

Faculty of computer Engineering and Technology

first PROJECT REPORT

line segmentation

SUPERVISIOR: aziz qarosh

Sarah Arar ID: 1160880

salamAH SALAMAH ID: 1171611

Line segmentation | | 3/5/2021AD

# Image Processing:

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image. Nowadays, image processing is among rapidly growing technologies. It forms core research area within engineering and computer science disciplines too.

Image processing basically includes three steps:

1. Importing the image via image acquisition tools;
2. Analyzing and manipulating the image;
3. Output in which result can be altered image or report that is based on image analysis

There are two types of methods used for image processing namely, analogue and digital image processing:

Analogue image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques.

Digital image processing techniques help in manipulation of the digital images by using computers. The three general phases that all types of data have to undergo while using digital technique are:

Pre-processing, enhancement, and display, information extraction.

Our main focus is on Pre-processing level which includes the following steps:

1- Read image

2- Resize image

3-Remove noise (Denoise)

4-Segmentation

5- Morphology (smoothing edges)

## Noise in digital image processing – Noise filtering:

Noise in images:

Image noise is random variation of brightness or color information in the images captured. It is degradation in image signal caused by external sources. Images containing multiplicative noise have the characteristic that the brighter the area the noisier it. But mostly it is additive. We can model a noisy image as

Sources of Image noise:

1. While image being sent electronically from one place to another.
2. Sensorheat while clicking an image.
3. With varying ISOFactor which varies with the capacity of camera to absorb light.

Noise is always presents in digital images during image acquisition, coding, transmission, and processing steps. It is very difficult to remove noise from the digital images without the prior knowledge of filtering techniques.

Filtering image data is a standard process used in almost every image processing system. Filters are used for this purpose. They remove noise from images by preserving the details of the same. The choice of filter depends on the filter behavior and type of data.

We all know that, noise is abrupt change in pixel values in an image. So when it comes to filtering of images, the first intuition that comes is to replace the value of each pixel with average of pixel around it. This process smooth’s the image.

### Type of image noise filters:

There are different types of image noise filters. They can typically be divided into 2 types:

Time domain filters:

1. Mean
2. Median
3. Gaussian
4. Bilateral
5. Weiner

Frequency domain filters:

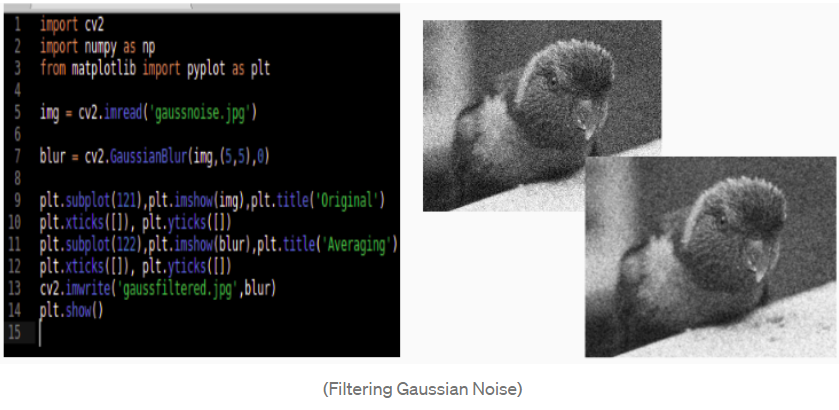
1. Low Pass filter
2. High Pass filter

The most used in Image Processing:

1. Gaussian filter:

In imageprocessing, a Gaussian blur (also known as Gaussian smoothing) is the result of blurring an image by a Gaussian function. It is a widely used effect in graphics software, typically to reduce image noise and reduce detail.

### Implementation of Gaussian Filter with OpenCV and Python:



# Line segmentation:

Text line segmentation is the process by which text lines in a document image are localized and extracted. It is an important step in off-line Handwritten Text Recognition (HTR) given that the input of these systems is the line image of the text to be transcribed.  
Separating text lines in unconstrained handwritten documents remains a challenge because the handwritten text lines are often un-uniformly skewed and curved, and the space between lines is not obvious.

# Thresholding:

**Thresholding** is the assignment of each pixel in an image to either a true or false class based on the pixel's value, location or both. The result of a thresholding operation is typically a [binary image](https://computervision.fandom.com/wiki/Binary_image) in which each pixel is assigned either a true or false value.

# Binary Image:

A binary image is a digital image that has only two possible values for each pixel. Binary images are also called *bi-level* or *two-level*. (The names black-and-white, B&W, monochrome or monochromatic are often used for this concept, but may also designate any images that have only one sample per pixel, such as grayscale images.)

Binary images often arise in image processing as masks or as the result of certain operations such as segmentation, [thresholding](https://computervision.fandom.com/wiki/Thresholding), and dithering.



# Morphological operations:

## Erosion and dilation:

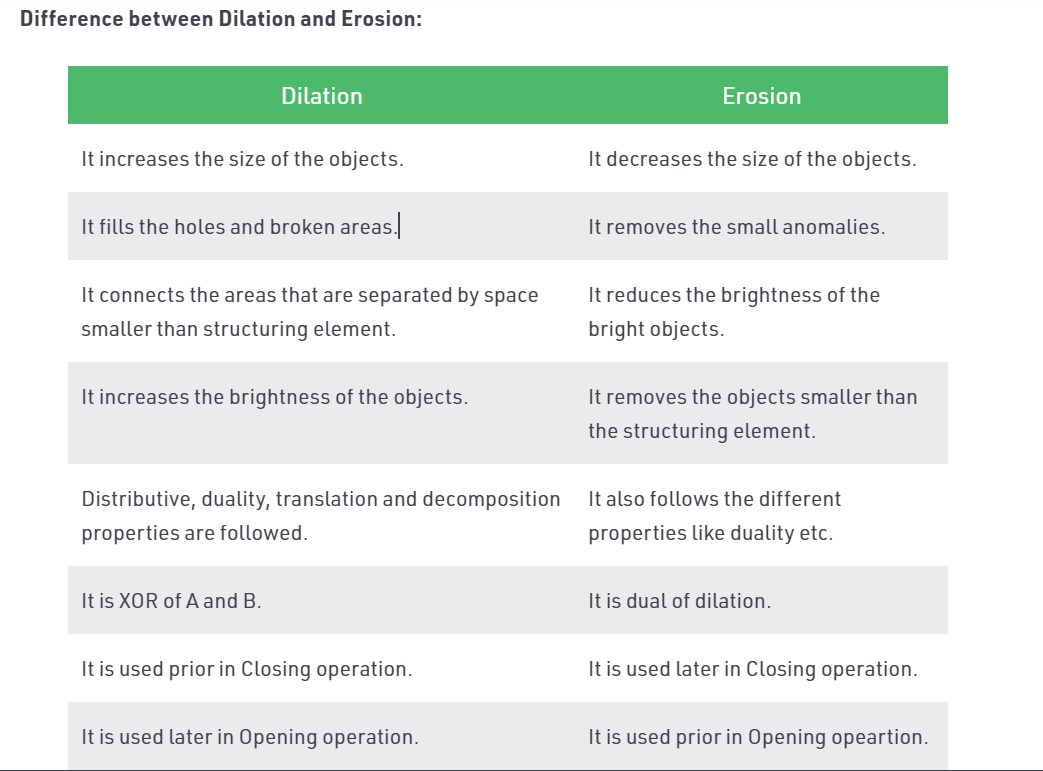
Dilation and Erosion are basic morphological processing operations that produce contrasting results when applied to either gray-scale or binary images.

Dilation is the reverse process with regions growing out from their boundaries.

Dilation is A XOR B.

Erosion involves the removal of pixels ate the edges of the region.

Erosion is just the dual of Dilation

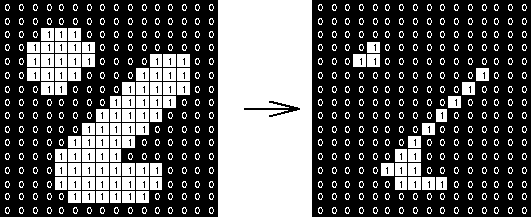


## Erosion process:

(Usually represented by ⊖) is one of two fundamental operations (the other being [dilation](https://en.wikipedia.org/wiki/Dilation_(morphology))) in [morphological image processing](https://en.wikipedia.org/wiki/Morphological_image_processing) from which all other morphological operations are based. It was originally defined for [binary images](https://en.wikipedia.org/wiki/Binary_image), later being extended to [grayscale](https://en.wikipedia.org/wiki/Grayscale) images

### How it is works?

The erosion operator takes two pieces of data as inputs. The first is the image which is to be eroded. The second is a (usually small) set of coordinate points known as a [structuring element](https://homepages.inf.ed.ac.uk/rbf/HIPR2/strctel.htm) (also known as a [kernel](https://homepages.inf.ed.ac.uk/rbf/HIPR2/kernel.htm)). It is this structuring element that determines the precise effect of the erosion on the input image.



## Dilation process:

It is a morphological operation used to enhance the features of an image. Dilation as a function requires two inputs, an image to be dilated, and a two dimensional  structuring element  Dilation has many applications, but is most commonly used to exaggerate features in an image that would otherwise be missed.

Dilation is frequently used to enhance the structure of an image, by either exaggerating features in the foreground, or conversely, obscuring features in the background.

# What is contour?

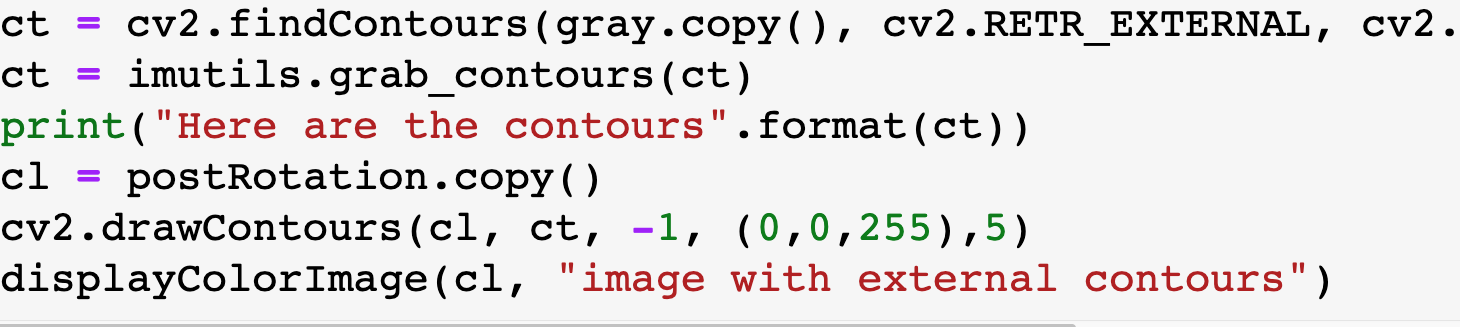
Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

1. For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.
2. Since OpenCV 3.2, [**findContours()**](https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html) no longer modifies the source image.
3. In OpenCV, finding contours is like finding white object from black background. Object to be found should be white and background should be black.

# Image Contouring:

Image contouring is process of identifying structural outlines of objects in an image which in turn can help us identify shape of the object.

Code for drawing contour with OpenCV:



## Bounding Boxes:

Bounding boxes are rectangular (or circular) area that encloses the object. For rectangle object it coincides with contour. Bounding boxes are useful for showing face boundaries in face recognition application.

## **Contour Properties:**

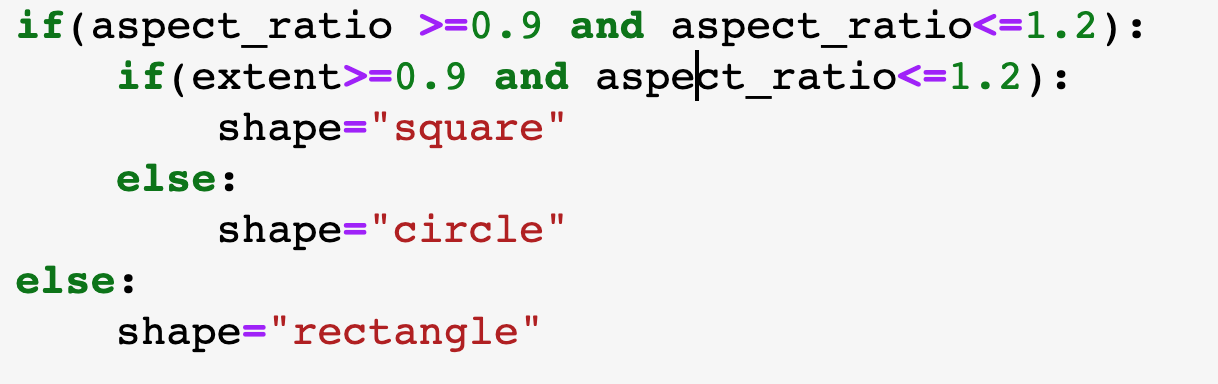
Contour properties like aspect ratio, extent, solidity are useful for detecting the shape.

Aspect ratio: width/height. For rectangle shape aspect ratio < 1 or > 1 but for square aspect ratio is 1

Extent:contour area/bounding box area. For rectangle or square extent is close to 1. For irregular sign like lightning symbol, extent < 1 as shown below.

Convex hull and solidity:

Convex hull is minimum convex area that encompasses all the points in set T  
solidity = contour area / convex area. For irregular objects its <1. For regular objects like circle and square it is close to 1.

We can use these properties together to determine shape

# Related works:

The algorithms that try to locate the text lines in a document are divided mainly by the information they take from the input document. As the input documents are the results of a digitization process, they are acquired, generally through scanning, as grayscale images. This grayscale representation is converted into binary or black and white for algorithms which are designed to work with this type of documents only. The binary representation conversion is done using a previously defined threshold level. All pixels that have gray levels above the threshold are converted to black, the rest are converted to white. A too high threshold will results in an image containing too little information for text recognition to be possible and a too low threshold will result in too many artifacts, again making the text recognition process impossible. Based on the observation that the body of the lines of text contains gray pixels and because the pixels that make the characters being of a darker shade of gray, some algorithms compute projection profiles representing the sum of all the pixels values in a given direction. The method works well for printed documents but fails to produce good results when applied to handwritten documents. To address these problems, different approaches were used: identify the local skew of the handwritten text, calculate the accumulated space between characters, and try to fill the space between characters or use attraction from the text pixels and repulsion from the previously detected line trying to estimate the text line boundaries close

<https://www.sciencedirect.com/science/article/abs/pii/S0031320308005293>

<https://ieeexplore.ieee.org/document/7333819>

<https://www.researchgate.net/publication/341982308_Survey_on_Segmentation_and_Recognition_of_Handwritten_Arabic_Script>

<https://towardsdatascience.com/image-pre-processing-c1aec0be3edf>

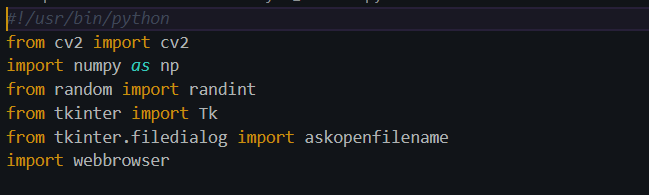
<https://stackoverflow.com/questions/52920472/segmenting-handwritten-text-into-lines>

<https://core.ac.uk/download/pdf/236053908.pdf>

# Approach and procedure:

For achieving the line segmentation process we have done pre-processing steps which can be concluded in the following steps:

Before going further in pre-processing steps we have imported the needed libraries as the following:



1. Read the image :   
   In this step we have stored the path on a variable name “ imgname” and we have check if it have loaded correctly using the conditional if statement , then using OpenCV method “imread” to load the image to our workspace ,stored in image variable

The function imread loads an image from the specified file and returns it. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format), the function returns an empty matrix ( [**Mat::data**](https://docs.opencv.org/master/d3/d63/classcv_1_1Mat.html)==NULL ).

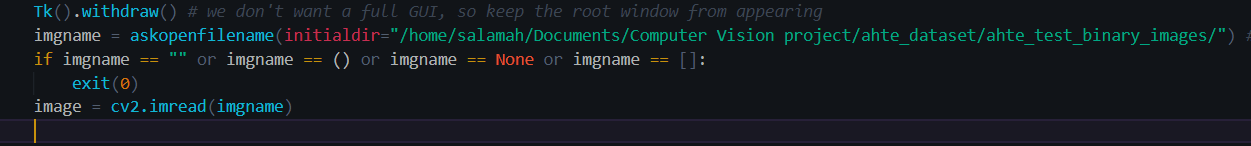
Imread syntax:



Parameters:

Filename: Name of file to be loaded   
flags: Flag that can take values of cv::ImreadModes

As it used in our code:



Then we have make an image copy and we have stored it in imcopy variable to continue our working using it using the following command:

    imgcpy = image.copy ()

2- Resize the image:

In this step we have resized the image to visualize the changes we make into this image

Using the OpenCV function:   
cv2.resize:

Resizing image can be done in many ways:

1. Preserve Aspect Ratio (height to width ratio of image is preserved)
   1. Downscale (Decrease the size of the image)
   2. Upscale (Increase the size of the image)
2. Do not preserve Aspect Ratio
   1. Resize only the width (Increase or decrease the width of the image keeping height unchanged)
   2. Resize only the height (Increase or decrease the height of the image keeping width unchanged)
3. Resize to specific width and height

## Resizing with Preserving the aspect ratio (upscale , downscale)

* + 1. Downscale :

Scale\_percent value holds the percentage by which image has to be scaled. Providing a value <100 downscales the image provided as shown in the following code:



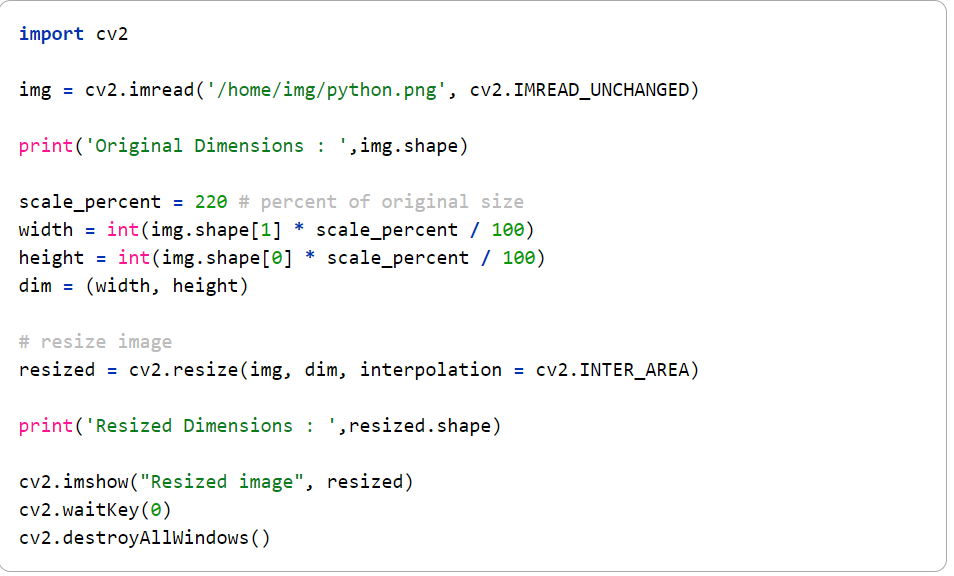
Output:

Original Dimensions: (149, 200, 4)

Resized Dimensions: (89, 120, 4)

* + 1. Upscale:

Scale\_percent value holds the percentage by which image has to be scaled. Providing a value >100 upscale the image provided.

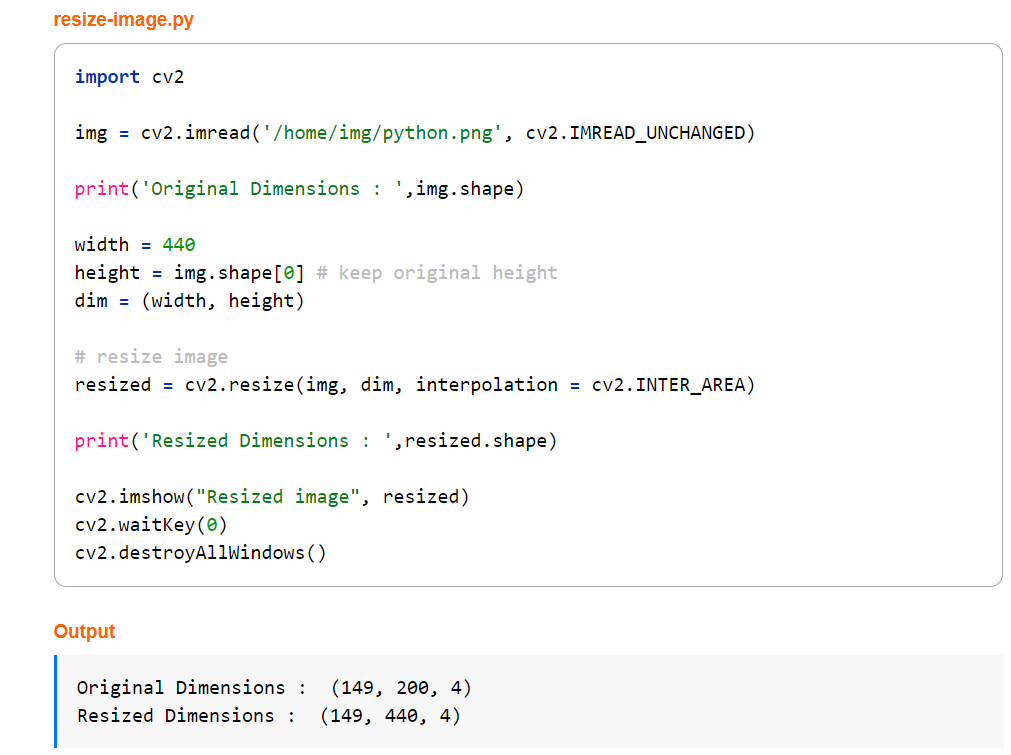


Original Dimensions: (149, 200, 4)

Resized Dimensions: (327, 440, 4)

## Resize without preserving aspect ratio:

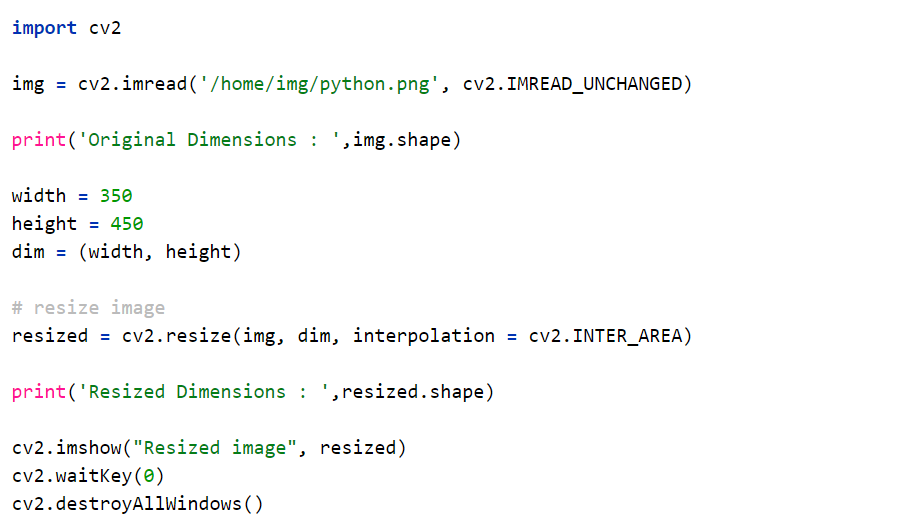
1. Changing the width keeping the height unchanged:



1. Changing the height keeping the width unchanged :



## Resize to specific width and height



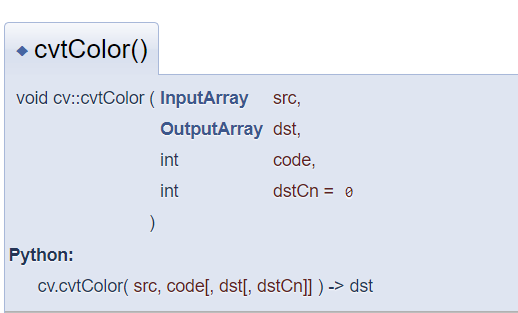
We have done in our code is the second approach which is changing the height without preserving aspect ratio using the following line of code:

    imgcpy = cv2.resize(imgcpy , (imgcpy.shape[1],imgcpy.shape[0]\*3))

## Converting to grayscale image:

A grayscale (or graylevel) image is simply one in which the only colors are shades of gray. The reason for differentiating such images from any other sort of color image is that less information needs to be provided for each pixel. In fact a `gray' color is one in which the red, green and blue components all have equal intensity in [RGB space](https://homepages.inf.ed.ac.uk/rbf/HIPR2/rgb.htm), and so it is only necessary to specify a single intensity value for each pixel, as opposed to the three intensities needed to specify each pixel in a [full color image](https://homepages.inf.ed.ac.uk/rbf/HIPR2/colimage.htm).

Using the cvtColor-convert color:



Converts an image from one color space to another.

The function converts an input image from one color space to another. In case of a transformation to-from RGB color space, the order of the channels should be specified explicitly (RGB or BGR). Note that the default color format in OpenCV is often referred to as RGB but it is actually BGR (the bytes are reversed). So the first byte in a standard (24-bit) color image will be an 8-bit Blue component, the second byte will be Green, and the third byte will be Red. The fourth, fifth, and sixth bytes would then be the second pixel (Blue, then Green, then Red), and so on.

As it shown in our code :

 gray = cv2.cvtColor(imgcpy, cv2.COLOR\_BGR2GRAY)

    cv2.imwrite("v1\_out/gray.png", gray)

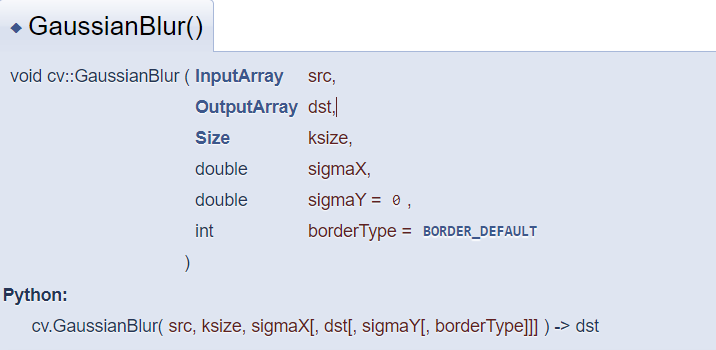
## Removing noise using Gaussian blur:

For smoothing purpose we have make a Gaussian filtration:

Gaussianblur (also known as Gaussiansmoothing) is the result of blurring an image by a Gaussian function. It is a widely used effect in graphics software, typically to reduce [image noise](https://en.wikipedia.org/wiki/Image_noise). The visual effect of this blurring technique is a smooth blur resembling that of viewing the [image](https://en.wikipedia.org/wiki/Image).

 blur = cv2.GaussianBlur(gray, (3,3), 0)

    cv2.imwrite("v1\_out/blur.png", blur)  
Cv2.GaussianBlur syntax:



Blurs an image using a Gaussian filter.

The function convolves the source image with the specified Gaussian kernel. In-place filtering is supported.

Parameters:

Src : input image

KSize : the Gaussian kernel size ksize.width, ksize.height

## Thresholding process:

Thresholding is a type of image segmentation, where we change the pixels of an image to make the image easier to analyze. In thresholding, we convert an image from color or grayscale into a binary image, i.e., one that is simply black and white.

We use the function: [**cv.threshold**](https://docs.opencv.org/3.4/d7/d1b/group__imgproc__misc.html) (src, dst, thresh, maxval, type)

Parameters

Src : input image

Threshold : threshold value

Maxval : max value to use with the cv.thresh \_binary

thresholding type - OpenCV provides different styles of thresholding and it is decided by the fourth parameter of the function.

## Finding contour:



Finds contours in a binary image. The function retrieves contours from the binary image using the algorithm. The contours are a useful tool for shape analysis and object detection and recognition. See squares.cpp in the OpenCV sample directory.

Parameter:

1. Image : src image to applying contour on
2. Contours: Detected contours. Each contour is stored as a vector of points
3. Mode: Contour retrieval mode
4. Method: contour approximation method
5. Offset: Optional offset by which every contour point is shifted

As in our code, there are three arguments in [**cv.findContours()**](https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html) function, first one is source image, second is contour retrieval mode, third is contour approximation method. And it outputs the contours and hierarchy. Contours is a Python list of all the contours in the image. Each individual contour is a Numpy array of (x,y) coordinates of boundary points of the object.

    cnts = cv2.findContours(thresh, cv2.RETR\_EXTERNAL , cv2.CHAIN\_APPROX\_SIMPLE)

## Drawing the contour:

To draw the contours, [cv.drawContours](https://docs.opencv.org/master/d6/d6e/group__imgproc__draw.html) function is used. It can also be used to draw any shape provided you have its boundary points. Its first argument is source image, second argument is the contours which should be passed as a Python list, third argument is index of contours (useful when drawing individual contour. To draw all contours, pass -1) and remaining arguments are color, thickness etc.

-To draw all the contours in an image:

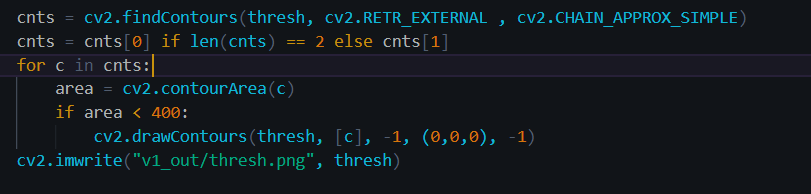
[cv.drawContours](https://docs.opencv.org/master/d6/d6e/group__imgproc__draw.html" \l "ga746c0625f1781f1ffc9056259103edbc)(img, contours, -1, (0,255,0), 3)

* To draw an individual contour, say 4th contour:

[cv.drawContours](https://docs.opencv.org/master/d6/d6e/group__imgproc__draw.html" \l "ga746c0625f1781f1ffc9056259103edbc)(img, contours, 3, (0,255,0), 3)

* But most of the time, below method will be useful:

cnt = contours[4]

[cv.drawContours](https://docs.opencv.org/master/d6/d6e/group__imgproc__draw.html" \l "ga746c0625f1781f1ffc9056259103edbc)(img, [cnt], 0, (0,255,0), 3)  
  


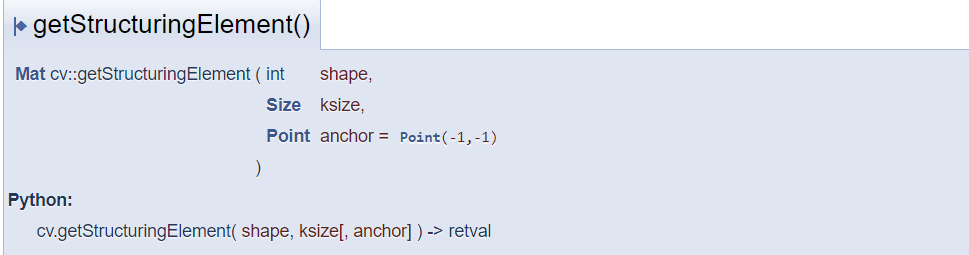
Draw contour syntax:



Draws contours outlines or filled contours.

The function draws contour outlines in the image if thickness≥0 or fills the area bounded by the contours if thickness<0 . The example below shows how to retrieve connected components from the binary image and label them

# Get structuring element:



Returns a structuring element of the specified size and shape for morphological operations.The function constructs and returns the structuring element that can be further passed to [**erode**](https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html), [**dilate**](https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html) or [**morphologyEx**](https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html).

Parameter:

Shape: Element shape that could be (MORPH\_RECT, MORPH\_ELIPS, MORPH\_CROSS)

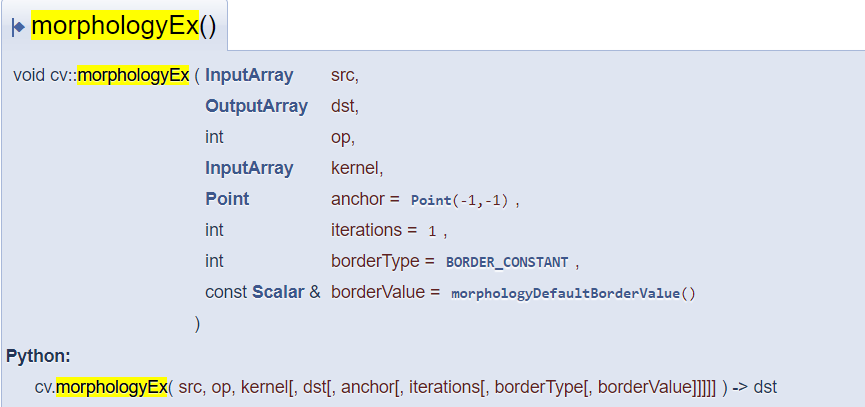
Ksize: the size of the structuring element

kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (20,5))

## Morphology:

Performs advanced morphological transformations.The function [**cv::morphologyEx**](https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html) can perform advanced morphological transformations using an erosion and dilation as basic operations.

Any of the operations can be done in-place. In case of multi-channel images, each channel is processed independently.

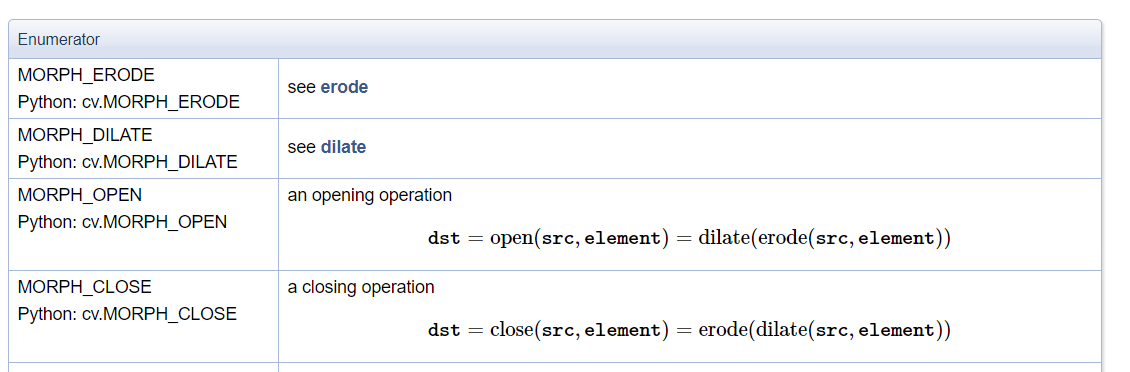


Src: source image.

Op: type of a morphogical operation

Kernel: Structuring element. It can be created using [**getStructuringElement**](https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html).

Iteration: Number of times erosion and dilation are applied



Our code:

 kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (20,5))

    close = 255 - cv2.morphologyEx(thresh, cv2.MORPH\_CLOSE, kernel, iterations=2)

    cv2.imwrite("v1\_out/close.png", close)

    cv2.imwrite("v1\_out/thresh\_v1.png", thresh)

## Np.zeros and np.ones

np.zeros "Return a new array of given shape and type, filled with zeros."

np.ones "Return a new array of given shape and type, filled with ones."

np.zeros (10)

Out: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

np.ones (10)

Out: array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])

 kernel = np.ones((1,60), np.uint8)

    close = cv2.erode(close,kernel,iterations=4)

    cv2.imwrite('v1\_out/close\_v2.png',close)

    cnts = cv2.findContours(close, cv2.RETR\_EXTERNAL , cv2.CHAIN\_APPROX\_SIMPLE)

    cnts = cnts[0] if len(cnts) == 2 else cnts[1]

    for c in cnts:

        area = cv2.contourArea(c)

        if area < 300:

            cv2.drawContours(close, [c], -1, (0,0,0), -1)

    kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (100,1))

    close =  cv2.morphologyEx(close, cv2.MORPH\_CLOSE, kernel, iterations=3)

    cnts = cv2.findContours(close, cv2.RETR\_EXTERNAL , cv2.CHAIN\_APPROX\_SIMPLE)

    cnts = cnts[0] if len(cnts) == 2 else cnts[1]

    for c in cnts:

        area = cv2.contourArea(c)

        if area < 10000:

            cv2.drawContours(close, [c], -1, (0,0,0), -1)

    cv2.imwrite("v1\_out/close\_v4.png", close)

    kernel = np.ones((1,30), np.uint8)

    close = cv2.erode(close,kernel,iterations=2)

    cv2.imwrite('v1\_out/close\_v5.png',close)

    kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (140,1))

    close = cv2.dilate(close,kernel, iterations=2)

    cv2.imwrite("v1\_out/close\_v6.png", close)

    close = cv2.resize(close , (imgcpy.shape[1],image.shape[0]))

    imgcpy = cv2.resize(imgcpy , (imgcpy.shape[1],image.shape[0]))

    (contours, \_) = cv2.findContours(close, cv2.RETR\_EXTERNAL,cv2.CHAIN\_APPROX\_SIMPLE)

    for cnt in contours:

        area = cv2.contourArea(cnt)

        if area > 10000:

            x,y,w,h = cv2.boundingRect(cnt)

            cv2.rectangle(imgcpy,(x-1,y-5),(x+w,y+h),(randint(0, 255),randint(0, 255),randint(0, 255)),5)

    cv2.imwrite("v1\_out/imgContoure.png", imgcpy)

    webbrowser.open("v1\_out/imgContoure.png")

As you can see the lifted code is a repetition for the same procedure, so we can get better results!

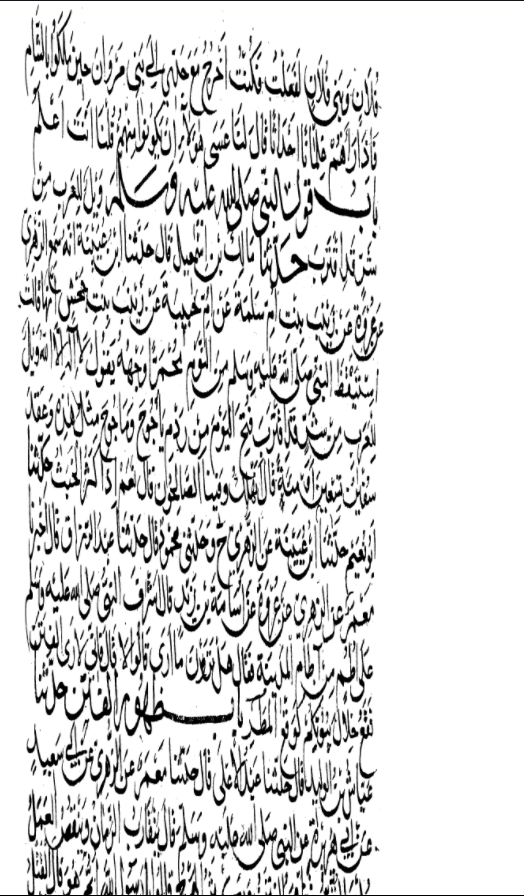
# Results and discussion:

By applying the previous operations we have ended with those results:

## Converting to grayscale :

1. gray = cv2.cvtColor(imgcpy, cv2.COLOR\_BGR2GRAY)
2. cv2.imwrite("v1\_out/gray.png", gray)

The output of this piece of code :



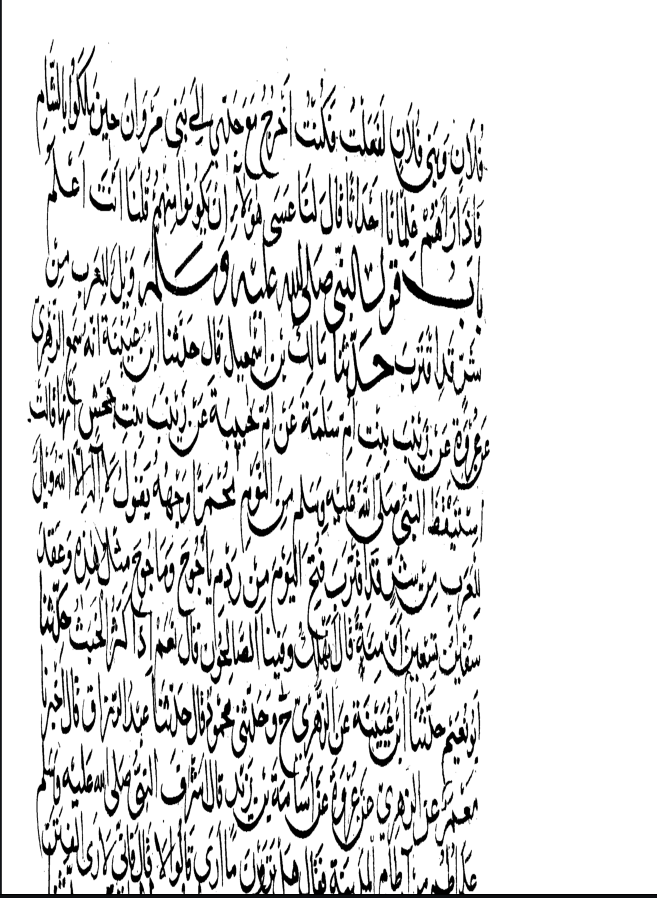
As the result of this process we will end with a picture has only two color after conversion of colors level we will get a pic which is varies between a complete black and complete white.

## Applying a Gaussian blur:

 Says photographer Kenton Waltz. “It softens everything out.” A type of low-pass filter, Gaussian blur smoothing uneven pixel values in an image by cutting out the extreme outliers.

 blur = cv2.GaussianBlur(gray, (3, 3), 0)

    cv2.imwrite("v1\_out/blur.png", blur)



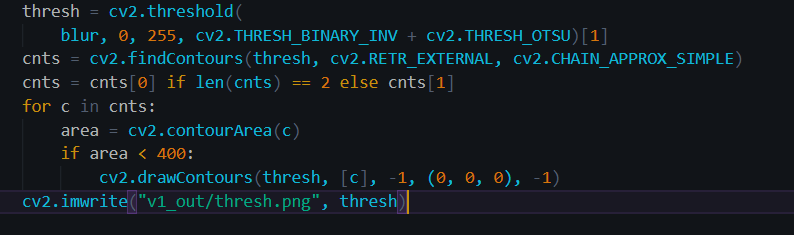
The output of this process is a smoothed copy of the original picture, blurring process is used to make the picture smooth in which the edges cannot be observed edges to, so if we have an image with the following dimensions 508\*340=172,720 pixels

How Gaussian blur works in image filtering?

Both grayscale and color images can contain a lot of noise, or random variation in brightness or hue among pixels. The pixels in these images have a high standard deviation, which just means there’s a lot of variation within groups of pixels. Because a photograph is two-dimensional, Gaussian blur uses two mathematical functions (one for the x-axis and one for the y) to create a third function, also known as a convolution. This third function creates a normal distribution of those pixel values, smoothing out some of the randomness. How much smoothing depends on the size of the blur radius you choose. Each pixel will pick up a new value set to a weighted average of its surrounding pixels, with more weight given to the closer ones than to those farther away. The result of all this math is that the image is hazier.

## Thresholding process:

Thresholding is a technique in OpenCV, which is the assignment of pixel values in relation to the threshold value provided. In thresholding, each pixel value is compared with the threshold value. If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to a maximum value (generally 255). Thresholding is a very popular segmentation technique, used for separating an object considered as a foreground from its background. A threshold is a value which has two regions on its either side i.e. below the threshold or above the threshold.  
In Computer Vision, this technique of thresholding is done on grayscale images. So initially, the image converted as shown above to gray scale.





## Erosion and dilation:

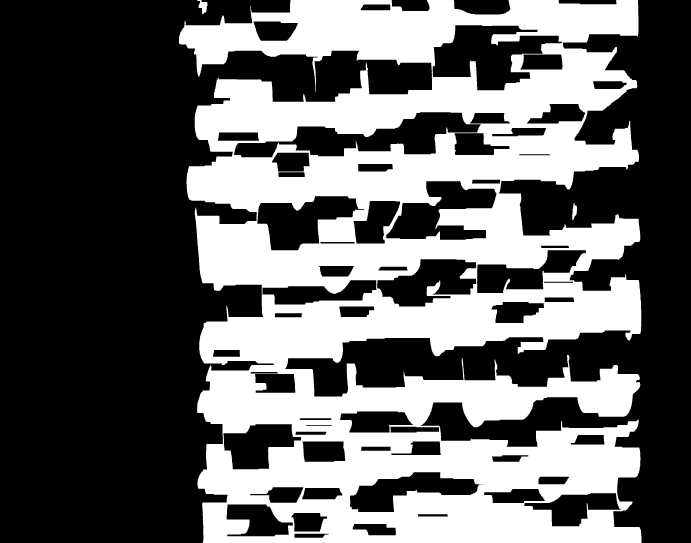
We have improved our results repeating the use of morphological operation such erosion and dilation as shown below:

close\_kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (35, 1))

    close = cv2.morphologyEx(thresh, cv2.MORPH\_CLOSE,

                             close\_kernel, iterations=3)

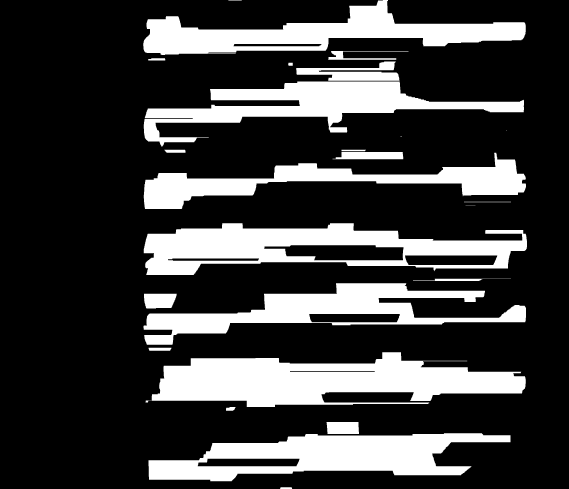
    cv2.imwrite('v1\_out/close\_v1.png', close)



 kernel = np.ones((1, 60), np.uint8)

    close = cv2.erode(close, kernel, iterations=4)

    cv2.imwrite('v1\_out/close\_v2.png', close)



  cnts = cv2.findContours(close, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

    cnts = cnts[0] if len(cnts) == 2 else cnts[1]

    for c in cnts:

        area = cv2.contourArea(c)

        if area < 300:

            cv2.drawContours(close, [c], -1, (0, 0, 0), -1)

    cv2.imwrite("v1\_out/close\_v3.png", close)



 kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (100, 1))

    close = cv2.morphologyEx(close, cv2.MORPH\_CLOSE, kernel, iterations=3)

    cnts = cv2.findContours(close, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

    cnts = cnts[0] if len(cnts) == 2 else cnts[1]

    for c in cnts:

        area = cv2.contourArea(c)

        if area < 10000:

            cv2.drawContours(close, [c], -1, (0, 0, 0), -1)

    cv2.imwrite("v1\_out/close\_v4.png", close)



 kernel = np.ones((1, 30), np.uint8)

    close = cv2.erode(close, kernel, iterations=2)

    cv2.imwrite('v1\_out/close\_v5.png', close)

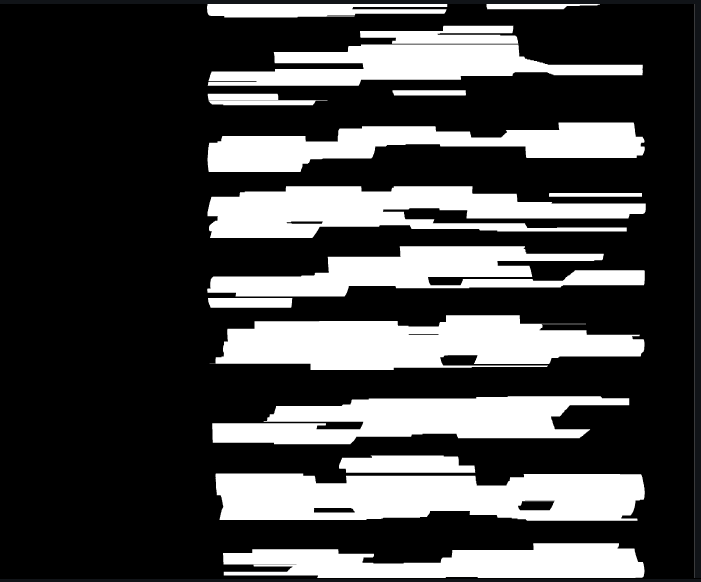
    kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (140, 1))

    close = cv2.dilate(close, kernel, iterations=2)

    cv2.imwrite("v1\_out/close\_v6.png", close)

close\_v5:



Close\_v6:   


## Final results:

 close = cv2.resize(close, (imgcpy.shape[1], image.shape[0]))

    imgcpy = cv2.resize(imgcpy, (imgcpy.shape[1], image.shape[0]))

    (contours, \_) = cv2.findContours(

        close, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

    for cnt in contours:

        area = cv2.contourArea(cnt)

        if area > 10000:

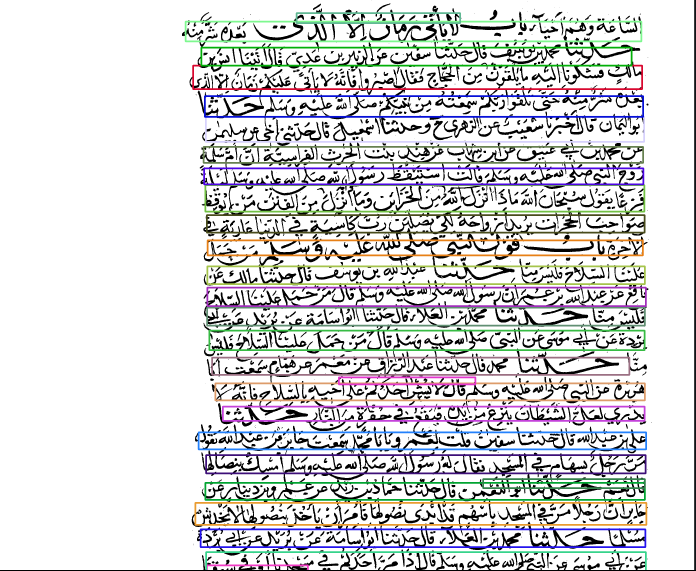
            x, y, w, h = cv2.boundingRect(cnt)

            cv2.rectangle(imgcpy, (x-1, y-5), (x+w, y+h),

                          (randint(0, 255), randint(0, 255), randint(0, 255)), 5)

    cv2.imwrite("v1\_out/imgContoure.png", imgcpy)

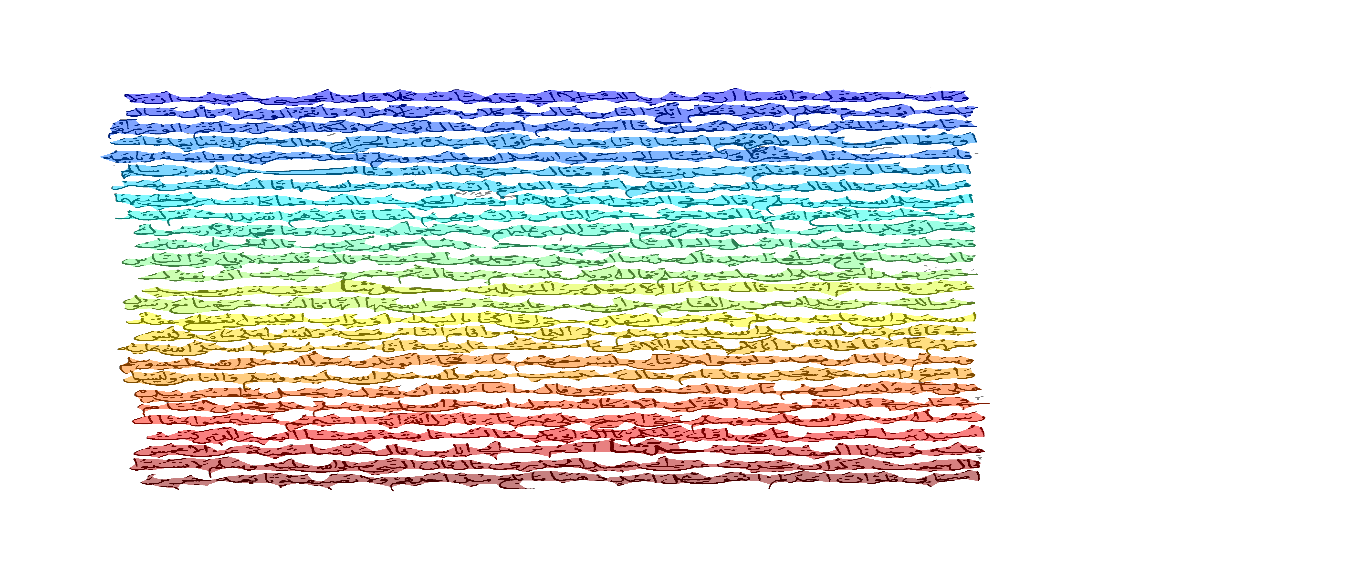
    webbrowser.open("v1\_out/imgContoure.png")

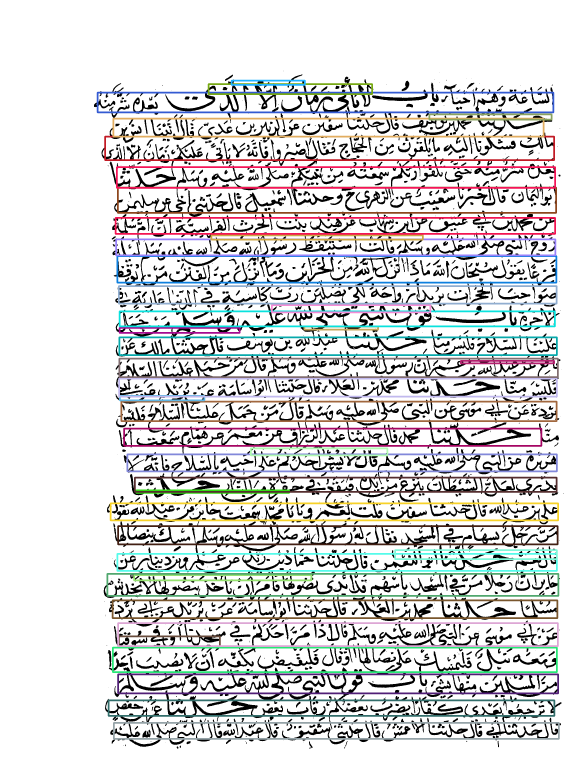


As you can see after trying to get the best results by using image processing technique, we have reached the result of line segmentation “contoured text lines”.

# Comparing to other related work:

Comparing with the data set you have sent we can say we have succeeded in the process of extraction and defining lines in Arabic-handwritten paragraph   
as you can see the data you have sent looks like this:

   
Our result:



# Limitation and contribution

Two limitations were facing us while developing and trying to get our best solution. The first one, connections between lines, it was a very hard to deal with, especially because we were using kernels and morphology, we already tried to make a general kernel that is valid for all the data set, but, the best kernel we could find was not that good on 2 or 3 images, but it was very good at all.

The another limit, was words or letters with very lighty color, when doing erosion, the line itself may be affected because these words will be fully deleted, we could solve this problem, by increasing the morphology ratio/kernel on the x axis, but there is still some minor problems, that solving them will make a new bugs and bigger problems.