Informe de actividades Susana Márquez Fondecyt 1191893 etapa Abril - Julio 2020

I. OBJECTIVE

To track Paenibacillus head, first it is important defining their position. To tackle this problem, we consider that the main movment between two time-lapses will be done by the heads. Then, we subtract different picture and propose .

II. STEPS TO FIND HEADS

The process to find the Paenibacillus' heads could be divided into three general steps; reading and adaptation of the pictures, the recognition of the contours from the differences of the pictures in time, and finally the grouping of points that give clue about where there are the heads. In each step, there are different filters with numbers that could be changed to improve the performance of the program.

In the next section, all the details will be explained including the Python coding and the mentioned filter numbers.

A. Read and adapt images

To identify heads, we consider the subtraction of a picture in four different time-lapses. The program starts by reading the images (lines 1-9). The first step is to define the parameters for the resizing, in this case, 40% (lines 10-13). After that, to facilitate the process, each picture is resized, converted to black and white, and smoothed by the OpenCV filter GaussianBlur (lines 14-17).

```
1  k0=0
2  in_fns = glob("./pictures/*.tif")
3  in_fns = sorted(in_fns)
4  fig,ax = plt.subplots(1)
5  (xi,yi,xf,yf)=(0,0,1920,2560)
6  img0A = cv2.imread(in_fns[k0])[yi:yf,xi:xf]
7  img0B = cv2.imread(in_fns[k0+1])[yi:yf,xi:xf]
8  img0C = cv2.imread(in_fns[k0+2])[yi:yf,xi:xf]
9  img0D = cv2.imread(in_fns[k0+3])[yi:yf,xi:xf]
10 scale_percent=40
11 width = int(img0A.shape[1] * scale_percent / 100)
12 height = int(img0A.shape[0] * scale_percent / 100)
13 dim = (width, height)
14 img00A = cv2.GaussianBlur(cv2.cvtColor(cv2.resize(img0A, dim, interpolation = cv2.INTER_AREA)
15                                                         cv2.COLOR_BGR2GRAY), (21, 21)
16 img00B = cv2.GaussianBlur(cv2.cvtColor(cv2.resize(img0B, dim, interpolation = cv2.INTER_AREA)
17                                                         cv2.COLOR_BGR2GRAY), (21, 21)
18 img00C = cv2.GaussianBlur(cv2.cvtColor(cv2.resize(img0C, dim, interpolation = cv2.INTER_AREA)
19                                                         cv2.COLOR_BGR2GRAY), (21, 21)
20 img00D = cv2.GaussianBlur(cv2.cvtColor(cv2.resize(img0D, dim, interpolation = cv2.INTER_AREA)
21                                                         cv2.COLOR_BGR2GRAY), (21, 21)
```

B. Recognition of the contours

The second part starts subtracting the different time-lapses images and defining the main contours of what moves in time. After that, the obtained curves are fitted to ellipses whose centers of mass and measures of the axes are used to find the positions of the heads.

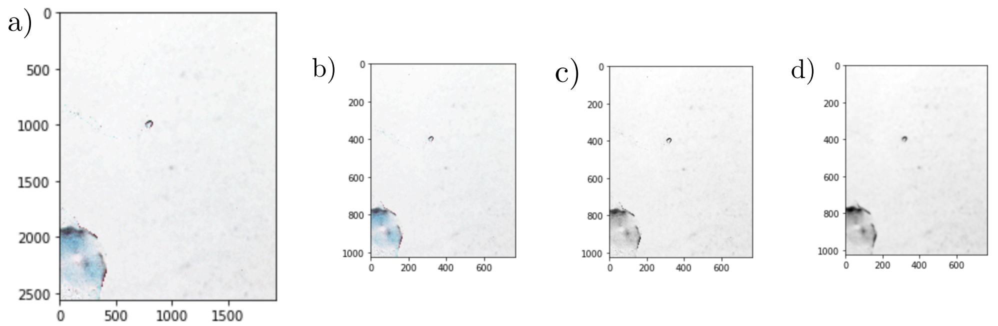


FIG. 1: Steps of adaptation of the images. a) Original image and size. b) Image resized. c) Image converted to black and white. d) Image blured.

1. Defining contours

To subtract the pictures in the time we use the function compare_ssim (lines 1-6). After that, the resulting pictures are binarized to simplify the process of contour find (lines 7-12). Figure 2 shows the result after this process.

```
1  (score, diff) = compare_ssim(img00A, img00B, full = True)
2  diff = (diff * 255).astype("uint8")
3  (score2, diff2) = compare_ssim(img00B, img00C, full = True)
4  diff2 = (diff2 * 255).astype("uint8")
5  (score3, diff3) = compare_ssim(img00C, img00D, full = True)
6  diff3 = (diff3 * 255).astype("uint8")
7  thresh = cv2.threshold(diff, 0, 255,
8            cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
9  thresh2 = cv2.threshold(diff2, 0, 255,
10           cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
11 thresh3 = cv2.threshold(diff3, 0, 255,
12           cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
```

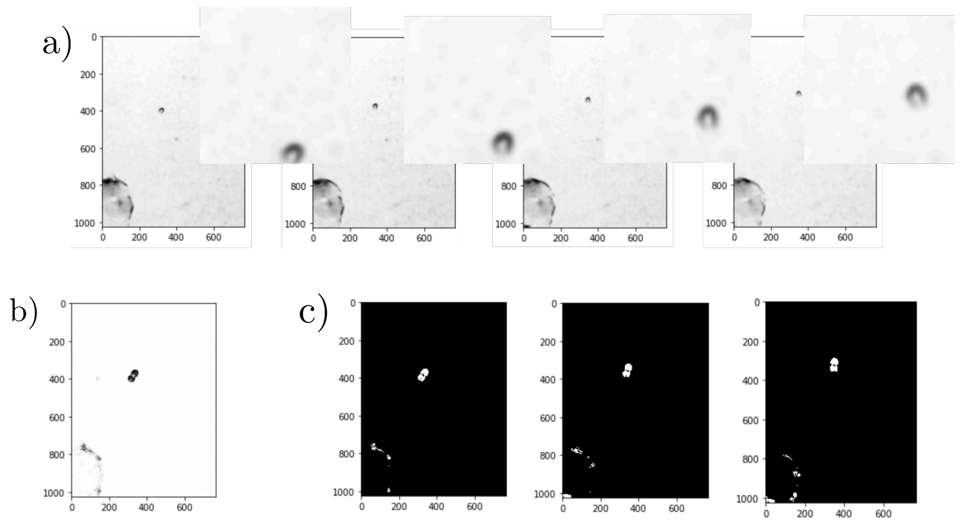


FIG. 2: Image subtraction. a) Four blurred images and the magnification to visualize the head displacement. b) Result of the subtraction of the two first pictures. c) Result of the three binarized subtraction images.

The binarized images are used to find the contours by the function findContours. To join the contours we use the code from the webpage *https://dsp.stackexchange.com/questions/2564/opencv-c-connect-nearby-contours-based-on-distance-between-them*. This code considers an additional function find_if_close defined apart (at the beginning of the program) that receives two contours and returns true if they are closer than a distance **mincontourdist**, and false otherwise.

```
mincontourdist=50;
def find_if_close(cnt1,cnt2):
    row1,row2 = cnt1.shape[0],cnt2.shape[0]
    for i in range(row1):
        for j in range(row2):
            dist = np.linalg.norm(cnt1[i]-cnt2[j])
            if abs(dist) < mincontourdist:
                return True
            elif i==row1-1 and j==row2-1:
                return False
```

After that, all the contours that are evaluated as close are arranged as a new shape with a bigger contour associated.

```
img=thresh
contours,hier = cv2.findContours(img.copy(),cv2.RETR_EXTERNAL,2)
LENGTH=len(contours)
status = np.zeros((LENGTH,1))
for i,cnt1 in enumerate(contours):
    x = i
    if i != LENGTH-1:
        for j,cnt2 in enumerate(contours[i+1:]):
            x = x+1
            dist = find_if_close(cnt1,cnt2)
            if dist == True:
                val = min(status[i],status[x])
                status[x] = status[i] = val
            else:
                if status[x]==status[i]:
                    status[x] = i+1

unified = []
maximum = int(status.max())+1
for i in range(maximum):
    pos = np.where(status==i)[0]
    if pos.size != 0:
        cont = np.vstack(contours[i] for i in pos)
        hull = cv2.convexHull(cont)
        unified.append(hull)
plt.imshow(cv2.drawContours(diff,unified,-1,(0,255,0),2))
plt.imshow(cv2.drawContours(thresh,unified,-1,255,-1),cmap='gray')
```

The last exposed code is repeated three times to find the contours associated with the differences between the first and second pictures, second and third pictures, and third and fourth pictures. For the picture shown at the beginning, we can observe in figure 3 the result after joining contours.

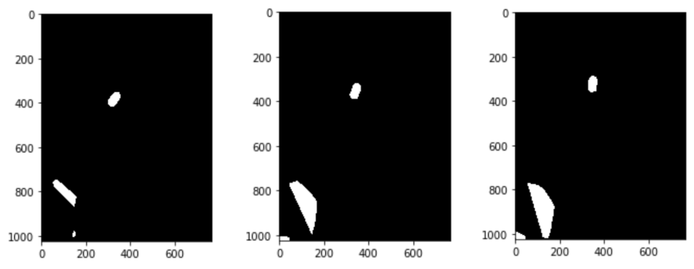


FIG. 3: Result of joined contours.

2. *Define ellipses*

Considering that Paeni's heads could be compared with ellipses, we associate each found shape to an ellipse taking saving their center of mass positions and the main features; area, and axes lengths. In this process, three filters are applied. First, it has to be considered that the definition of ellipses required six or more points in the shape contour (lines 1 and 7). Additionally, a minimum area (**areasmin**) is set in order to discard the smallest movements which probably not correspond to heads (line 9). Finally, circular shapes presumably are vortexes, therefore if the ellipse's axes have a difference smaller than a set variable **difaxes** are discard too (lines 2 and 10).

As it was mentioned, the positions of the center of mass, the areas and axes lengths are reserved in arrays positions0, areas0 and axes0 (lines 3 - 5 and 11 - 13).

```
1   areamin=5000;
2   difaxes=10;
3   positions0=[];
4   areas0=[];
5   ejes0=[];
6   for i in range(0,len(unified)):
7       if len(unified[i])>5:
8           area=np.pi*cv2.fitEllipse(unified[i])[1][0]*cv2.fitEllipse(unified[i])[1][1]
9           if area>areamin:
10              if np.absolute(cv2.fitEllipse(unified[i])[1][0]
11                         -cv2.fitEllipse(unified[i])[1][1]) > difaxes:
12                  positions0.append(cv2.fitEllipse(unified[i])[0])
13                  ejes0.append(cv2.fitEllipse(unified[i])[1])
14                  areas0.append(np.pi*cv2.fitEllipse(unified[i])[1][0]
15                                         *cv2.fitEllipse(unified[i])[1][1])
```

The presented code and classification of the shape are run for each binarized picture, and the positions, areas, and axes are reserved in arrays with the same names but the index 1 and 2. In the figure 4, it is presented the center of mass of the set ellipses in each time-lapses subtraction.

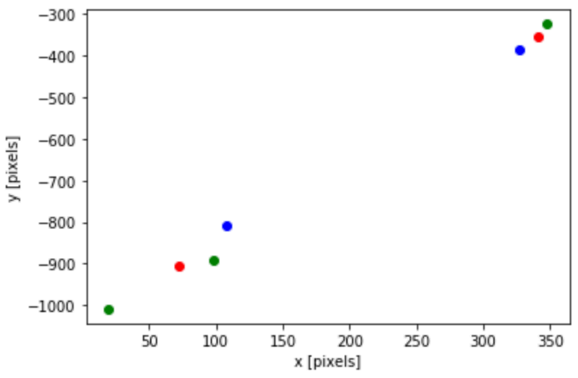


FIG. 4: The positions of ellipses centers of mass. Blue points correspond to the first subtraction ellipses, red points correspond to second subtraction, and green points correspond to third subtraction.

C. Grouping of points

To identify which of the found shapes are Paeni's head it is considered the expected persistent movement in time. Therefore, the found ellipses are grouped according to the distance of their center of mass and the overlapping. Finally, taking into account that the Paeni movement presumably is in a straight line, a linear fit is done per group in order to discard other circle movements or possible vortex.

The first step is grouping the ellipses identified in the two first subtraction pictures (the clusters are saved in the variable clst0). In this case, the criteria to consider two ellipses in the same cluster is that they are half overlapped. That is, the distance between their center of mass (dist) is higher than a minimum distance **dmin**, and at the same time shorter than the longer semi-axes of one of the ellipses (lines 8 and 9). In lines 4 and 6, the length of the semi-axes is defined. The selection is summarized in figure 5.

```
1  dmin=20;
2  clst0=[];
3  for i in range(0,len(positions0)):
4      distmin0=ejes0[i][1]/2;
5      for  j in range(0,len(positions1)):
6          distmin1=ejes1[j][1]/2;
7          dist=distance.euclidean(positions0[i],positions1[j])
8          if dmin<dist<distmin0 or dmin<dist<distmin1:
9              clst0.append([i,j])
```

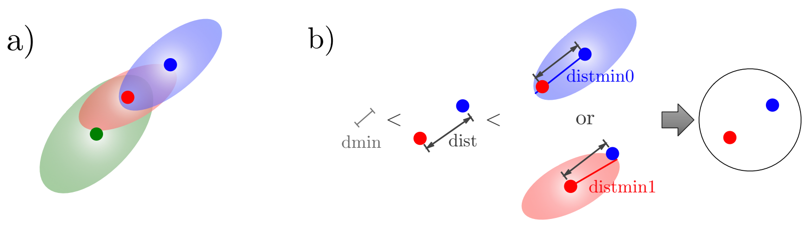


FIG. 5: a) An example of three ellipses that resulted from the subtraction of the image in the different time-lapses and their centers of mass. They are overlapped enough to form a cluster. b) A scheme to show the cases in which the two first centers of mass will be classified in the same cluster. The distance between the center of mass should be longer than dmin and shorter than one of the ellipse major semi-axes.

After that, the clusters are re-form appending an ellipse of the third image subtraction group to the clusters (clst0) previously defined. There are two cases in which an ellipse will be introduced in a previous array. The first possibility is that the ellipses that form a cluster are both overlapped with a third ellipse. Similar to the previous section, it is requested that the distance between the centers of mass of ellipses (defined in lines 7 snd 8) is longer than dmin and shorter than the major semi-axe of the new third ellipse (lines 4, 9-11). This is shown in figure 6 a).

```
1  clst1=[];
2  size1=[];
3  for i in range(0,len(positions2)):
4      distmin2=ejes2[i][1]/2;
5      for j in range(0,len(clst0)):
6          distmin1=ejes1[clst0[j][1]][1]/2;
7          dist1=distance.euclidean(positions2[i],positions0[clst0[j][0]])
8          dist2=distance.euclidean(positions2[i],positions1[clst0[j][1]])
9          if dmin<dist1<distmin2 and dmin<dist2<distmin2:
10             clst1.append([positions0[clst0[j][0]],positions1[clst0[j][1]],positions2[i]])
11             size1.append(ejes0[clst0[j][0]][0])
12         elif dmin<dist2<distmin2 and dmin<dist2<distmin1:
13             clst1.append([positions0[clst0[j][0]],positions1[clst0[j][1]],positions2[i]])
14             size1.append(ejes0[clst0[j][0]][0])
```

The other alternative is considering that the velocity in some heads could be high and the first and third ellipses (in time) are not be overlapped enough. In this case, only the distance between second and third centers of mass is considered (dist2, line 8), but different from the first cluster formation, it is requested that the distance dist2 should be longer than dmin, and shorter than both major semi-axes (lines 4, 6, 12-14). This is shown in figure 6 b).

Once the arrangements are ready, it is made a linear fitted using the three points of each group (lines 2 - 5, figure 7 a)). The error associated with this adjustment defines the last filter; since the head's movement should be persistent if the error is higher than **errorMin** (in this case is 0.9) the cluster is discarded (lines 6 - 11). The final result is given by an array called heads which contains all the positions (x, y) of the centers of mass of the first ellipses belonging to the clusters that passed all the filters (lines 12 - 19). The final output is shown in figure 7 b).

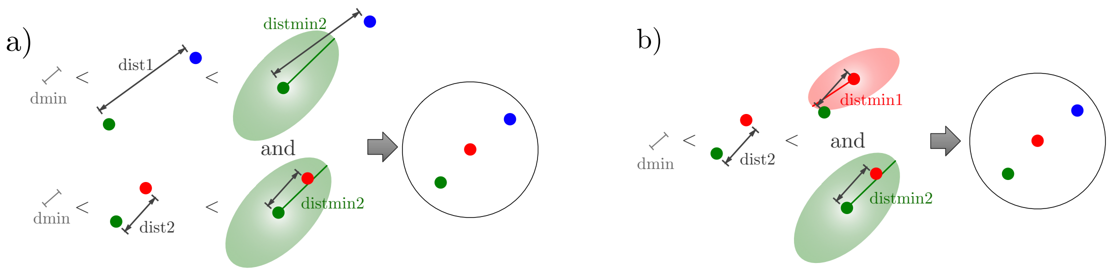
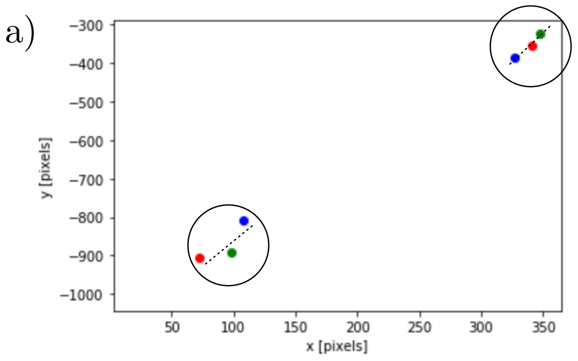


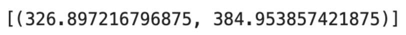
FIG. 6: The scheme shows the two alternatives to add a third point to a cluster. a) The distance between centers of mass should be shorter than the major semi-axe of the third ellipse. b) The distance between the second and third centers of mass should be shorter than the major semi-axes of the second and third ellipses.

```
error=[];
for i in range(0,len(clst1)):
    model=np.polyfit(np.transpose(clst1[i])[0],np.transpose(clst1[i])[1],1)
    predict = np.poly1d(model)
    error.append(r2_score(np.transpose(clst1[i])[1], predict(np.transpose(clst1[i])[0])))
j=[];
errorMin=0,9;
for i in range(0,len(clst1)):
    if error[i]>errorMin:
        j.append(i);
clst=[];
size=[];
for i in range(0,len(j)):
    clst.append(clst1[j[i]])
    size.append(size1[j[i]])
heads=[i[0] for i in clst];
heads
```

Here, the error errorMin could be relaxed in order to identify the heads that are turning, and eventually forming a vortex.



b)



```
[(326.897216796875, 384.953857421875)]
```

FIG. 7: a) Linear adjustment per group. b) The final output that shows heads positions (x, y) (in this case, only one).

III. SUMMARY OF FILTERS

The program considers five variables in different parts of the code that can change to make the head selection more flexible or strict. Here these filters are reviewed.

- mincontourdist=50
  The function find_if_close is used to find if two contours are close enough. If the distance in pixels is shorter than mincontourdist the function will return True, otherwise, False. Then, mincontourdist could be a smaller (higher) number to be more strict (flexible) and obtain more (less) contours smaller (bigger).

- areasmin=5000
  Before defining ellipses, all the shapes whose area is smaller than areasmin are discarded. If this variable is relaxed it is possible to identify smaller heads initially dismissed.

- difaxes=10
  It is considered that vortexes will form more circular shapes in the time. Therefore, ellipses whose axes are too similar (difference smaller than ten pixels), the contour is discarded.

- lstinlinedmin
  To form the ellipses clusters the distance between the centers of mass should be higher than a minimum distance to discard vortex. To consider heads with a specific velocity, this number could be increased (decreased) to include centers of mass farther (closer).

- errorMin
  For now, the program is adapted to identify heads that are moving in a straight line. Hence, this variable could be relaxed if it is desired to consider more circular movements.

IV. CONNECTION WITH TRACKING CODE

In order to follow Paeni's heads, it is key to have the initial box position and size. The square length is defined from the longer ellipse semi-axe (line 3), while the position, considering that the head center of mass should be in the center of the square (illustrated in figure 8 a)). Then, to initialize the tracking, the (x, y) top-left vertex of the square is determined by subtracting half of the box length to the center of mass coordinates (lines 4, 5). The result in the example is showed in figure 8 b).

```
1 nhead=0;
2 fig,ax = plt.subplots(1)
3 h,w=1*size[nhead],1*size[nhead]
4 xmin=heads[nhead][0]-(w/2)
5 ymin=heads[nhead][1]-(h/2)
6 rect = patches.Rectangle((xmin,ymin),w,h,linewidth=1,edgecolor='r',facecolor='none')
7 ax.imshow(img00A,cmap='gray')
8 ax.add_patch(rect)
9 plt.show()
```

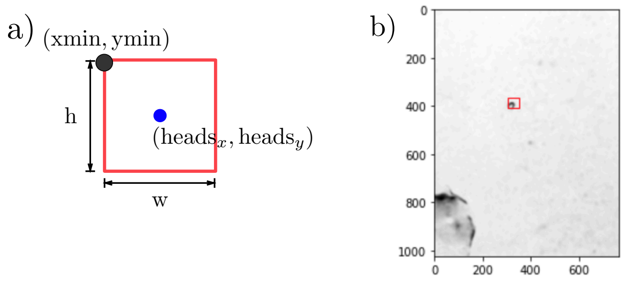


FIG. 8: a) The scheme shows the relation between the code output and the initialization tracking variables. b) Tracking initialization box.

V. RESULTS

A. Positive results

The figure 9 shows some examples where the heads are well identified.

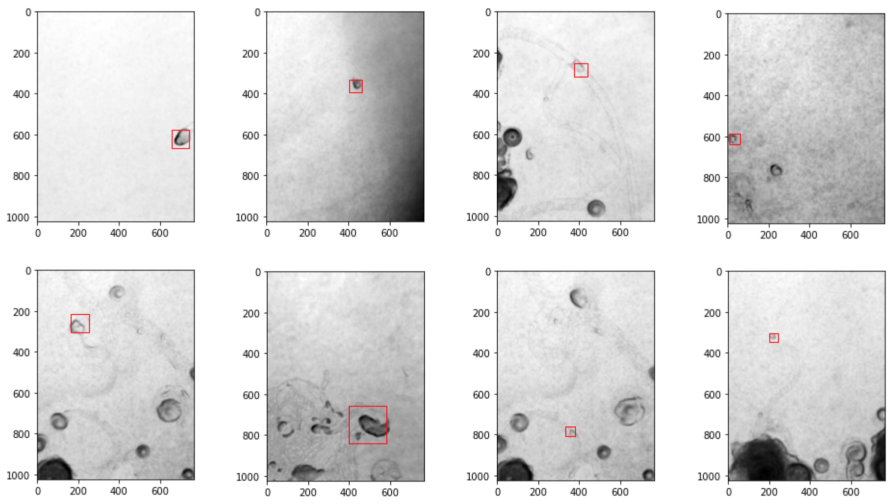


FIG. 9: Satifactory results.

B. Results to improve

In some cases, the program is not able to identify the heads. In the first picture in figure 10 there are two heads too close and the subtraction of the image is not able to distinguish them as different shapes. The other pictures show has too much noise moving which makes hard the process of identification.

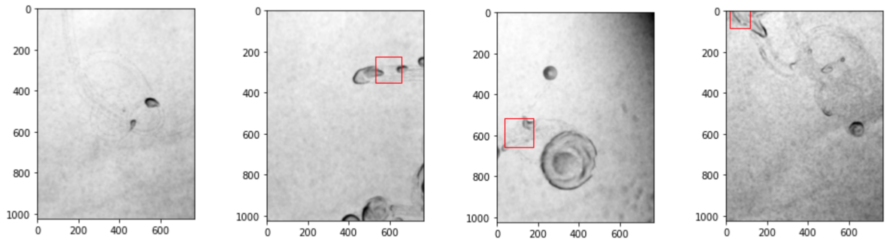


FIG. 10: Deficient results.