```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#!/usr/bin/env python
# coding: utf-8

# In[90]:


import numpy as np
import cv2
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
from matplotlib import pylab
# pylab.rcParams['figure.figsize'] = (10, 5)
import matplotlib as mpl
import seaborn as sns
import statistics
from skimage.feature import peak_local_max
from skimage.morphology import watershed
from scipy import ndimage
import imutils
# mpl.rcParams['figure.dpi'] = 150
#%matplotlib qt

# In[2]:


#get_ipython().run_line_magic('matplotlib', 'qt')

# Better to avoid inline mode, as the particles are quite small most of the times.


# #### Step 1 : Get the scale, if you don't already know it.

# In[43]:


def get_tem_scale(img_path,y1=None,y2=None,x1=None,x2=None, threshold_type = cv2.THRESH_BINARY,lower_thresh = 220):

    '''
    This function takes in an image and tries to find the scale by measuring the line segment usually given at the bottom left
    if the TEM micrograph. It approximates a rectangle for this and reports the width of rectangle as the scale.

    Adjust the crop paramters x1,x2,y1 and y2 to fit the scale within the cropped area.

    Parameters:
    img_path :  Path of the TEM image.
    y1 = start index for cropping along vertical axis, must be an integer.
    y2 = end index for cropping along vertical axis, must be an integer.
    x1 = start index for cropping along horizontal axis, must be an integer.
    x2 = end index for cropping along horizontal axis, must be an integer.

    Prints width of the approximating rectangle.

    '''

    img = cv2.imread(img_path)
    '''
    if np.any(img):
        pass
    else:
        print('Image could not be read, check path or check if image is corrupt.')
        return None
    '''
    if y1 == None:
        y1 = int(0.85*img.shape[0])
    if y2 == None:
        y2 = int(0.99*img.shape[0])
    if x1 == None:
        x1 = int(0.5*img.shape[1])
    if x2 == None:
        x2 = int(1*img.shape[1])
```

```
    crop_img = img[y1:y2,x1:x2]
    print(img.shape)
    if np.any(crop_img):
        pass
    else:
        print('Cropped Image is empty, check crop dimensions.')

    imshow(crop_img)
    scale_gray = cv2.cvtColor(crop_img,cv2.COLOR_BGR2GRAY)

    # choose threshold type  as cv2.THRESH_BINARY_INV if scale region is black.
    if threshold_type == cv2.THRESH_BINARY_INV:
        lower_thresh = 0
    ret, thresh = cv2.threshold(scale_gray, lower_thresh, 255, threshold_type)
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_FLOODFILL, cv2.CHAIN_APPROX_SIMPLE)

    # filter noisy detection
    contours = [c for c in contours if cv2.contourArea(c) > 200]
    contours.sort(key=lambda c: (cv2.boundingRect(c)[1], cv2.boundingRect(c)[0]))

    x,y,w,h = cv2.boundingRect(contours[-1])
    thresh=cv2.rectangle(thresh, (x,y),(x+w,y+h+30), (0,0,255), 20)
    print('x : %f , y = %f , w = %f , h = %f '%(x,y,w,h))
    print('The width in pixels is : ',w)
    #cv2.imshow('scale_marked',crop_img)
    imshow(thresh)
    cv2.imwrite('/content/scale.jpg',thresh)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    return w



# #### Step 2: Find all features, irrespective of size and generate contours by thresholding.

# In[4]:


def find_particles(img_path,scale=None,thresh = 80,save_images = False):

    '''
    This function finds particles by contouring, Image which is loade converted to grayscale.
    A gaussian Blur is then used to remove a bit of noise in the images, which helps with over-detection.
    Threshold style by default is cv2.THRESH_BINARY.

    Threshold by default is 45. Change this according to your data.

    Parameters:
    img_path :  Path of the TEM image.
    scale : Either already known or found from get_tem_scale()
    thresh :  lower limit of threshold.
    save_images : If true, the images are saved to same directory as the original images, False by default.

    '''
    # Read the image
    img = cv2.imread(img_path)
    print(img.shape)
    '''
    if np.any(img):
        pass
    else:
        print('Image could not be read, check path or check if image is corrupt.')
        return None
    '''
    #img = cv2.resize(img,(0,0),fx=0.25,fy=0.25)

    # Gaussian Blur ot reduce noise
    #img = cv2.GaussianBlur(img,(5,5),0)
    img =   cv2.medianBlur(img, 5)
    # convert to grascale
    gray= cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)

    # thresholding, making a binary image.
    ret,thresh = cv2.threshold(gray,thresh,255,cv2.THRESH_BINARY)
    th2 = cv2.adaptiveThreshold(gray,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,5,2)
    ret3,th3 = cv2.threshold(gray,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
```

```
    thresh = cv2.Canny(thresh, 30, 150)
    #ret3,thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    #img_thresh = cv2.adaptiveThreshold(gray,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,1151,1)
    #cv2.namedWindow("Threshold image", cv2.WINDOW_NORMAL)
    #cv2.imshow('Threshold image',thresh)
    contours,hierarchy = cv2.findContours(thresh,cv2.RETR_CCOMP,cv2.CHAIN_APPROX_SIMPLE)
    image = img.copy()
    shifted = cv2.pyrMeanShiftFiltering(image, 21, 51)
    gray = cv2.cvtColor(shifted, cv2.COLOR_BGR2GRAY)
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
    D = ndimage.distance_transform_edt(thresh)
    localMax = peak_local_max(D, indices=False, min_distance=4, labels=thresh)
    markers = ndimage.label(localMax, structure=np.ones((3, 3)))[0]
    labels = watershed(-D, markers, mask=thresh)
    new_con=[]
    for label in np.unique(labels):
        # if the label is zero, we are examining the 'background'
        # so simply ignore it
        if label == 0:
          continue
        # otherwise, allocate memory for the label region and draw
        # it on the mask
        mask = np.zeros(gray.shape, dtype="uint8")
        mask[labels == label] = 255
        # detect contours in the mask and grab the largest one
        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)
        c = max(cnts, key=cv2.contourArea)
        new_con.append(c)
        # draw a circle enclosing the object
        ((x, y), r) = cv2.minEnclosingCircle(c)
        cv2.circle(image, (int(x), int(y)), int(r), (0, 255, 0), 2)
        cv2.putText(image, "#{}".format(label), (int(x) - 10, int(y)), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

    img2 = img.copy()
    cv2.imwrite('/content/watershed.jpg',thresh)
    index = -1
    thickness = 1
    color = (255,0,0)
    '''
    # drawing contours.
    cv2.drawContours(img2,contours,index,color,thickness)
    cv2.namedWindow("contours", cv2.WINDOW_NORMAL)

    #cv2.imshow('contours',img2)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    '''
    cv2.drawContours(img2,contours,index,color,thickness)
    # saving images
    if save_images==True:
        cv2.imwrite(img_path+'_thresh.tif',thresh)
        cv2.imwrite(img_path+'_contours.tif',img2)
    return contours,hierarchy,new_con


# #### Step 3: Plot the size distribution.

# In[93]:


def size_distribution_plot(img_path,contours=None,scale=None,r_min=.001,r_max=.5, bins=None, save_fig = True):

    '''
    This generates a size ditribtution plot, the sizes can be managed by r_min and r_max.

    Parameters :
    img_path :  Path of the TEM image.
    contours: Obatained from find_particles()
    scale : Already known or found from get_tem_scale()
    r_min :  Minimum radius of particles to be marked. By default, it is equal to scale.
    r_max = Maximum radius of particles to be marked. By default, it is 100 times the scale.
    bins :  bins in the histogram which is plotted.

    Prints the mean particles size. (Note that the diameters are returned.)
    If you want to exclude all particles under 10 nm, give r_max as 5.
    Returns an array of particle sizes.
```

```
    '''

    # Change this according to your data.
    if r_min == None:
        r_min = 0.5*scale
    if r_max == None:
        r_max = 50*scale
    #######

    img = cv2.imread(img_path)
    '''
    if np.any(img):
        pass
    else:
        print('Image could not be read, check path or check if image is corrupt.')
        return None
    '''
    fig,ax = plt.subplots(1,1,figsize=(10,20))
    ax.imshow(img)
    #ax[1].imshow(img)
    sizes = []
    size_2=[]
    main_scale=500
    mn=100
    mx=0
    for i in range(len(contours)):
        cnt = contours[i]
        (x,y),radius = cv2.minEnclosingCircle(cnt)
        center = (int(x),int(y))
        radius=int(radius)
        if radius > r_min*scale and radius < r_max*scale:
            sizes.append(radius)
            size_2.append(((radius*2*main_scale)/scale))
            c=plt.Circle((x,y),radius,color='r' ,linewidth=0.8, fill=False)
            if (radius*2*main_scale)/scale>mx:
              mx=(radius*2*main_scale)/scale
            if (radius*2*main_scale)/scale<mn:
              mn=(radius*2*main_scale)/scale

            # commented out in case of a lot of particles, it is a very messy plot.
            q=str(int((radius*2*main_scale)/scale))
            q+=" µm"
            ax.annotate(q,(x-10,y))

            ax.add_patch(c)
    plt.savefig('detected_particle_red_circled.png')
    sizes = (np.array(sizes)*2) # Note that these are diameters
    print(size_2)
    fig1,ax1 = plt.subplots()
    if bins == None:
        bins = len(set(size_2))
    print(bins)
    ax1.hist(size_2,facecolor='g', alpha=0.75, histtype='bar', ec='black', bins=bins)
    #sns.distplot(sizes,bins=bins)

    plt.xlabel('Diameter (µm)')
    plt.ylabel('Number of Particles')
    plt.title("Size distribution of Particles for PLLA_2MH image")
    st="Particles Statistic:\nMean: {:.2f}, SD: {:.2f} \nMax: {:.2f} Min: {:.2f}".format(np.mean(size_2),statistics.pstdev(size_2),mx,mn)
    plt.text(100, 5, st)
    print('Mean diameter : %0.2f'%(np.mean(size_2)))
    print('Total Particles : %g'%(len(size_2)))
    res = statistics.pstdev(size_2)
    print('Mean diameter : %0.2f'%res)

    if save_fig == True:
        plt.savefig("Histogram.jpg")

    plt.tight_layout()
    plt.show()
    return np.sort(sizes)


# In[ ]:
```

```
path='/content/PLGA7525_200uL_2MH.tif'
print(path)
image = cv2.imread(path)
cv2.imwrite('PLGA_2MH.jpg', image)
```

```
/content/PLGA7525_200uL_2MH.tif
True
```

```
path='/content/tem.jpg'
```

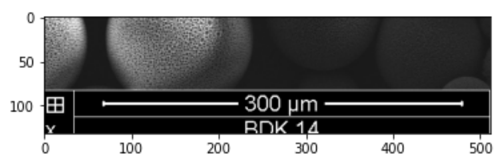
```
scal=get_tem_scale("/content/PLLA2_200uL_2MP.jpg")
```

```
(943, 1024, 3)
---------------------------------------------------------------------------
error                                     Traceback (most recent call last)
<ipython-input-11-5bc35e496766> in <module>()
----> 1 scal=get_tem_scale("/content/PLLA2_200uL_2MP.jpg")

<ipython-input-9-2905a56ec9cb> in get_tem_scale(img_path, y1, y2, x1, x2, threshold_type, lower_thresh)
     84         lower_thresh = 0
     85     ret, thresh = cv2.threshold(scale_gray, lower_thresh, 255, threshold_type)
---> 86     contours, hierarchy = cv2.findContours(thresh, cv2.RETR_FLOODFILL, cv2.CHAIN_APPROX_SIMPLE)
     87
     88     # filter noisy detection

error: OpenCV(4.1.2) /io/opencv/modules/imgproc/src/contours.cpp:197: error: (-210:Unsupported format or combination of formats) [Start]
supports CV_32SC1 images only in function 'cvStartFindContours_Impl'
```

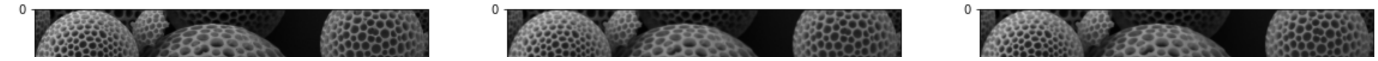
SEARCH STACK OVERFLOW




```
# here, PLLA_3MH, scale=275.  PCL_2MH, scale=232.  (PLGA_2MH,2MP,DMB: main_scale=415, mod_scale=277, zoom=200)
#(PCL_2MP,3MP: main_scale=423, mod_scale=282, zoom=400)
#(PCL_DMP,DMB,3MH; PLGA_3MH,3MP: main_scale=353, mod_scale=235, zoom=500)
#main_scale=423, mod_scale=282, zoom=300)
```

```
img = cv2.imread(path)
img1 = cv2.GaussianBlur(img,(5,5),0)
img2 =   cv2.medianBlur(img, 5)
fig,ax = plt.subplots(1,3,figsize=(20,10))
ax[0].imshow(img)
ax[1].imshow(img1)
ax[2].imshow(img2)
    # thresholding, making a binary image.
#ret,thresh = cv2.threshold(gray,thresh,255,cv2.THRESH_BINARY)
```

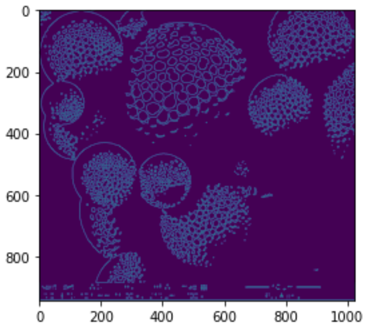
```
<matplotlib.image.AxesImage at 0x7f9a85f68a90>
```



```
gray= cv2.cvtColor(img2,cv2.COLOR_RGB2GRAY)
blurred = cv2.GaussianBlur(gray, (5, 5), 0)
ret,th1 = cv2.threshold(gray,80,255,cv2.THRESH_BINARY)
ret3,th3 = cv2.threshold(gray,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
th2 = cv2.adaptiveThreshold(gray,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,5,2)
cu = cv2.Canny(th1, 30, 150)
```

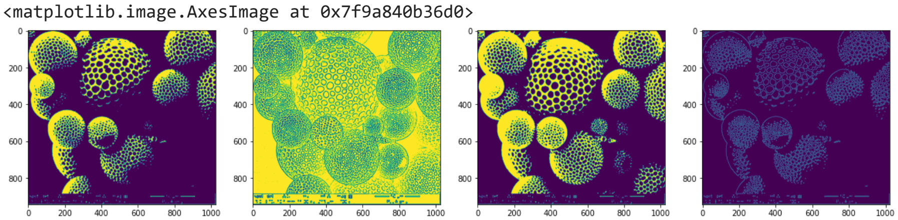
```
imshow(cu)
```

```
<matplotlib.image.AxesImage at 0x7f9a85a23d90>
```



```
fig,ax = plt.subplots(1,4,figsize=(20,10))
ax[0].imshow(th1)
ax[1].imshow(th2)
ax[2].imshow(th3)
ax[3].imshow(cu)
```

```
<matplotlib.image.AxesImage at 0x7f9a840b36d0>
```



```
contour,hierarchy,c=find_particles("/content/detected_particle.jpg", scale=235)
```

```
(591, 682, 3)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:159: FutureWarning: indices argument is deprecated and will be removed in v
/usr/local/lib/python3.7/dist-packages/skimage/morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated an
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
```

```
scale=int((682/1024)*423)
print(scale)
```

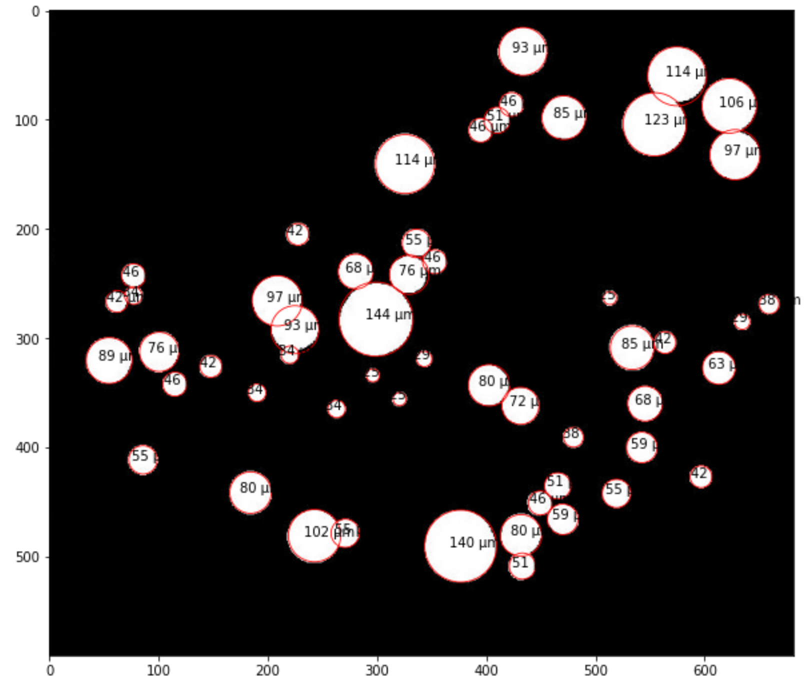
```
281
```

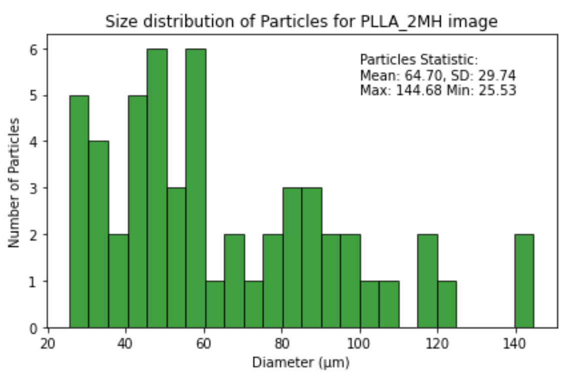
```
np.asarray(hierarchy).shape
count=0
for i in range(len(hierarchy[0])):
  if hierarchy[0][i][3]==-1:
    count+=1
print(len(hierarchy[0]))
print(count)
print(hierarchy[0][0][3])
```

```
18169
37
-1
```

```
a=size_distribution_plot("/content/detected_particle.jpg",contours=c,scale=235)
```

```
[93.61702127659575, 114.8936170212766, 46.808510638297875, 106.38297872340425, 85.106382
24
Mean diameter : 64.70
Total Particles : 54
Mean diameter : 29.74
```






```
a=[10.0,12.0,20.0,13.0]
b=[]
for i in a:
  b.append(int(i*(50/166)))
b
```

```
[3, 3, 6, 3]
```