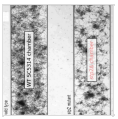

Feb 03, 2020

Bioflux Analyses: Image Preprocessing

In 1 collection

Tobias Weise[1]

[1]BioControl Jena GmbH

1 Works for me dx.doi.org/10.17504/protocols.io.bak3icyn

Tobias Weise
BioControl Jena GmbH

ABSTRACT

Biofilm formation under shear flow conditions was monitored using the Bioflux1000 device (Fluxion Biosciences, Inc.). In short, Candida albicans overnight cultures were washed in pre-warmed RPMI medium. Cells were seeded for 2-5 sec from the outlet well into the channels of Bioflux1000 flow chambers, which were primed before with warm medium. The cells were allowed to adhere to the channels for 90 min without any flow, followed by removal of non-adherent cells by flowing fresh, pre-warmed RPMI medium for 5 sec. Shear flow was set for time series experiments over 24 h biofilm formation and images were captured every 20 min. Two channels were investigated in parallel having a 10 × magnification to allow a direct comparison between a mutant and a reference (wild-type) strain. Image capturing and stacks to movies was performed using the MetaMorph® Software (Molecular Devices).

Source material provided as AVI files was converted into single TIFF images as well as data frames containing meta data annotations. The individual image contains two growth chambers (wild type and mutant) separated by four edge lines. Images were rotated automatically to vertical alignment in order to carry out an automated chamber detection and analysis. The mean pixel intensity (i. e. grey scale value; reflecting cell density) of the individual chamber was calculated and added into the respective data frame.

All computations were performed using the programming language python (version 3.6.9) and the additional packages numpy (version 1.16.2), opencv-python (version 4.1.1.26), pandas (version 0.25.0) and scikit-image (version 0.15.0).

Video File to Frames

1 Convert video file into single images

The source material provided as .AVI file is converted into single .TIFF images using *opencv-python.* Additional, a data frame containing meta data annotations is created.

1.1 Import of the required packages.

```
import numpy as np
import pandas as pd
import cv2
```

1.2 Example code for reading the video file from source path and saving the individual images to target path.

```
cap = cv2.VideoCapture(source_path/video_file.AVI)
i = 0

# read frames from video file
while(cap.isOpened()):

    ret,frame = cap.read()

    if ret == False:
        break

    # convert to greyscale image
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # save to file
    cv2.imwrite((target_path + 'frame_%s.TIFF'%i), frame)

    i += 1

cap.release()
cv2.destroyAllWindows()
```

1.3 Example output of individual image (2D array) .

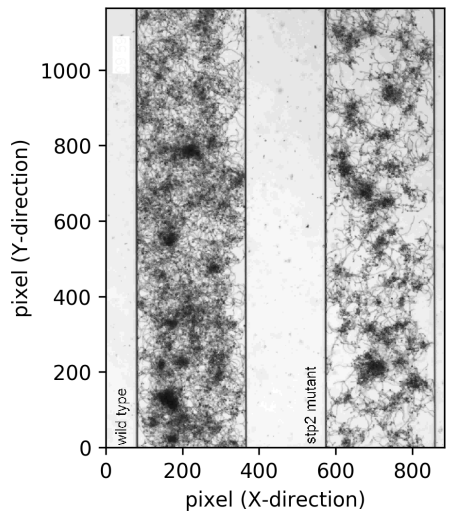

Fig. 1.1 Example of Single Video Image.

Image Processing

2 Automated Image Rotation

Image processing required vertical aligned chamber edges (Fig. 2.1-A). Unaligned video frames would result in peak values scattered in X-direction (conferring to Fig. 2.1-C and D). In order to gain vertical alignment, the rotation angle is estimated by minimising the peak width in X-direction by using a basin-hopping algorithm (scipy.optimize.basinhopping). Standard settings of the package are applied with respect to the convergence criteria. Initial rotation angle is set to 0° for all frames.

2.1 Import of the required packages.

```
import numpy as np
import pandas as pd
import cv2
import skimage
import scipy
```

2.2 Example code of the Automated Image Rotation definition.

Strategy:

1. calculate the median pixel intensity along the y-axis of image
2. create derivative of median values (absolute values) to find the change in median pixel intensity along y-axis of image
3. select peak values (representing position of chamber lines / peak width correlates to slope of chamber line)
4. rotate image to minimise the peak width

```
def score(a):

    # rotate image by given angle 'a'
    img_rot = skimage.transform.rotate(img,a,resize=True)

    # calculate median of pixel intensity along the y-axis of image
    xmedian = np.zeros(len(img_rot))
    for i in range(len(img_rot)):
        xmedian[i] = np.median(img_rot[i])

    # add median values to data frame
    df = pd.DataFrame(data={'xmedian':xmedian})

    # create derivative along the median values (absolute values)
    # 'xdiff' represents change in median pixel intensity along y-axis of image
    df['xdiff'] = pd.DataFrame.diff(df.xmedian, periods=1, axis=0).abs()

    # select 'xdiff' values >= 98% percentile  to new data frame
    qdf = df[df.xdiff >= np.nanpercentile(df.xdiff,98)]
    qdf = qdf.reset_index()
    qdf.rename(columns={'index':'imgdex'},inplace=True)

    # cluster selected xdiff values into 4 clusters
    # (representing the position of camber lines in y-direction)
    qdf['cluster'] = pd.cut(qdf.imgdex,4,right=True,labels=[0,1,2,3])

    # calculating the width of cluster
    imgdex = np.zeros(4)
    for i in range(4):
        imgdex[i] = qdf.imgdex[qdf.cluster == i].max() - \
                    qdf.imgdex[qdf.cluster == i].min()
    imgdex = np.array(imgdex, dtype='float')

    # returning sum of cluster width
    return imgdex.sum()
```

2.3 Example code of Automated Image Rotation execution.

```
# load meta data frame
data = pd.read_pickle((data_path + 'data.pkl'))

# loop over all images
angle = np.zeros(len(data))
for i in range(len(data)):

    # read respective image and invert
    img =  255 - skimage.io.imread((source_path + 'frame_%s.TIF'%i))

    # estimate rotation angle using basinhopping initial rotation angle = 0°
    answ = scipy.optimize.basinhopping(score,0,niter=200)
    angle[i] = answ.x[0]

# append angles to meta data frame + save data frame to file
data['angle'] = angle
data.to_pickle((data_path + 'data.pkl'))
```

3 Chamber Edge Determination

The individual image contains two growth chambers separated by four edge lines (Fig. 2.1-A). Median pixel intensity (in Y-direction) is calculated for each position in X-direction (Fig 2.1-B), and subsequently derived into the absolute difference in median pixel intensity (Fig. 2.1-C). Values greater or equal to the 98 % percentile of the respective data set (peak values) are assigned into four equidistant clusters (Fig. 2.1-D). The maximum peak value of the respective cluster returns the X-position of the respective chamber edge.

Visualisation of Chamber Edge Determination.

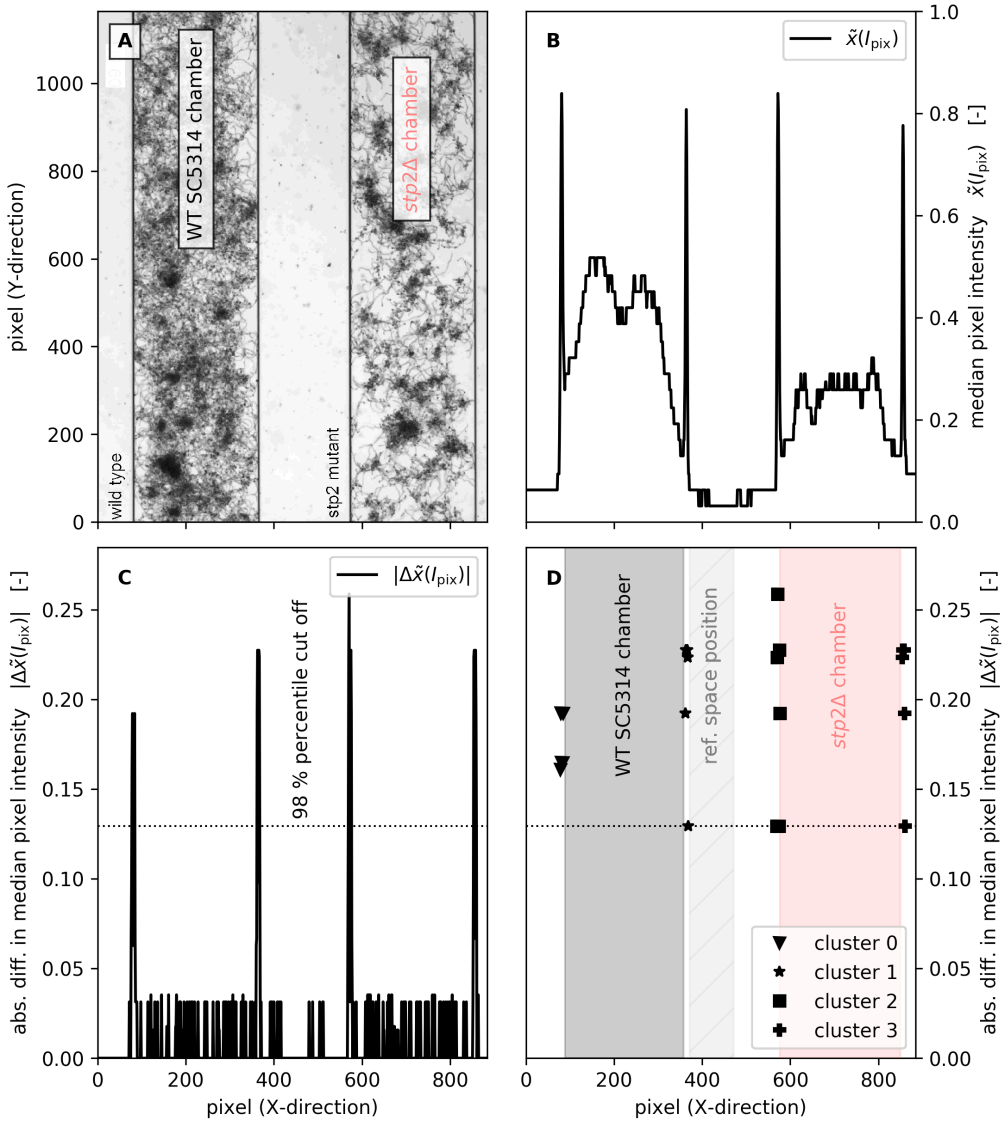

Fig. 2.1 Visualisation of Chamber Edge Determination; A: Example video image; B: median pixel intensity along Y-direction; C: Absolutes of difference in median pixel intensity; D: Final assignment of chambers and reference space (in X-direction) based on the maximum peak value of the respective cluster.

3.1 Import of the required packages.

```
import numpy as np
import pandas as pd
import skimage
```

3.2 Example code of the Chamber Edge Determination definition.

Strategy:

1. calculate the median pixel intensity along the y-axis of image
2. create derivative of median values (absolute values) to find the change in median pixel intensity along x-axis of image
3. select peak values (representing position of chamber lines)
4. calculation of 6 positions (left chamber (2 lines), inner space (2 lines), right chamber (2 lines))

```
def find_lines(img):

    #invert image
    img_bin = 255 - img

    # calculate median intensity row-wise
    xmedian = np.zeros(len(img_bin))

    for i in range(len(img_bin)):
        xmedian[i] = np.median(img_bin[i])

    # median values to DataFrame
    df = pd.DataFrame(data={'xmedian':xmedian})

    # calculate absolutes of differences of pixels row-wise
    df['xdiff'] = pd.DataFrame.diff(df.xmedian, periods=1, axis=0).abs()

    # calculate 98% percentile & pic values above this threshold
    qdf = df[df.xdiff >= np.nanpercentile(df.xdiff,98)].copy()
    qdf.reset_index(inplace=True)
    qdf.rename(columns={'index':'imgdex'},inplace=True)

    # assign imgdex to four clusters
    qdf['cluster'] = pd.cut(qdf.imgdex,4,right=True,labels=[0,1,2,3])

    # assign 6 lines (using 5 pixels safety distance)
    imgdex = np.zeros(6)

    imgdex[0] = qdf.imgdex[(qdf.cluster == 0) & \
    (qdf.xdiff == qdf.xdiff[(qdf.cluster == 0)].max())].max() + 5
    imgdex[1] = qdf.imgdex[(qdf.cluster == 1) & \
    (qdf.xdiff == qdf.xdiff[(qdf.cluster == 1)].max())].min() - 5

    imgdex[2] = qdf.imgdex[(qdf.cluster == 1) & \
    (qdf.xdiff == qdf.xdiff[(qdf.cluster == 1)].max())].max() + 5
    imgdex[3] = qdf.imgdex[(qdf.cluster == 2) & \
    (qdf.xdiff == qdf.xdiff[(qdf.cluster == 2)].max())].min() - 5

    imgdex[4] = qdf.imgdex[(qdf.cluster == 2) & \
    (qdf.xdiff == qdf.xdiff[(qdf.cluster == 2)].max())].max() + 5
    imgdex[5] = qdf.imgdex[(qdf.cluster == 3) & \
    (qdf.xdiff == qdf.xdiff[(qdf.cluster == 3)].max())].min() - 5

    imgdex = np.array(imgdex, dtype='int')

    # return lines as integer array
    return imgdex
```

3.3 Example code of the Chamber Edge Determination execution.

```
# load meta data frame
data = pd.read_pickle((data_path + 'data.pkl'))

# loop over all images
line1, line2, line3, line4, line5, line6 = [np.zeros(len(data)) for i in range(6)]

for i in range(len(data)):
    clear_output()

    # read respective image
    img = skimage.io.imread((source_path + 'frame_%s.TIFF'%i))

    # rotate image
    img = skimage.transform.rotate(img,data.angle.median(),\
                                   resize=True,mode='constant',cval=1)
    # find lines
    lines = find_lines(img)

    line1[i], line2[i], line3[i], line4[i], line5[i], line6[i] = lines

# append positions to meta data frame
data['line1'] = line1
data['line2'] = line2
data['line3'] = line3
data['line4'] = line4
data['line5'] = line5
data['line6'] = line6

# convert values into integers
data.line1 = pd.to_numeric(data.line1, errors='raise', downcast='integer')
data.line2 = pd.to_numeric(data.line2, errors='raise', downcast='integer')
data.line3 = pd.to_numeric(data.line3, errors='raise', downcast='integer')
data.line4 = pd.to_numeric(data.line4, errors='raise', downcast='integer')
data.line5 = pd.to_numeric(data.line5, errors='raise', downcast='integer')
data.line6 = pd.to_numeric(data.line6, errors='raise', downcast='integer')

save meta data frame to file
data.to_pickle((data_path + 'data.pkl'))
```

4 Chamber Selection and Background Removal

Chambers were selected from the respective video frame between the determined chamber edges (according to Fig. 2.1-D). In order to exclude the chamber edge itself, a safety distance of 5 pixels in X-direction is applied. Also, a background removal was performed by subtracting the median intensity of the reference space resolved in Y-direction. Reference was selected as a 100 pixel wide strip next to cluster 1 (Fig. 2.1-D).

4.1 Import of the required packages.

```
import numpy as np
import pandas as pd
import skimage
from skimage import img_as_uint
import cv2
```

4.2 Example code of image rotation and chamber cutting execution.

```
# load meta data frame from file
data = pd.read_pickle((data_path + 'data.pkl'))

# loop over all images
for i in range(len(data)):

    clear_output()
    print('processing file ',i+1,' of ',len(data))

    # read respective image
    img = skimage.io.imread((source_path + 'frame_%s.TIFF'%i))

    # rotate image
    img = skimage.transform.rotate(img,data.angle.median(),resize=False)

    # cut image into desired cambers
    # applying 10 pixels as safety distance in x-direction
    img_cham_A = img[data.line1[i]:data.line2[i], 10:len(img[0])-10]
    img_cham_B = img[data.line5[i]:data.line6[i], 10:len(img[0])-10]

    # save images to folder
    skimage.io.imsave((target_path + '00_WT/frame_%s_WT.TIFF'%i), \
                      img_as_uint(img_cham_A))
    skimage.io.imsave((target_path + '01_MT/frame_%s_MT.TIFF'%i), \
                      img_as_uint(img_cham_B))
```

4.3 Example code of background removal execution.

```
# load meta data frame from file
data = pd.read_pickle((data_path + 'data.pkl'))

# loop over all images
for i in range(len(data)):

    clear_output()
    print('sample',k+1,'processing file ',i+1,' of ',len(data))

    # read respective image
    # read + rotate original image
    img = cv2.imread((source_path + 'frame_%s.TIFF'%i),0)
    img = skimage.transform.rotate(img,data.angle.median(),resize=False)
    img = img_as_ubyte(img)

    # invert image
    img = 255 - img

    # read chamber images
    img_cham_A = cv2.imread((source_path + '00_WT/frame_%s_WT.TIFF'%i),0)
    img_cham_B = cv2.imread((source_path + '01_MT/frame_%s_MT.TIFF'%i),0)

    # invert chamber images
    img_cham_A = 255 - img_cham_A
    img_cham_B = 255 - img_cham_B

    # rotate chamber images 90°
    img_cham_A = np.array([list(i) for i in zip(*img_cham_A)], dtype=np.int16)
    img_cham_B = np.array([list(i) for i in zip(*img_cham_B)], dtype=np.int16)

    # cut reference space (+ invert)
    refspace = img[data.line3[i]:data.line3[i]+100, 10:len(img[0])-10]

    # rotate refspace 90°
    refspace90 = np.array([list(i) for i in zip(*refspace)], dtype=np.int16)

    # calculate median intensity of background
    background = np.zeros(len(refspace90))
    for j in range(len(refspace90)):
        background[j] = np.median(refspace90[j])
    background = np.array(background,dtype=np.int16)

    # create new empty image
    cham_A_new = np.zeros(img_cham_A.shape, dtype=np.int16)
    cham_B_new = np.zeros(img_cham_B.shape, dtype=np.int16)

    # subtract background
    for l in range(len(background)):
        cham_A_new[l] = img_cham_A[l] - background[l]
        cham_B_new[l] = img_cham_B[l] - background[l]

    # rotate and invert new images back to 0°
    cham_A_new = 255 - np.array([list(i) for i in zip(*cham_A_new)], \
                                dtype=np.int16).clip(0, 255)
    cham_B_new = 255 - np.array([list(i) for i in zip(*cham_B_new)], \
                                dtype=np.int16).clip(0, 255)

    # save images to folder
    cv2.imwrite((target_path[k] + '00_WT/frame_%s_WT.TIFF'%i),cham_A_new)
    cv2.imwrite((target_path[k] + '01_MT/frame_%s_MT.TIFF'%i),cham_B_new)
```

Calculation of Pixel Intensity

5 The mean pixel intensity I_{pix} (i. e. gray scale value; reflecting cell density) for each of the two chambers is calculated. The values are added into the respective data frame. Growth rates μ are derived from I_{pix} using the central difference approximation below.

$$\mu(t_i) = \frac{I_{pix}(t_{i+1}) - I_{pix}(t_{i-1})}{(t_{i+1} - t_{i-1}) \cdot I_{pix}(t_i)}$$

Legend: $\mu(t_i)$ - growth rate μ at time point t_i, $I_{pix}(t_i)$ - mean pixel intensity at time point t_i, i - i^{th} measurement

5.1 Import of the required packages.

```
import numpy as np
import pandas as pd
import skimage
from skimage import img_as_float
```

5.2 Example code of Mean Pixel Intensity Calculation.

```
data = pd.read_pickle((data_path + 'data.pkl'))

# create time column (within this batch: measurements every 20min -> index / 3)
data['time'] = data.index / 3 # [h]

meanGrayWT, meanGrayMT, \
varGrayWT, varGrayMT = [np.zeros(len(data)) for i in range(4)]

for k in range(len(data)):

    imgWT = 1 - img_as_float(skimage.io.imread((source_path + \
                                                '00_WT/frame_%s_WT.TIF'%k)))
    imgMT = 1 - img_as_float(skimage.io.imread((source_path + \
                                                '01_MT/frame_%s_MT.TIF'%k)))

    meanGrayWT[k] = np.mean(imgWT)
    meanGrayMT[k] = np.mean(imgMT)

data['greyWTint'] = meanGrayWT
data['greyMTint'] = meanGrayMT

data['greyWTmu'] = np.gradient(data.greyWTint) / data.greyWTint * 3
data['greyMTmu'] = np.gradient(data.greyMTint) / data.greyMTint * 3

data.to_pickle((data_path + 'data.pkl'))
```

5.3 Visualisation of the calculated data.

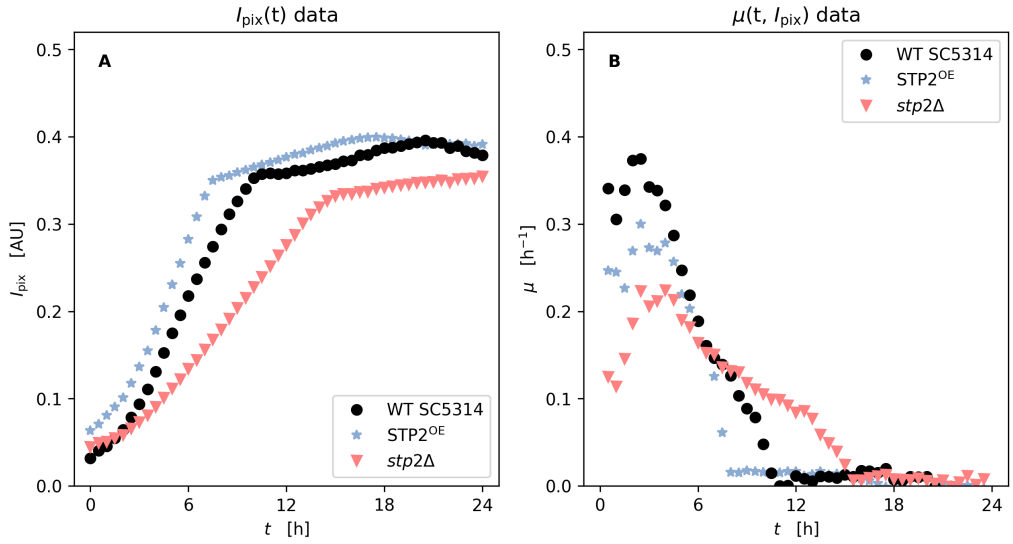

Fig. 3.1 Visualisation of the data calculated for wild type strain (black) and mutant strains (red/blue) from the individual images; A: calculated mean pixel intensities I_{pix}; B: calculated growth rate µ.