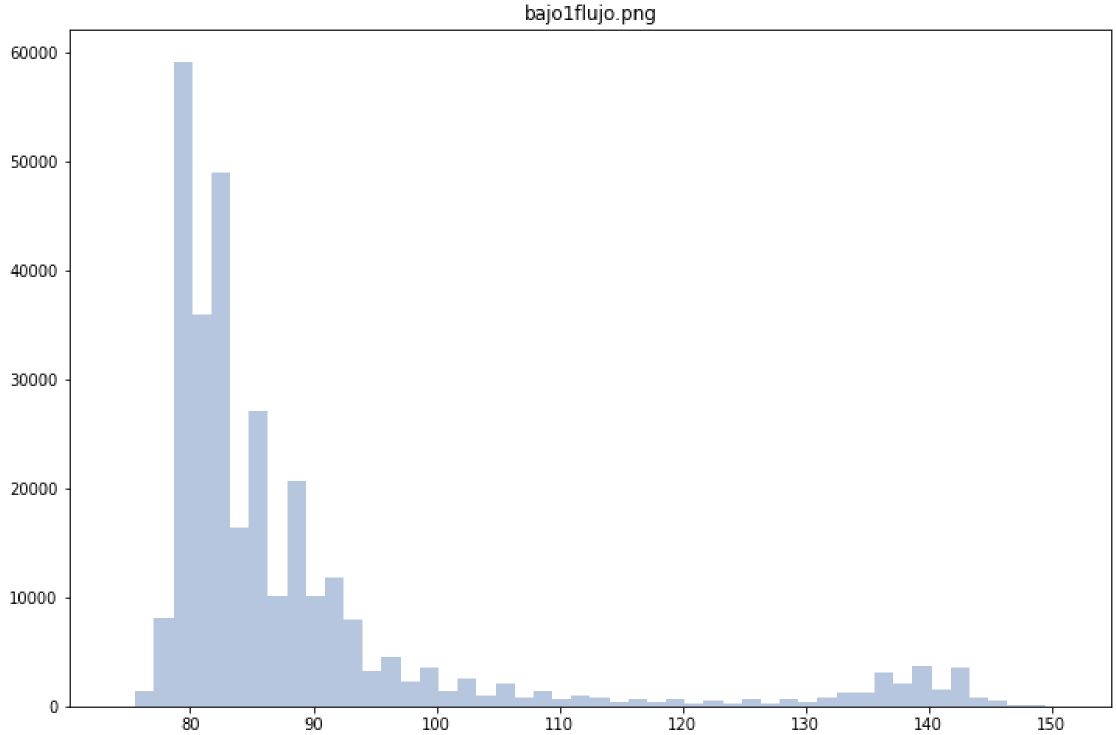

bajo3flujo.png normalizada

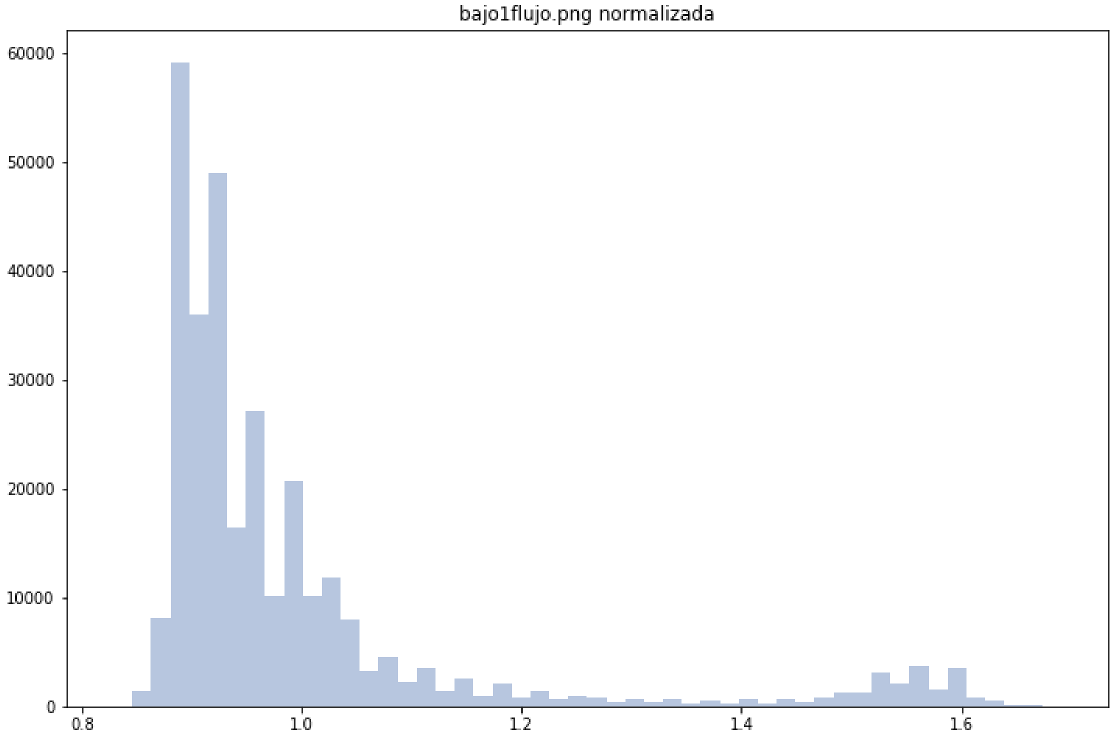


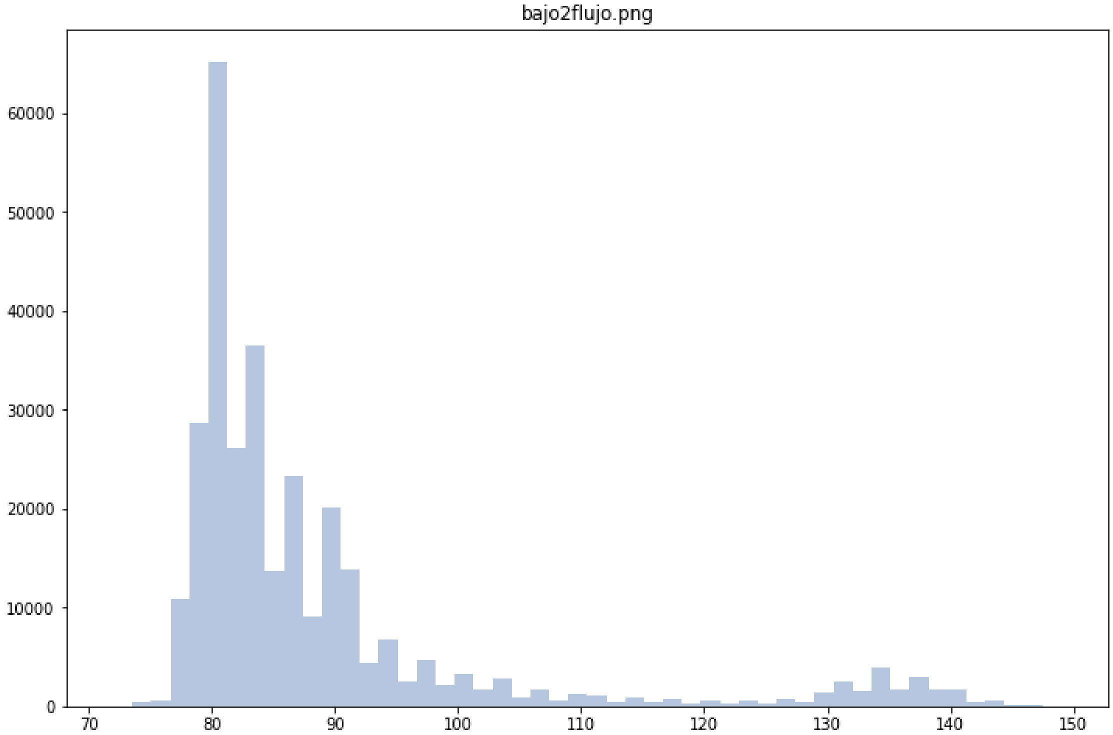
```
[206]: for llave in llaves:
           plt.figure()
           sns.distplot(mangueras[llave].flatten(), kde=False)
           plt.title(llave)
           plt.figure()
           sns.distplot(mangueras_normalizadas[llave].flatten(), kde=False)
           plt.title(f"{llave} normalizada")
```

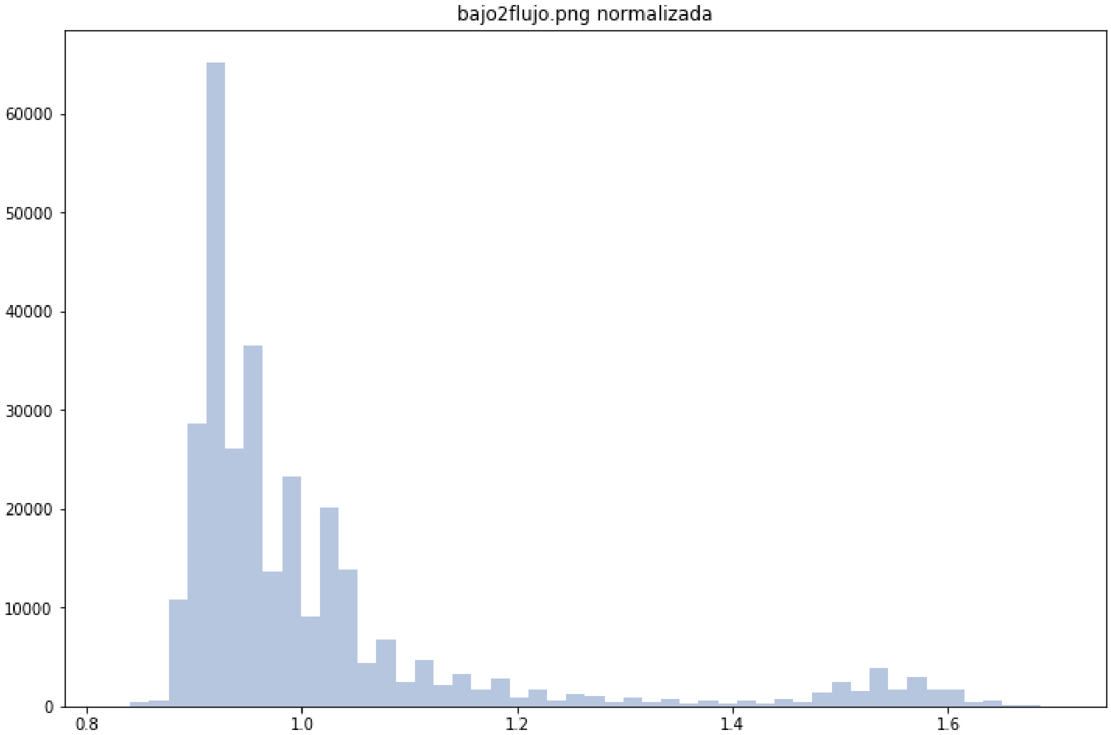


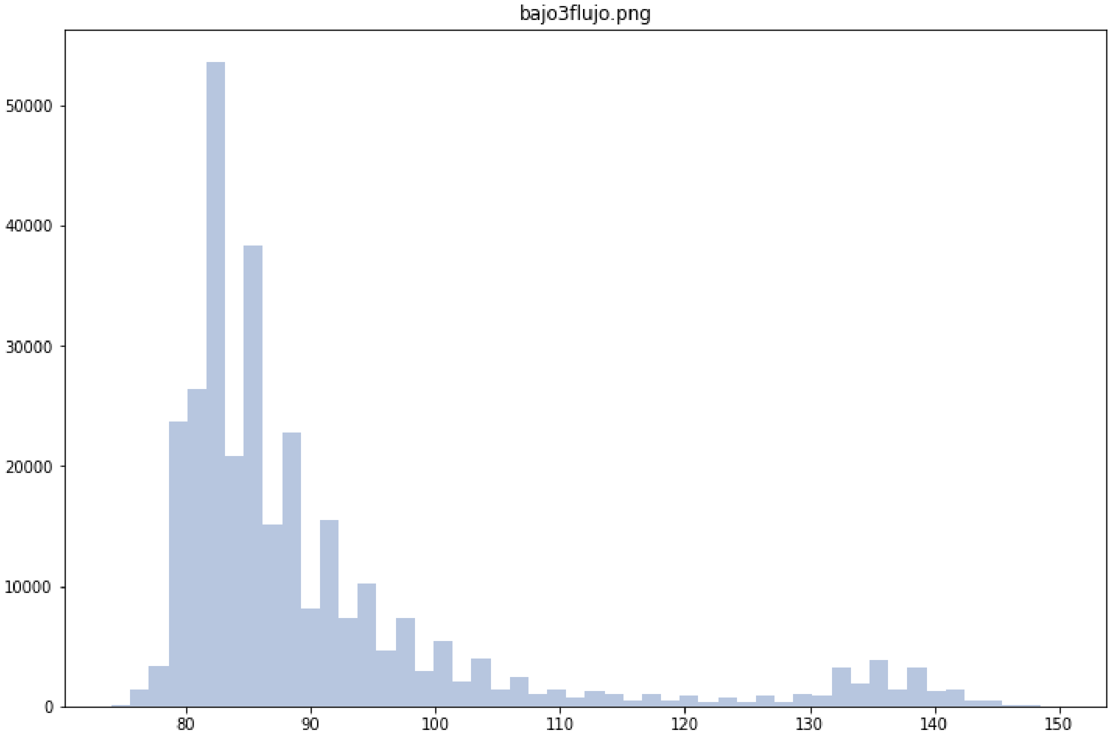

altoflujo.png normalizada
70000
60000
50000
40000
30000
20000
10000
0
0.8
1.0
1.2
1.4
1.6

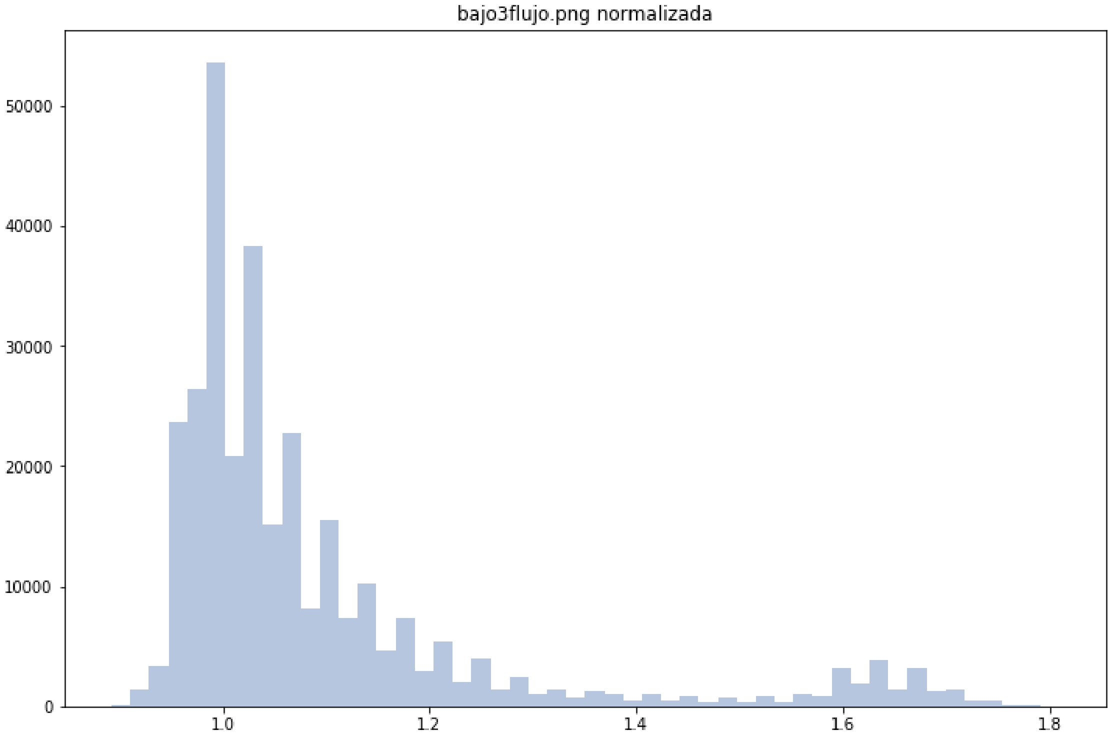

bajo1flujo.png
60000
50000
40000
30000
20000
10000
0
80
90
100
110
120
130
140
150

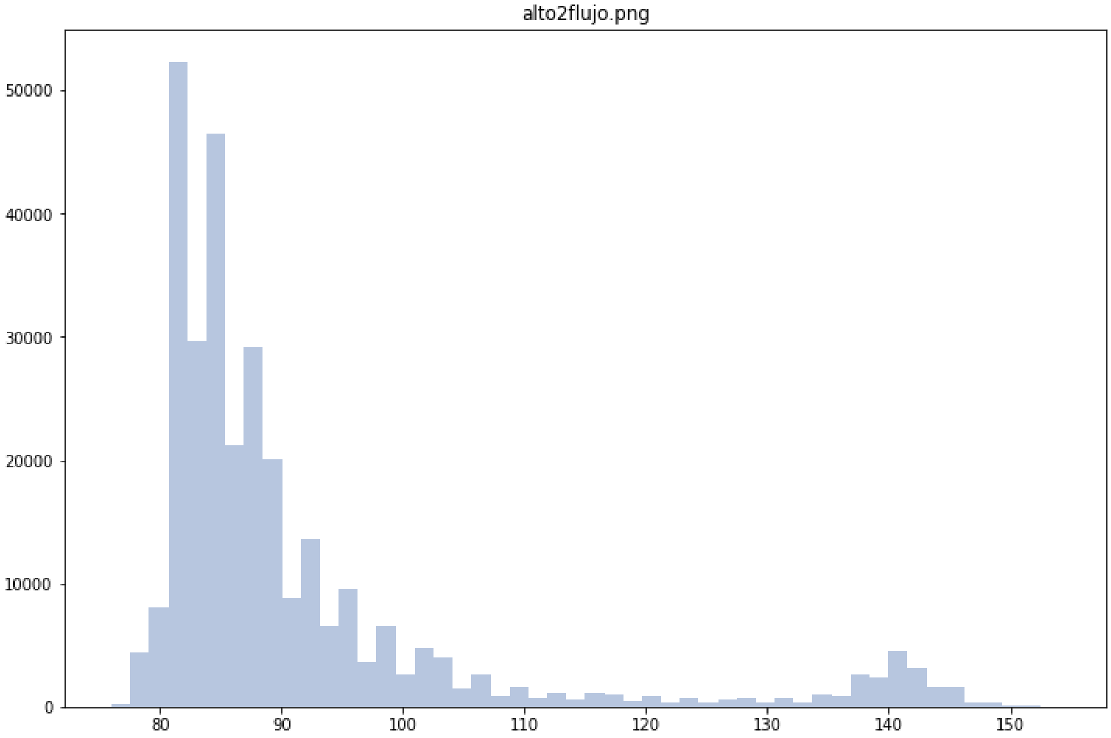

bajo1flujo.png normalizada
0
10000
20000
30000
40000
50000
60000
0.8
1.0
1.2
1.4
1.6

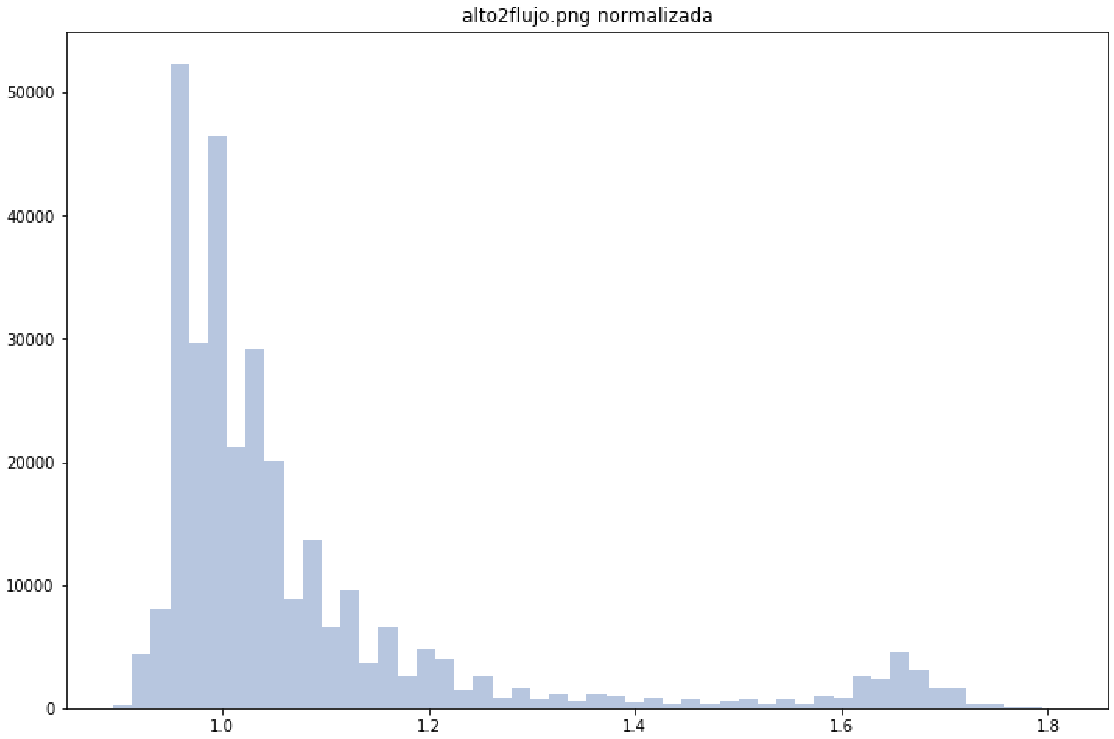

bajo2flujo.png
0
10000
20000
30000
40000
50000
60000
70
80
90
100
110
120
130
140
150


bajo2flujo.png normalizada
60000
50000
40000
30000
20000
10000
0
0.8
1.0
1.2
1.4
1.6


bajo3flujo.png
50000
40000
30000
20000
10000
0
80
90
100
110
120
130
140
150


bajo3flujo.png normalizada
50000
40000
30000
20000
10000
0
1.0
1.2
1.4
1.6
1.8


alto2flujo.png
50000
40000
30000
20000
10000
0
80
90
100
110
120
130
140
150


alto2flujo.png normalizada
0
10000
20000
30000
40000
50000
1.0
1.2
1.4
1.6
1.8

[ ]: