The report: Bacteria detection SimpleGUI

Introduction to Digital Image Processing (LTTI.00.019)

Prepared by: Aleksandra Shabanova

May 20, 2021

FINAL PROJECT

General description

Bacteria are widely used microorganisms both in science and industry. A bacterial cell is a simple model that can be used to produce food products, antibiotics, vaccines, fuels, and other goods by utilizing the bacterial natural capabilities or genetically altering them. Thus, many scientists look for tools to optimize research to rival a fast-growing field. And Image processing gives a helping hand allowing the creation of an interface for fast detection and quantifying of bacterial cells from the image.

Used python packages

The interface for bacterial detection was done using SimpleGUI, a Python package allowing creation of a simple and user-friendly GUI. To implement the image processing modifications OpenCV, a library of functions for real-time computer vision, and NumPy, a python-based library for the handling of arrays and matrices, were used. While to show the plots matplotlib.pyplot was applied.

Graphical interface design

The design of the interface based on the pySimpleGUI tutorial. It allowed the creation of basic-user-interface elements along with matplotlib integration and the computer vision functions (see Figure 1). To resize the window of GUI one can use the following line in the code: *window = sg.Window("Bacteria detection", layout, location=(x, y)),* where x is the width of the window, y - the height. The initial dimensions are 800x400.

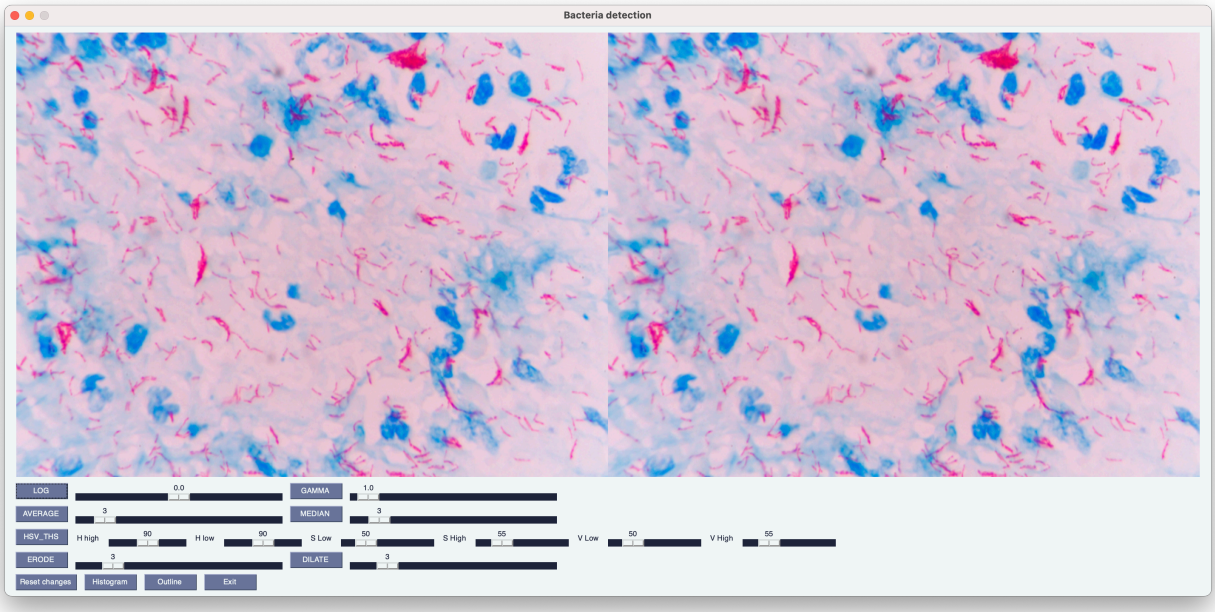

Figure 1: The starting window with two images, where changes are applied on the left image, while the right one is the initial image. In the lower-left corner, all possible functions are shown with their corresponding buttons and, if required, sliders.

General code logic

The code is composed of two parts. The first one is defying the layout. Pressing of buttons calls the event, while sliders allow defining the values for functions within the event (see Figure 1).The second part is the events

themselves. Initially, before the event loop, the image is read, and the user can choose the file by typing its name in *img = cv2.imread('your_filename')*. The image is copied to two variables, **img_tmp** and **image**. The former is the image that can be modified and the letter – the default image.
In the last part, a graphical user interface needs to run inside a loop that waits for the user's action. In our case, the user can apply 11 events:

Image enhancement functions

1. *Log* – a logarithmic transformation of the image where the low grey value portion of the image is partially expanded and the high grey value portion is partially compressed. This allows emphasis on the low grayscale portion of the image. The log transformations can be defined by this formula **s = c log(r + 1)**, where **s** and **r** are the pixel values of the output and the input image. **c** is a constant. The value 1 is added to each of the input image pixel value as, in the case of 0-pixel intensity, log(0) is equal to infinity. Thus, the addition of 1 sets the minimum value to 1. The value of c in the log transform adjust the kind of enhancement you are looking for (see Figure 1). (Source)

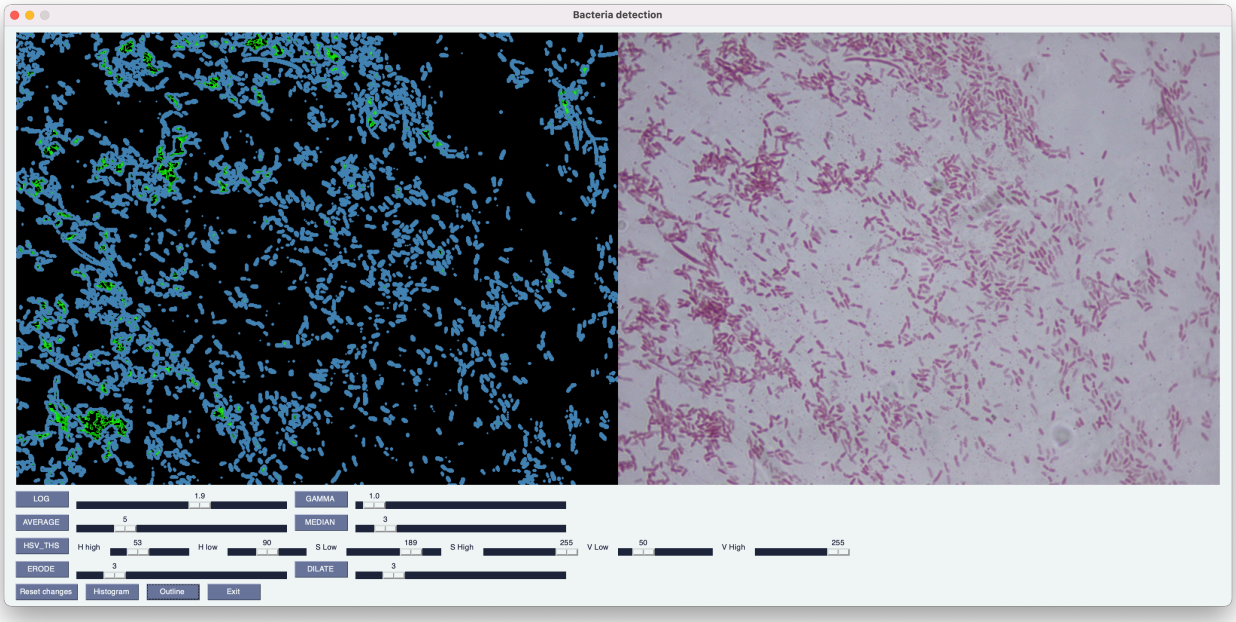

Figure 2: Example of how logarithmic transformation could be applied for better bacteria detection. The constant of 1.9 used to get bacteria in green color and then a thresholded mask created with the outline for bacteria shapes.

2. *Gamma* – Gamma correction or Power Law Transform. Firstly, image pixel intensities are shifted from the range [0, 255] to [0, 1.0], from where the gamma-corrected image is received by applying the formula **O = I ^ G**. I is the scaled input image's pixel values, **G** – gamma value and **O** – output image's values. **G > 1** will shift the image towards the darker end of the spectrum, while values < 1 will make the image appear lighter (see Figure 3).

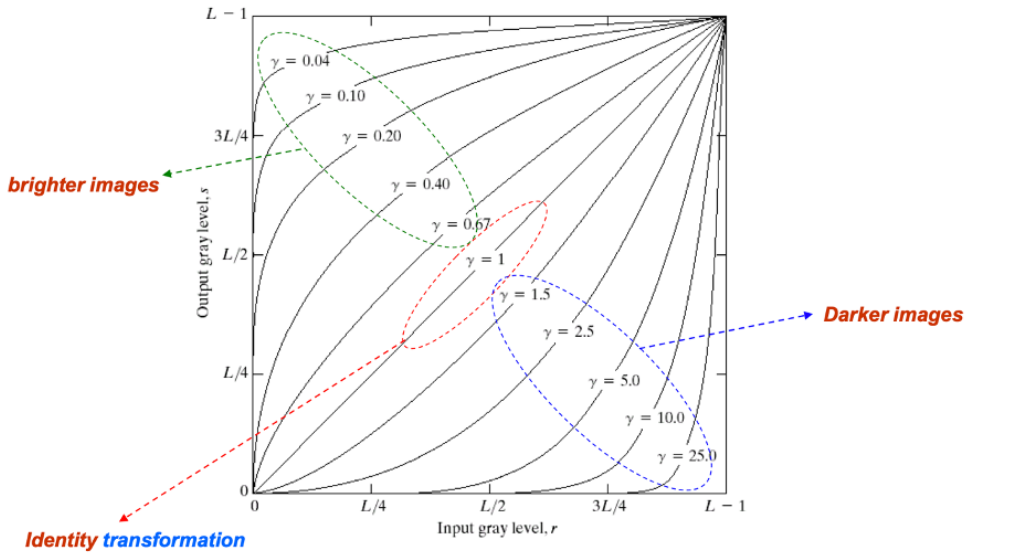

Figure 3: The affect of gamma constant value on the brightness of image. (Source)

Image Restoration filters

3. *Average* - a filter that helps to smooth the local variations in the image and to reduce the noise as a result of blurring. To implement it, OpenCV function cv2.blur() was used which convolves the image with a filter with odd dimensions. It simply takes the average of all the pixels under the kernel area and replaces the central element with this average.
4. *Median* - a filter that provides less blurring than the other linear smoothing filters, as well as high effectiveness in Salt&Pepper noise removal. The function cv2.medianBlur() was used that computes the median of all the pixels under the kernel with odd dimensions and the central pixel is replaced with this median value.

Thresholding

5. HSV_THS - the thresholding of image by applying inRange function where the lower and higher boundaries for H, S, V can be chosen by user (see Figure 4). It allows to create black and white mask, where the thresholded values are white, and background - black. In our case, the color of bacteria is saved by applying this mask to the initial image.

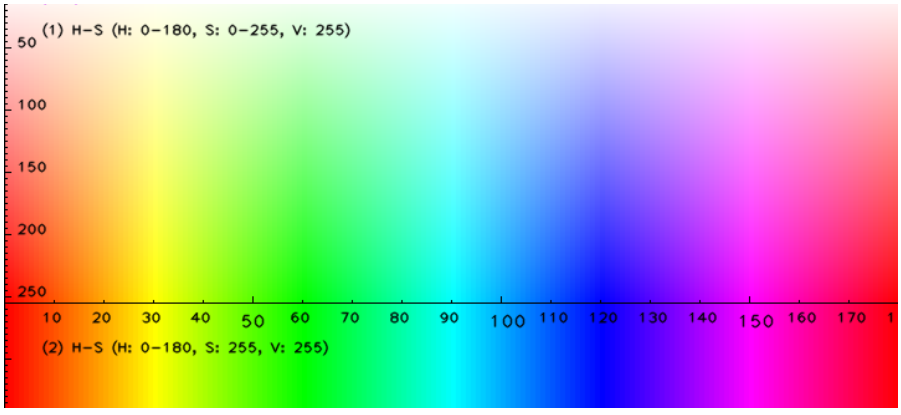

Figure 4: The H,S,V values could be chosen from OpenCV HSV color map.

Morphological Image Processing functions

6. *Erode* - allows removing pixels on object boundaries by convolving the image with the square kernel with odd dimensions. The number of added pixels is defined by the kernel size that can be chosen by the user.
7. *Dilate* - allows adding pixels on object boundaries by convolving the image with the square kernel with odd dimensions. The number of added pixels is defined by the kernel size, which can be chosen by the user.

Note: Erosion followed by dilation helps to perform filtering.

Other functions

8. *Reset changes* – resets image to the default one
9. *Histogram* – create the histogram where x value is the approximate bacteria area calculated based on contours of thresholded bacteria shapes, y value – how many bacteria have this area. The xlim can be changed based on what you are looking for: *plt.xlim(your_min_value, your_max_value)*. The default range (-10,100).

The histogram can be created only after the thresholding. Otherwise, the error will pop up as not thresholded/masked image was created.

10. *Outline* – creates the outline for detected bacteria shapes.

It should be applied after the thresholding. Otherwise, the error will pop up as not thresholded/masked image was created.

11. *Exit* – both click on exit and X finishes the event loop and window is closing.

Steps to get the correct histogram:

1. Apply image transformation functions, if required, to make the cells more visible on the background.
2. Apply the thresholding for the cells.
3. Draw outline to visualize the small cells (it is required for the test3.jpg image).
4. Plot the histogram for your cells based on their areas.

Problems encountered during the work on project

Firstly, creating the GUI, but sample code helped a lot with the basic setup of the interface. Then the thresholding using HSV as in the sample code the range of sliders was set incorrectly. And now values are for OpenCV HSV: hue range is [0,179], saturation range - [0,255], and value range - [0,255], based on source.
The problem raised as well with logarithmic transformation where the type of image for calculations should have been float and then it should be converted back to uinit8 to work with it further. Otherwise, the error occurs (see Figure 5). In the case of outline, the OpenCV functions worked well. However, the code written during Practical 6 caused some problems and was too long compared to the OpenCV solution.
Finally, the histogram creation was followed by series of problems. Firstly, the area calculations. Luckily, there was an OpenCV function counterArea() which allows to calculate the area based on contour values. Then the problem raised with plotting as bacteria sizes are too small, almost all-around 0, with few exceptions, the graph can be not readable in the case of the higher number of cells in image. Thus, the xlim was set which can be optimized for different purposes in the code.

```
<ipython-input-1-90df90aa1937>:225: RuntimeWarning: divide by zero enco
untered in log
  log_img = ((const * (np.log(img_tmp + 1)))).astype("uint8")
```

```
log
```

Figure 5: The error in case of incorrect datatype in log transformation implementation.

The possible improvements for the existed solution

- Adding more functions. For example, more image restoration filters for better removal of characteristic noise.
- Improving the design of GUI, making it more user-friendly and adjustable to any computer.
- Saving the last version of the image, so the user could back up.
- The welcome window will allow to choose the file and open it next in the main menu.