Informe de actividades Susana Márquez Fondecyt 1191893 etapa 2019

I. OBJECTIVE

Characterize head showed by Paenibacillus to understand if there is a relation between the size, the time after inoculation, velocity, and other variables.

II. DATA INFORMATION

The Paenibacillus pictures where taken using a Nikon and a Homescope microscope. The details of each sequence of pictures are in the file 'info.dat'. Here we can find the following information:

- **Folder**: name of the current folder. It is the original folder in the case that we want to move some pictures, we can just copy the info.dat.
- **Date**: Day and hour of the first taken picture.
- **Microscope and objective**: microscope used and lens.
- **Interval frames**: Time between loops of pictures.
- **Loop**: Number of pictures taken each time.
- **Name**:Who has inoculated the plates.
- **Inoculation Date**: Including the elapsed time from the inoculation to the first picture.
- **Medium**: Including kind and quantity of medium
- **Bacteria**: Code (PBL1001)
- **Disolution**: If it correspond (1:100 or 1:1000 from de original -80C)
- **OD**: Of the inoculation drop.
- **Temperature**: Cultivation temperature

III. TRACKING OF HEAD

In the next section, I will give details of the program the is used to follow heads. In the first part (after importing the packages) the images are imported and resized. After that, the Paeni head to follow is manually selected, which gives the parameters to initialize the tracking, implemented in the third part. Finally, taking the positions that the tracking process gives, we analyze the velocity and transform the result to the right units.

A. Importation and resized

```
in_fns = glob("./pictures/*.tif")
in_fns = sorted(in_fns)
fig,ax = plt.subplots(1)
(xi,yi,xf,yf)=(0,0,1920,2650)
#(xi,yi,xf,yf)=(0,800,1200,1700)
img0 = cv2.imread(in_fns[0])[yi:yf,xi:xf]
ax.imshow(img0)
plt.show()
scale_percent=40
width = int(img0.shape[1] * scale_percent / 100)
```

```
11 height = int(img0.shape[0] * scale_percent / 100)
12 dim = (width, height)
13 img00 = cv2.resize(img0, dim, interpolation = cv2.INTER_AREA)
14 ax.imshow(img00)
15 plt.show()
```

This section is to select the part of the image that will be used (if the head is too small, only a portion of the picture should be considered). The first two lines are to find all the image . tif in the folder 'pictures' that will be considered and order them. Then we prepare the code to show the final selection (line 3). (xi,yi,xf,yf) correspond to the initial (xi,yi) and final (xf,yf) coordinates of the selected image portion, declared on line 5, and use to define image margins on line 6, where it is read. For the fault, they are set with the width and length of Nikon pictures (1920,2650) (line 4). Additionally, in this section, line 9, the percent of rescaling is declared with the variable scale_percent. On line 13 the image is resized using the new parameters defined on lines 10, 11, and 12. Finally, the picture is showed (lines 14, 15).

After deciding the scale_percent and (xi,yi,xf,yf), the same process is done for all the images, as it is shown in the

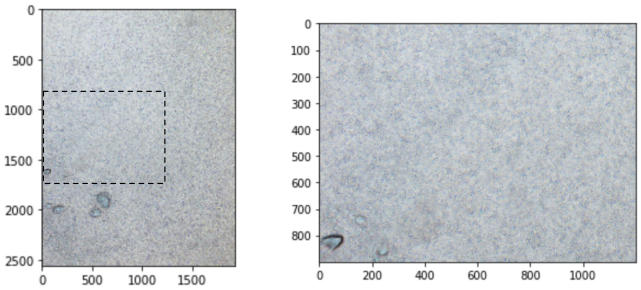

FIG. 1: Left: Example showed when (xi,yi,xf,yf) = (0,0,1920,2650) are the default numbers. The green line was drown to visualized the selected area. Right: In this case a smaller area has been choosing given by (xi,yi,xf,yf) = (0,800,1200,1700).

next lines. The main difference is on line 9 where a loop is inserted to make the same process with each picture.

```
1  imgAll=[];
2  img0 = cv2.imread(in_fns[0])[yi:yf,xi:xf]
3  scale_percent=40
4  width = int(img0.shape[1] * scale_percent / 100)
5  height = int(img0.shape[0] * scale_percent / 100)
6  dim = (width, height)
7  img00 = cv2.resize(img0, dim, interpolation = cv2.INTER_AREA)
8  imgAll.append(img00)
9  for i in range(1,len(in_fns)):
10     img0 = cv2.imread(in_fns[i])[yi:yf,xi:xf]
11     img00 = cv2.resize(img0, dim, interpolation = cv2.INTER_AREA)
12     imgAll.append(img00)
```

B. Selection of Paeni head

Considering that the code for the tracking takes as an initial point to follow a box defined by initial positions (xmin, ymin) and its width (w) and high (h), in this part manually these parameters could be selected (lines 2 - 4). To visualize the selected box, a square in red is drawn (lines 5 - 7).

```
1 fig,ax = plt.subplots(1)
2 xmin=5
3 ymin=305
4 h,w=40,40
5 rect = patches.Rectangle((xmin,ymin),w,h,linewidth=1,edgecolor='r',facecolor='none')
6 ax.imshow(imgAll[0])
7 ax.add_patch(rect)
8 plt.show()
```

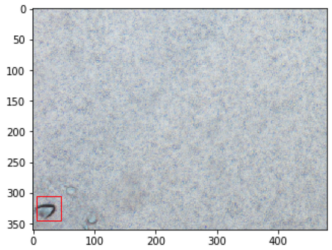

FIG. 2: The red square indicates the initial box selected for tracking.

C. Tracking

```
if __name__ == '__main__' :

    tracker_types = ['BOOSTING', 'MIL','KCF', 'TLD', 'MEDIANFLOW', 'GOTURN', 'MOSSE', 'CSRT']
    tracker_type = tracker_types[3]

    if tracker_type == 'BOOSTING':
        tracker = cv2.TrackerBoosting_create()
    if tracker_type == 'MIL':
        tracker = cv2.TrackerMIL_create()
    if tracker_type == 'KCF':
        tracker = cv2.TrackerKCF_create()
    if tracker_type == 'TLD':
        tracker = cv2.TrackerTLD_create()
    if tracker_type == 'MEDIANFLOW':
        tracker = cv2.TrackerMedianFlow_create()
    if tracker_type == 'GOTURN':
        tracker = cv2.TrackerGOTURN_create()
    if tracker_type == 'MOSSE':
        tracker = cv2.TrackerMOSSE_create()
    if tracker_type == "CSRT":
        tracker = cv2.TrackerCSRT_create()

    height , width , layers = imgAll[0].shape
    video = cv2.VideoWriter('output.avi',cv2.VideoWriter_fourcc('M', 'J', 'P', 'G'), 20, (widt

    Data = np.zeros([len(imgAll),2])

    img = cv2.cvtColor(imgAll[0], cv2.COLOR_BGR2GRAY)
    frame = cv2.GaussianBlur(img, (21, 21), 0)

    # Initialize tracker with first frame and bounding box
     bbox = (xmin, ymin, w, h)
    #bbox = cv2.selectROI(frame, False)
    ok = tracker.init(frame, bbox)
    Data[0]=bbox[0]+bbox[2]/2,bbox[1]+bbox[3]/2
    p1 = (int(bbox[0]), int(bbox[1]))
    p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
    cv2.rectangle(imgAll[0], p1, p2, (0,255,0), 2, 1)
    print(0,p1,p2)
    video.write(imgAll[0])

    for m in range(1,len(imgAll)):
        img = cv2.cvtColor(imgAll[m], cv2.COLOR_BGR2GRAY)
```

```
44          frame = cv2.GaussianBlur(img, (21, 21), 0)
45          ok, bbox = tracker.update(frame)
46          if ok:
47              # Tracking success
48              p1 = (int(bbox[0]), int(bbox[1]))
49              p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
50              #cv2.rectangle(frame, p1, p2, (255,0,0), 2, 1)
51          else :
52               #Tracking failure
53              print('Tracking failure detected')
54              np.savez('data', Data)
55              video.release()
56              sys.exit()
57          #assign final value
58          Data[m] = np.array([(p1[0]+p2[0])/2,(p1[1]+p2[1])/2])
59          cv2.rectangle(imgAll[m], p1, p2, (0,255,0), 2, 1)
60          print(m,p1,p2)
61          video.write(imgAll[m])
62
63      np.savez('data', Data)
64      video.release()
```

The tracking was adapted from the code proposed on this webpage. Four main parts could be identified: the selection of tracking type (lines 3 - 21), the tracking video initialization (lines 23, 24), the first step and box declaration (lines 31 - 40), and the tracking loop including the movie update (lines 42 - 64). Each part is detailed below.

- **Selection of tracking part**
  Depending on the kind of objected to study, different types of tracking where designed using different mechanisms. The details about each can be found on the original webpage. For Paeni heads the best results are found using type 2 (KCF, Kernelized Correlation Filters) or 3 (TLD, Tracking, learning and detection). This information is specified on line 4.

- **Video initialization**
  To create the video with the tracking we use the function cv2.VideoWriter. To initialize it, the heigh, width, and layers are defined (line 23) taking as a reference the first read picture shape. Additionally, on line 26, an empty array, Data, is initialized to save the center point of the tracking box for each frame, which corresponds to the whole head movement.

- **First step and box declaration**
  On line 32 the parameters declared in the section 'Selection of Paeni head', are used to declare the box (line 32) and, with the initial frame, initialize the tracker (line 34). If ok is true, the tracker started successful. After each tracker process step, including this, the first one, two actualizations have to be done. On the one hand, the center of the box should be saved in Data (line 35). To calculate the center we use the xmin,ymin,width and high box, corresponding to bbox[0], bbox[1], bbox[2] and bbox[3] respectively. Therefore, the x middle point corresponds to:

$$\text{xmin} + \frac{\text{width}}{2} = \text{bbox[0]+bbox[2]/2},$$

  and the y middle point corresponds to:

$$\text{ymin} + \frac{\text{high}}{2} = \text{bbox[1]+bbox[3]/2}.$$

  Additionally, the video should be updated to save the new frame including the tracking box. To draw the last one the p1=(xmin, ymin) and p2=(xmax, ymax) are required. p1 = (bbox[0],bbox[1]) is known, while p2 have to be defined by p2 = (bbox[0] + bbox[2], bbox[1] + bbox[3]) due xmax = xmin + width, and ymax = ymin + high (lines 36 and 37). In line 38 the rectangle is drawn, and the information is updated in video.write in line 40. Finally, to make the tracking easier, all the pictures are passed to grayscale and Gaussian blur filter (lines 28 and 29).

- **Tracking Loop**
  In this section basically the same steps are repeated for each frame: passing the picture by the filters (lines 43 and 44), update the tracker (line 45), the Data array (line 58), and the video (lines 48, 49, 59 and 61). When the loop finishes the video is released (line 64). If the tracking fails, ok will be false, the video will be released, and the loop will be broken (lines 51 - 56).

D. Velocity analysis

To visualize the movement Data can be plotted using matplot.

```
1  Data2=Data[1:19]
2  plt.title('Tracking')
3  plt.xlabel('x')
4  plt.ylabel('y')
5  plt.plot(np.transpose(Data2)[0],-np.transpose(Data2)[1],'o')
```

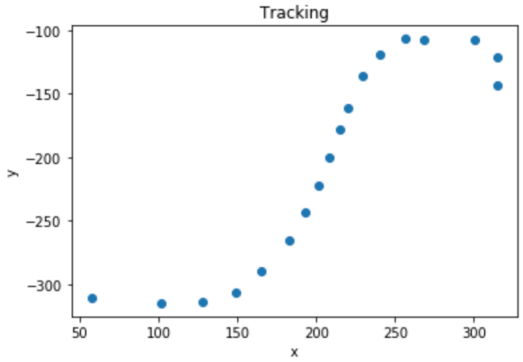

FIG. 3: Matplot results. The points show the spatial head trajectory saved in Data. The positions are not resized.

The velocity is calculated dividing the displacement of the head by the time step, which in most of the cases corresponds to 2 minutes. However, the displacement given by the tracking has to be resized due to the initial image resized and to consider the pixels length scale.

```
1  vel=[]
2  scale=105
3  for i in range(1,len(Data2)-1):
4      dist=distance.euclidean((Data2[i][0],Data2[i][1]),(Data2[i+1][0],Data2[i+1][1]))
5      dist2=dist*(100000/(2*scale*scale_percent))
6      vel.append(dist2)
7  plt.ylim(0, 500)
8  plt.plot(vel)
9  plt.show()
10 np.mean(vel),np.std(vel)
```

At the beginning scale_percent reduce the image size in a 40%, then the displacement should be multiply by 100/ scale_percent to recover the original size. Additionally, in the code, scale is the number of pixels that represent 1000 µm, therefore the displacement should also multiplied by 1000/scale. The complete resized process is summarized in image 4. In the code, for each pair of points the euclidean distance is calculated (line 4). In the line 5 the found distance is multiply by the factor(100000/(2*scale*scale_percent)), the two is due to the time step. The lines 7 - 9 are to plot velocity in time graphic. Line 10 gives the mean velocity.

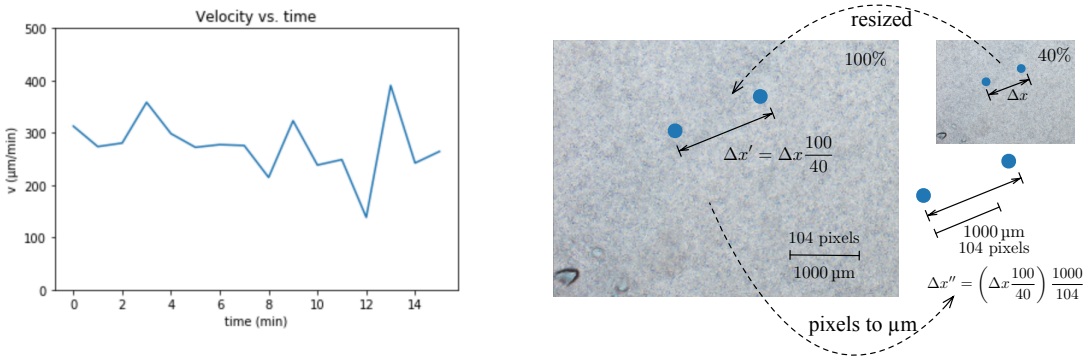

FIG. 4: Left: Velocity in time graphic. Right: Summary of process to resize the velocity and find it in units µm.

IV. IMAGE RESIZED

```
1  readPict = glob("./pictures/*.tif")
2  readPict = sorted(readPict)
3  scale_percent=30
4  #first picture
5  im=cv2.imread(readPict[0])
6  imGray=cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
7  width = int(imGray.shape[1] * scale_percent / 100)
8  height = int(imGray.shape[0] * scale_percent / 100)
9  dim = (width, height)
10 imGrayS = cv2.resize(imGray, dim, interpolation = cv2.INTER_AREA)
11 cv2.imwrite('picture0.png',imGrayS)
12 for i in range(0,len(readPict)):
13     im=cv2.imread(readPict[i])
14     imGray=cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
15     imGrayS = cv2.resize(imGray, dim, interpolation = cv2.INTER_AREA)
16     cv2.imwrite("picture"+str(i)+".png",imGrayS)
```

To upload the pictures to the Drive, scale_percent=30. The code read each picture from a folder (lines 5, 13) and passes them to grayscale (lines 6, 14). The parameters to resized pictures are taken from the first picture (lines 7 - 9), and the resized using the Opencv function 'resized' (lines 10, 15).