Lauriane Lopes

INCISION QUALITY EVALUATION

ZDO – E

FINAL REPORT

2022 - 2023

Goal of the project: evaluate the quality of surgical stitching based on the image of the incision and the stitch.

This report presents a possible resolution to the project objective.

I. Main code

a. General presentation

The evaluation of surgical incisions plays a crucial role in the medical field. An accurate and well-made incision is essential for the success of a surgical procedure. However, these incisions cannot always be directly verified by an expert due to, for example, the recent increase in remote learning. One way to be able to evaluate the quality of the incision is to use computer vision as it is a technology capable of understanding and interpreting the visual content of images.

Therefore, this project was carried out using computer vision methods: this field is vast and involves many steps to complete this work. All this should be based on precise parameters to assess the quality of the incision such as the spacing between the different sutures or the angle between the main incision and the perpendicular threads.

b. Detailed explanation

In this section, the different methods used will be described along with the associated code.

First, several libraries have been used and can be divided into two parts: those made for the file structures (json) and those internal to the codes. Among these ones, we find *numpy* and the two libraries specific to computer vision: s*kimage* and *openCV*.

```python
#technical packages
import json
```

✓ 0.0s Python

```python
#work packages
import math

import numpy as np

import cv2

import skimage.io
from skimage import exposure, morphology, measure
from skimage.transform import (hough_line, hough_line_peaks)
```

✓ 0.1s Python

Before starting, a first step of data acquisition is necessary: 200 images of surgical incisions are available for this project. These images are read via *skimage* and are stored in black and white for further processing.

```
def main(*images):
```

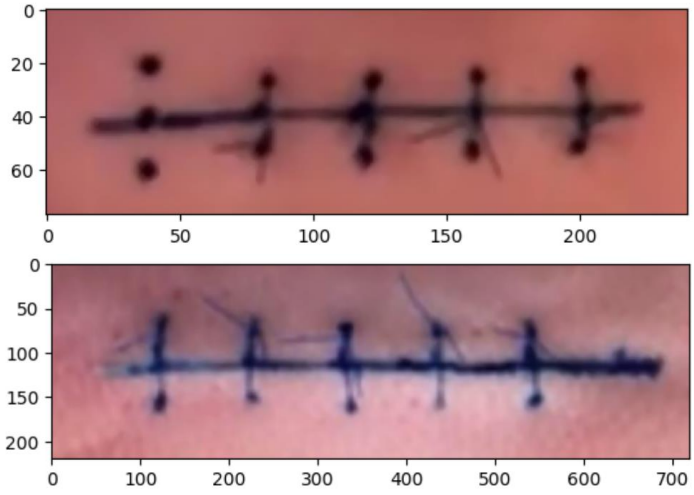
```
    for image in images:

        """FIRST PART: GET THE IMAGE READY FOR THE EVALUATION"""
        # image acquisition
        imagebw = skimage.io.imread(image, as_gray=True)
```

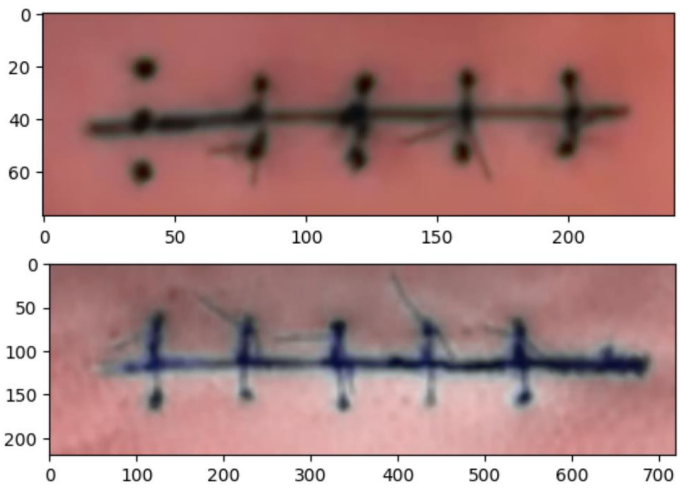
Now, here is the main function of this code. It brings together all the essential steps for the application of computer vision. Firstly, the image from the data set needs to be enhanced: obviously, the images are photographed differently and this implies several levels of contrast, brightness or even sharpness. Since these parameters vary from one image to another, it is necessary to choose adaptive functions, which explains the *equalize_adapthist()* function: it allows to improve contrast and luminosity thanks to the CLAHE method.

Also, to reduce the possible noise on the image, a filter is applied. Since a non-linear filter allows detecting edges, it is a filter of this type that is used, more precisely a median filter. This filter is available in both image processing libraries. However, a slight defect, a blurred contour, is present on the images processed with the filter from *skimage*: this is why the function from *openCV* is used for this work.

Processed with OpenCV



Processed with Skimage




```
        # image enhancement
        img_enhanced = exposure.equalize_adapthist(imagebw)
        img_enhanced = exposure.equalize_adapthist(image)
        img_filtered = cv2.medianBlur(img_enhanced.astype('float32'), 3)
```

For the segmentation step, the objective is to group significant objects and differentiate them from the background. An adaptive method is again used due to the large number of images: the Otsu method is suitable for this project as it returns an adapted threshold for each image. Also, the *block_size* parameter allows some precision: indeed, it defines the size of a pixel neighbourhood that

is used to calculate a threshold value for the pixel. Its value here will be 19, which is a rather small value, allowing to obtain a local threshold.

```
# image segmentation
block_size = 19
threshold = skimage.filters.threshold_otsu(img_filtered,block_size)
img_seg = img_filtered < threshold
```

With the segmented image, the objects are distinct but have ill-defined boundaries. To counter this problem, morphological operations are applied, as they allow the shape of the objects in the image to be reworked so that they are more easily distinguishable. After several tests, a certain combination of operations was chosen. First, an opening is applied to the image, thus smoothing the contours of the objects, removing small irregularities and separating objects that were very close to each other. Next, an area opening is used to close the holes in the objects, to regularise the shapes and to eliminate the small regions that are considered as noise for the evaluation of the incision.

```
# morphological operations
img_op = morphology.binary_opening(img_seg)
img_morpho = morphology.area_closing(img_op)
```

Then, now that the shapes of the objects have been improved, it is time to move on to the description of the objects. For this purpose, labelling is used to distinguish the different objects detected from the background. Following this, the *regionprops_table()* function from *skimage* is used to calculate different measures and properties for each region or object in the image such as area or perimeter. This is very useful as it is now possible to extract the object with the largest area, the incision, and remove all others. At the end, the incision and its stitches connected by the thread are the only object left on the image.

```
# object description
label = measure.label(img_morpho, background=0)
props = measure.regionprops_table(label, image, properties=['label','area','equivalent_diameter','mean_intensity
maxval = max(props['area'])
img_labelled = morphology.remove_small_objects(label, min_size=maxval-1)
```

The last step before analysing the quality of the incision is skeletonization as this allows a simplified and fine view of the incision.

The image has been improved and processed so that the incision is as visible and well designed as possible: it is now possible to assess the quality of the medical procedure. The idea to be able to successfully evaluate the incision is the following: first, we obtain the horizontal line representing the incision. Next, the Hough transform is used to obtain the vertical lines in the image, which are the sutures.

Let's go back to the step-by-step explanation. To obtain the horizontal line, we extract the indices of the elements other than 0 from the skeleton (which is now binary). Then, for the coordinates, we choose those at the extremities at the abscissa level.

```
    """SECOND PART: PREPARATION TO EVALUATION"""
    # get the horizontal line
    vertical = []
    intersections = []
    angles = []

    xx,yy = np.nonzero(img_skeleton)
    indmin = np.argmin(yy)
    indmax = np.argmax(yy)
    incision_line = ((yy[indmin],xx[indmin],yy[indmax],xx[indmax]))
```

Next, the vertical lines are detected using the Hough transform, which is always applied to the skeleton.

```
thetas = np.arange(-np.pi / 4, np.pi / 4, np.pi / 64).astype(float)
```

```
    # get the vertical ones
    h, theta, d = hough_line(img_skeleton, theta=thetas)
```

However, some lines are sometimes detected but these are false: this is due to the poor quality of the images which are sometimes visible on certain skeletons. To remove these lines from the evaluation, we use the *classifyLines()* function. This compares the difference between the abscissas to a certain tolerance: if the difference is greater than the defined tolerance, the processed line will not be retained.

```
def classifyLines(vLines, tolerance_verticale = 30):
```

With the coordinates of the different segments, it is possible to determine the intersections between them and also the resulting angle. The *intersectLines()* function for intersections is provided for this project.

```
def intersectLines(pt1, pt2, ptA, ptB):
```

The function for the angles, *calculateAngles()*, has also been given.

```
def calculateAngles(pt1, pt2, ptA, ptB):
```

All the arguments required for the json output file have been calculated and are accessible. Before creating the output file, we use json serialization to ensure that there are no format issues.

```
def convertJson(data):
    if isinstance(data, np.int64):
        return int(data)
    raise TypeError(f"Object of type {type(data)} is not JSON serializable")
```

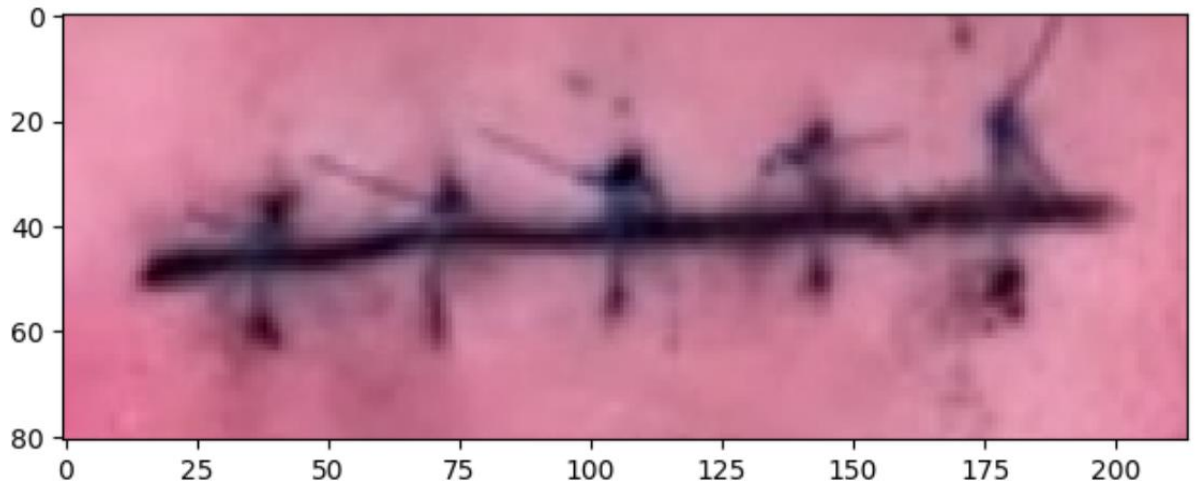
Finally, here is how the output file, *output.json*, is created and organised.

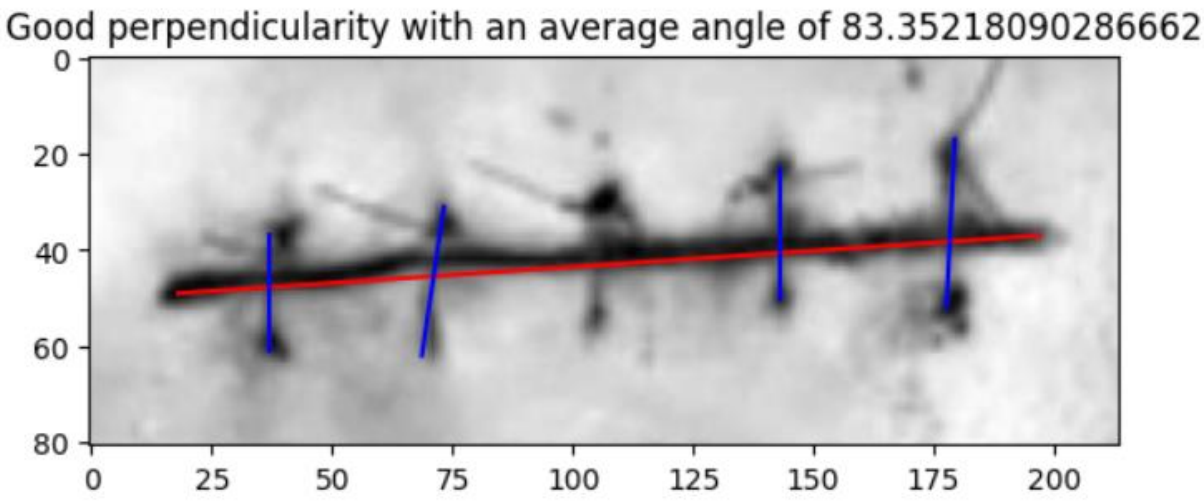
```
  """JSON OUTPUT"""
  data = [
      { "filename": str(sys.argv[3+i]),
        "incision_polyline": incision_line,
        "crossing_positions": intersections,
        "crossing_angles": angles,
        "coordinates_lines": lines,
        "evaluation": evaluation,
      },
  ]
```

II. Code demonstration

a. Results

Several images from the data set give very conclusive results. The main horizontal incision and the crossing positions have consistent coordinates, and the associated angles are even satisfactory. Here is a preview of an original image, followed by its final skeleton transformation and finally its output data:



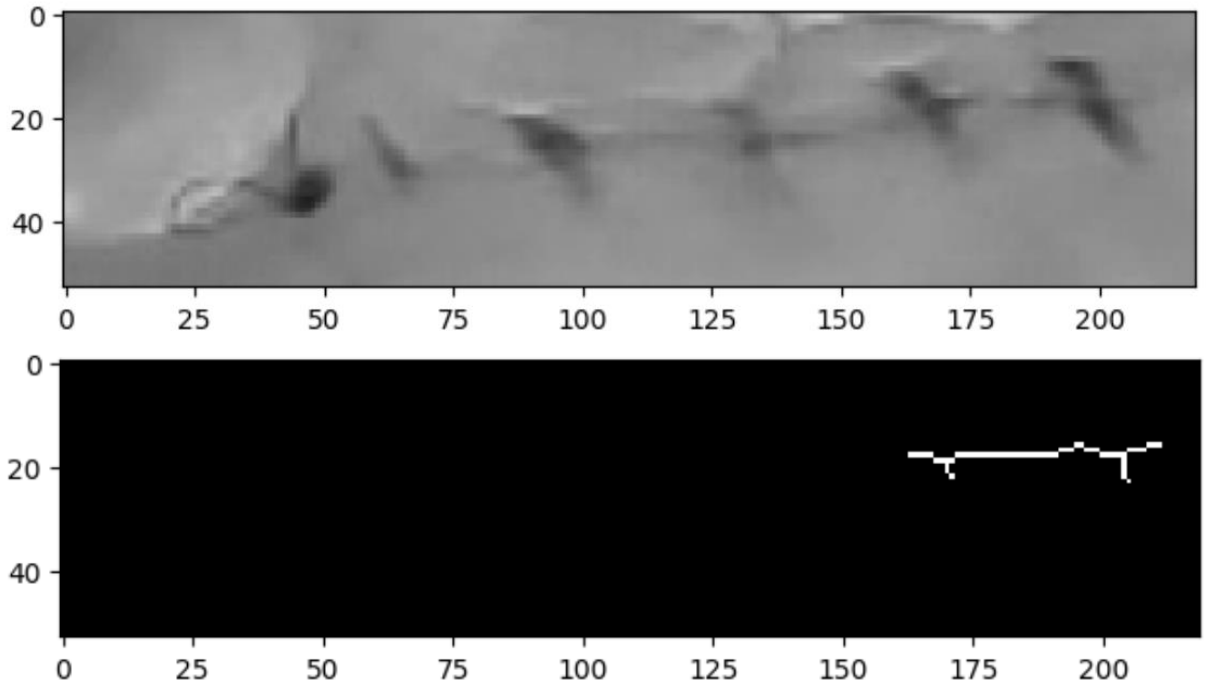



```
[
    [
        {
            "filename": "incision79",
            "incision_polyline": [
                18,
                49,
                197,
                37
            ],
            "crossing_positions": [
                [
                    178.33792386662444,
                    38.25108890279613
                ],
                [
                    142.99999999999994,
                    40.62011173184358
                ],
                [
                    71.10212856993604,
                    45.440080766261275
                ],
                [
                    36.99999999999995,
                    47.726256983240223
                ]
            ],
            "crossing_angles": [
                83.35218090286666,
                86.16468090286665,
                77.72718090286664,
                86.16468090286665
```

The evaluation option for the quality of the incision is unfortunately partial for this project. It only includes analysis of the angle and not the distance between the sutures. The average of the angles is calculated and, depending on the interval, this is classified differently.

```
if perpendicularity >= 85:
    evaluation.append((perpendicularity,"Excellent perpendicularity"))
elif 74 < perpendicularity < 85:
    evaluation.append((perpendicularity,"Good perpendicularity"))
else:
    evaluation.append((perpendicularity,"Bad perpendicularity"))
```

Obviously, all data sets in this type of project contain images of lower quality and this leads to some inconsistent results.



For the time given to this work and the tools used, there are unfortunately failures: these are the limits of the project.

b. Limits

This project does have some limitations: for some images, the code is not efficient enough. This problem is understandable since even with a powerful enhancement, some images that are initially of poor quality would not be usable: this project requires an automatic approach, so it is impossible to do case by case.

If more time had been allocated to this project, some improvements could have been made. For example, many techniques exist for each step of computer vision: it would have been interesting to test more of them to see which approach would be best.

This subject has allowed to discover a new field with many possibilities. Unfortunately, due to lack of time and experience, some steps like classification or the use of neural networks could not be developed: it would have been challenging to use these innovative techniques.