CDS540 Fall 2024 Assignment 1

--Text Detection from Images Using OpenCV

LIU XU -- xuliu@ln.hk

STU ID : 5584661

Oct 15th, 2024

**Link:**

Youtube:[CDS540 Assignment 1 demo (youtube.com)](https://www.youtube.com/watch?v=HQ5kqODRnHU)

（ https://youtu.be/HQ5kqODRnHU ）

Github: [XZJLL/CV\_IndividualAssignment (github.com)](https://github.com/XZJLL/CV_IndividualAssignment)

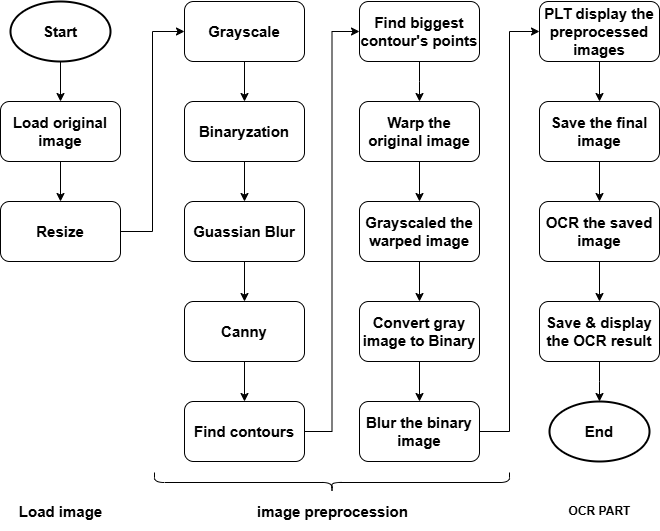
**Project Design:**

1. Environment:

VSCODE+Python3.7+OPENCV4+Pytesseract(The OCR/text detection method)

1. Flow Chart (& function and algorithm used):

Version 1(basic function):



|  |  |
| --- | --- |
| Process | Major functions and Algorithm used |
| load image  Show image  Write image | cv2.imread(filename,flag)  cv2.imshow(“winname”,image) / plt.title()+plt.imshow()+plt.show()  cv2.imwrite(“filename”,image) |
| Grayscale | cv2.cvtColor(image,cv.COLOR\_RGB2GRAY) |
| Binary | cv2.threshold(image,Threshold,255,cv.THRESH\_BINARY) |
| Blur | cv2.GaussianBlur(image,(kernel,kernel),sigma X) |
| Find edges | cv2.Canny(image, L\_Thresh, H\_Thresh) |
| Find contours  Find the largest contour  Draw contours | cv2.findContours(image,mode,method)  sorted(Contours, key=cv.contourArea, reverse=True)  Perimeter = cv2.arcLength(cnt, True)  Approx rectangle = cv2.approxPolyDP(cnt,epsilon,True),iterate to find the closest contour  cv2.drawContours(image,contours,contourID,color,lineThick,lineType) |
| Warp | Matrix = cv2.getPerspectiveTransform(src, dst)  cv2.warpPerspective(image, Matrix, (width, height)) |
| OCR  OCR box  Text display | pytesseract.image\_to\_string(image)  boxes = pytesseract.image\_to\_data(image), enumerate to find useful data  cv.putText(img, text, pos, fontFace,fontScale,color) |
| OCR result file saving  OCR result file reading and printing | with open('filename.txt', 'w') as file:  f.write(str(text))  with open("filename.txt", "r") as file:  print(file.read()) |

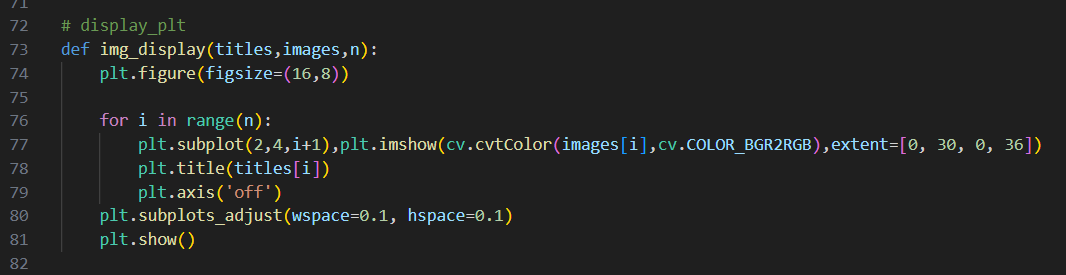
**Code Explanation and Analysis:**

The Code can be separated into three parts briefly: the image loading part, the image preprocessing part, and the OCR part.

The code used for loading the image is cv. Imread () and for writing image is cv. imwrite()

The project uses 2 methods to handle the source image, the opencv way and the matplotlib way. It is necessary to write the cv2.waitKey() function to display the image in OPENCV way, while plt.show() is neccesary for plt.imshow() to display image in matplotlib way.

The code to display multiple images in a single window is:



First set up the figure size, then display each image in the location in a loop, and finally adjust the distance between each image.

1. Image preprocessing techniques and algorithms:

* Grayscale: 1729167525759

In the function, the “image” is the image that needs to be converted, flag “cv.COLOR\_RGB2GRAY” means to turn the image to gray.

* Binary:

1729167782632

In the function, the first variable is the source image, then the low threshold and the high threshold, the pixels with values higher than the first threshold will be turned into value 255. flag “THRESH\_BINARY” specifies the method used. After several tests, it turns out that 120 is the best low-threshold value to output the best result.

* Blur:

1729168116233

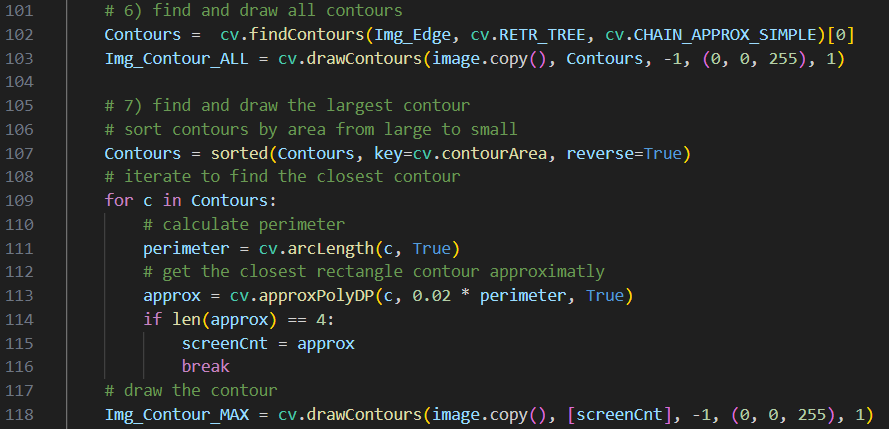
There are many ways to get rid of the noise of the image, here in the project, GaussianBlur has the best result. The second variable in the function is kernel size, the bigger the kernel size, the image turns out to be blurred.

* Edge Detection

1729168290560

The canny function is a popular method to do edge detection. It can find the edge in the image, and this algorithm can eliminate some noise in the image too. The second and the third variable here is the lower and upper limit, only the edge inside this limit will be displayed.

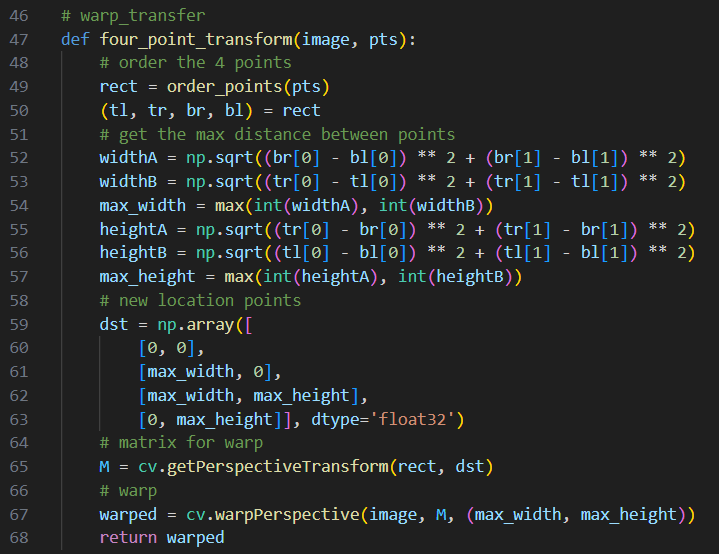
* Find contour:



In this part, I use cv.findContours() to find the contours in the image, then use cv.drawContours to draw all contours. The image used in the cv.findContours() should be a Binary image, the second variable in the function is the mode used to detect contours, and the third variable is the method used to increase the speed. Be careful, this function will draw on the input image, so use the image.copy() instead of the source image.

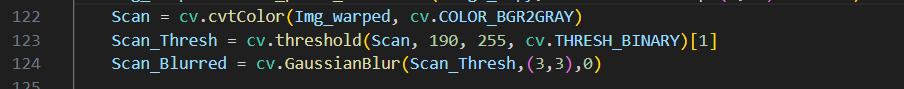
After getting all the contours of the image, I use the sort() function to list contours from large to small. And get the approximate outer rectangle contour of the image. This rectangle contour can used in the next part: Perspective Transform.

* Perspective transform:



1729174081508

This part uses Perspective Transform to transform the tilted image transformed to the right place. cv2.getPerspectiveTransform() can calculate the Matrix for perspective transform. The first variable in this function is the original points, the second variable is the new points. Then use cv.warpPerspective() to warp the image. Use the tl pts -brpts to get the max distance.

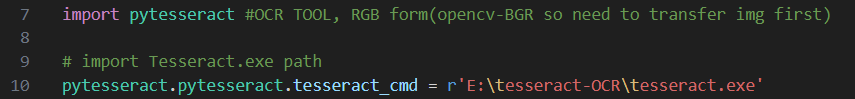


After getting the warped image, convert it to grayscale, binary, and blur again.

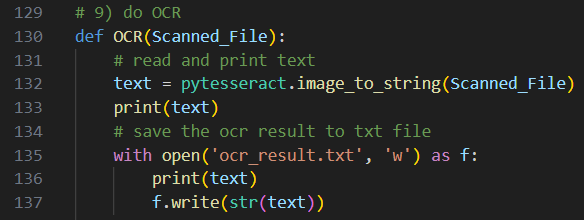
1. Text detection system:

The OCR tool used here is Pytesseract.

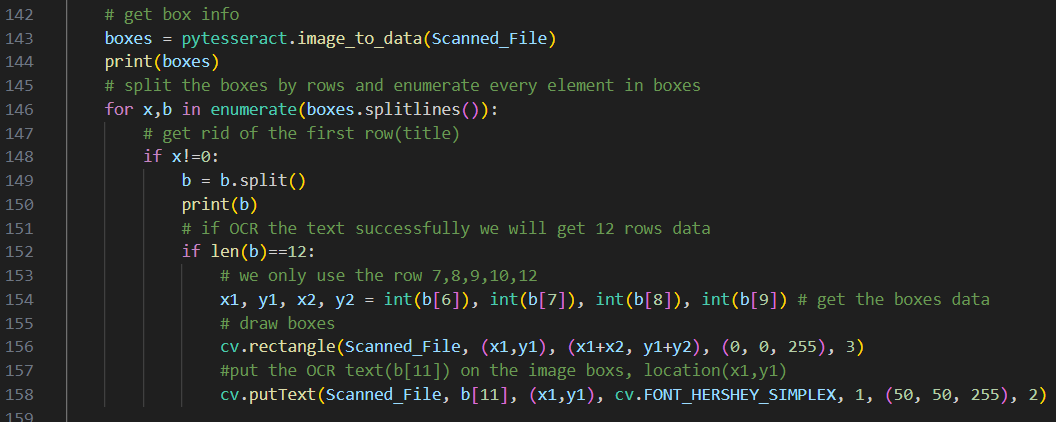
This is the text detection part:



First, load the tool.



Then use the by tesseract.image\_to\_string() to get the text from the input image, and save the OCR result in the txt file.



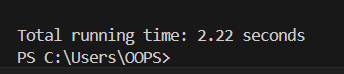
Finally, use the function pytesseract.image\_to\_data() to get the data, and use the data in rows 7,8,9,10,12 to get the location of the boxes. Then draw boxes on the image and put text nearby.

**Result analysis**

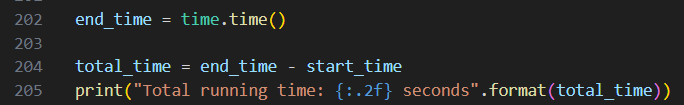
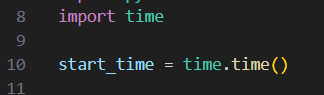
Image preprocess result:



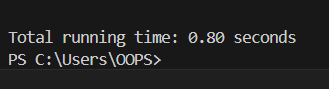
1. Speed:



The code used to calculate the runtime of the program is:

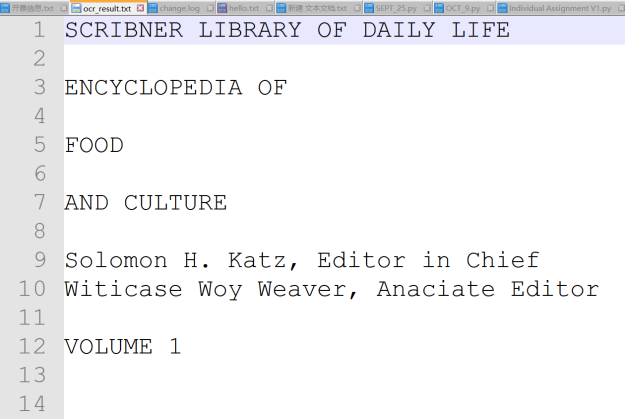


If you don’t want to show the preprocessed image, and only want to get the OCR result’s TXT file, the runtime will be shorter without clicking to close the image displaying windows.



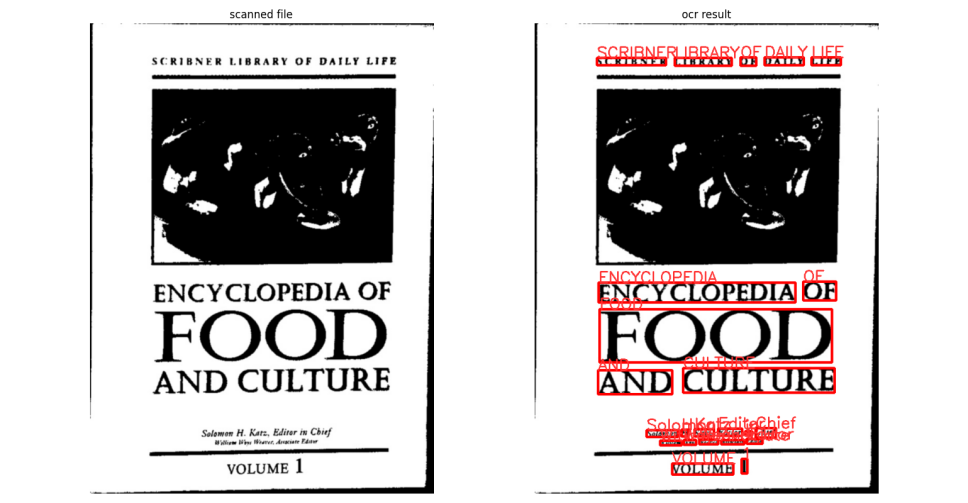
1. Accuracy

The OCR output text file:



Most of the words are correct, except the “Williams”,”waye” and “Associate” in line 10. This is because they are too thin, and some letters become too blurred after the perspective transformation.

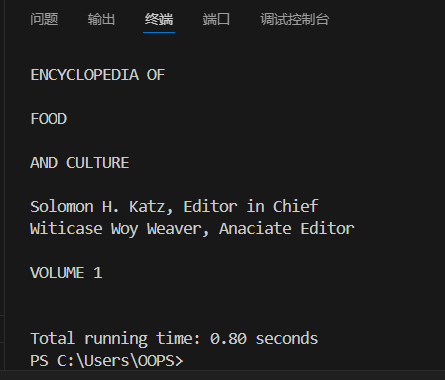
1. Boxes drawing



All boxes are in the right places, no misdetection.

1. Text extraction and displaying

The text is output in a txt file and print in terminal:



**System’s strength and weakness:**

1. Strength

I have tried many times, and the output OCR result is very stable,, approximatly fast, the accuracy is OK, most of the words are correct..

1. Weakness

It is inconvenient to change the variables in the code. If you want to test the code on other sample pictures, you need to rewrite some code in the file.

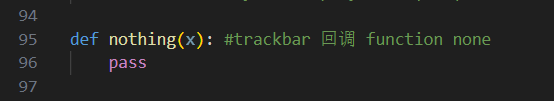
1. What can be improved:

It will be better to change the variables while the code is running so that people can see the result as they change the variables.

So I wrote the second version of this code, you can use the trackbar to adjust the variables during the image preprocessing part.

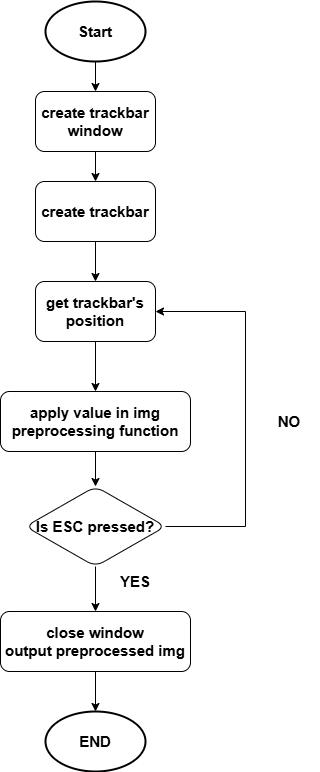
**Version 2**

In this version, I add the trackbar to let people change the variables in the image preprocessing part. Also, I added erode and dilate functions to help get rid of the noise in the image.

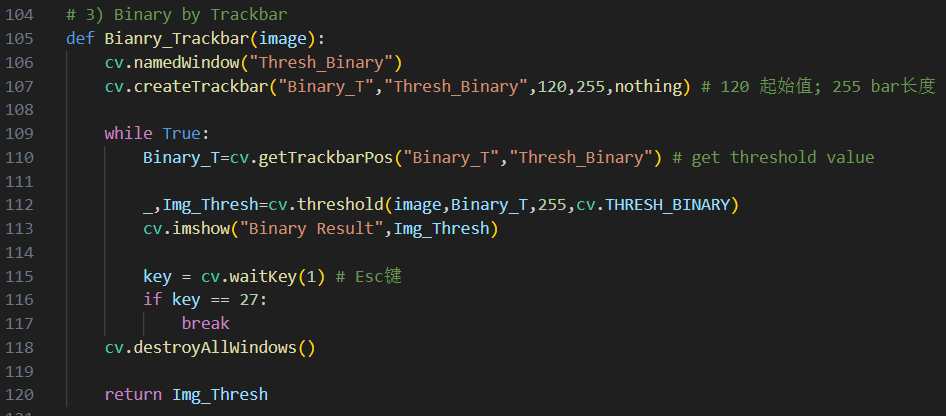


Above is the callback function for the trackbar.

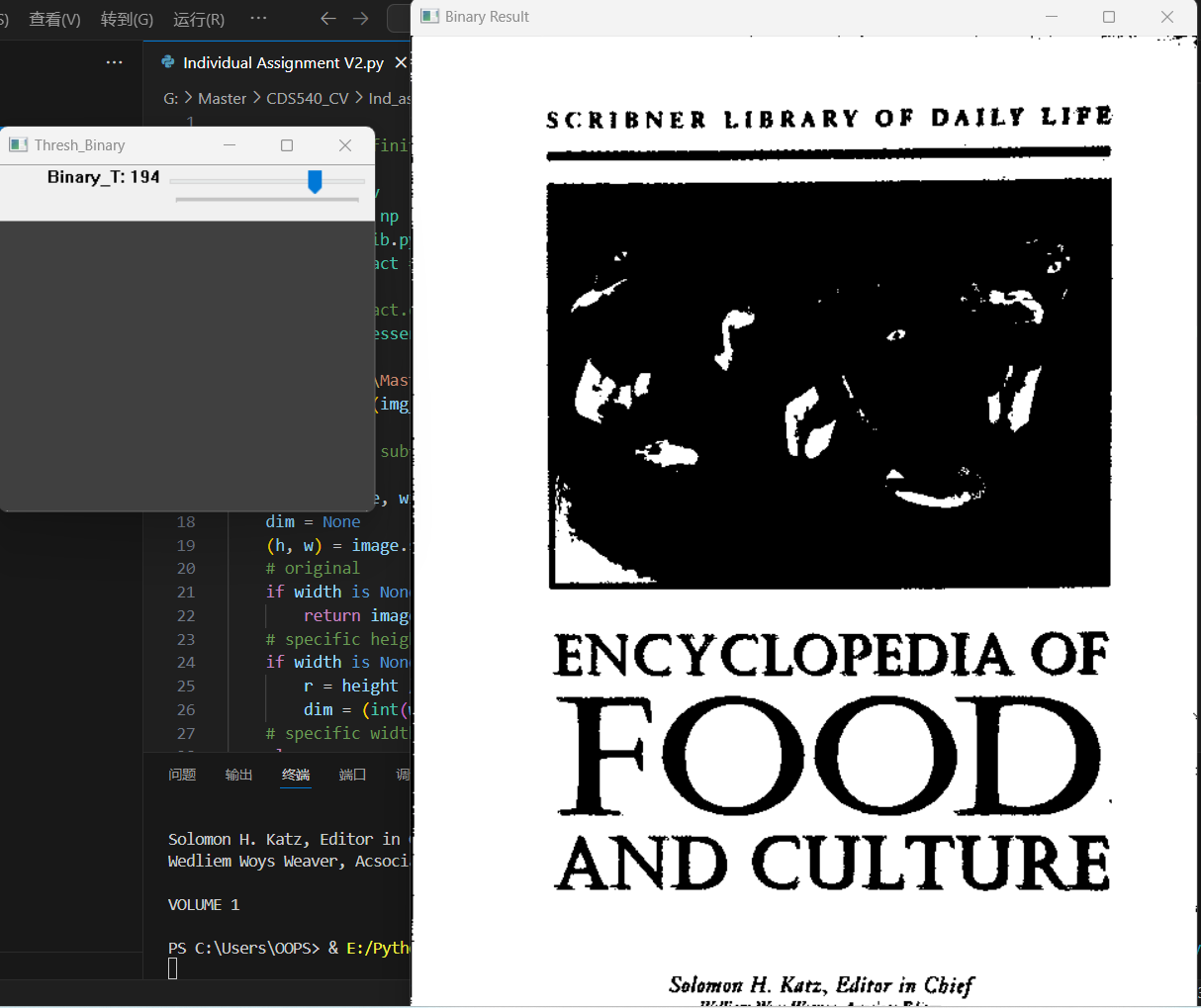
The image preprocessing function changed too. Instead of using fixed variables in the function, the user can use a trackbar to find the best value of the variables making the preprocessing part a little bit more practical. The steps to make a trackbar into a preprocess function are as follows:



The following is an example code of the trackbar used in the function:

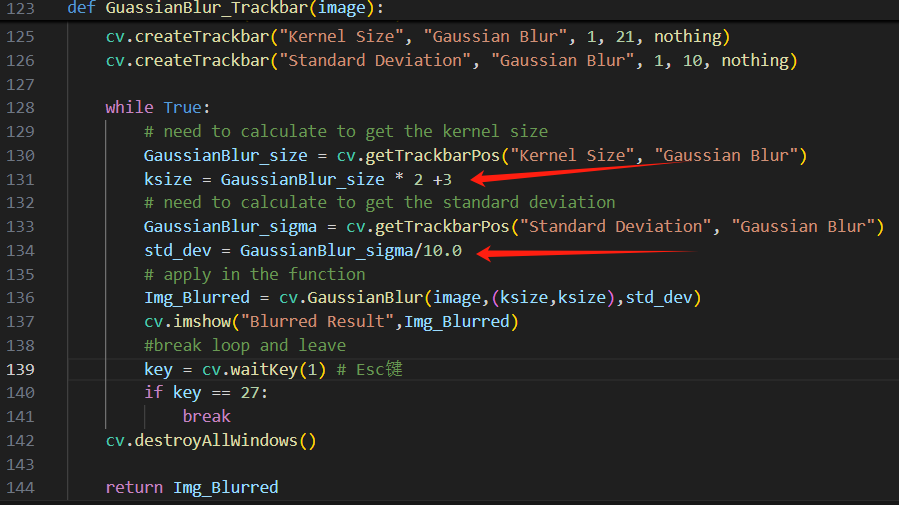


The result is:

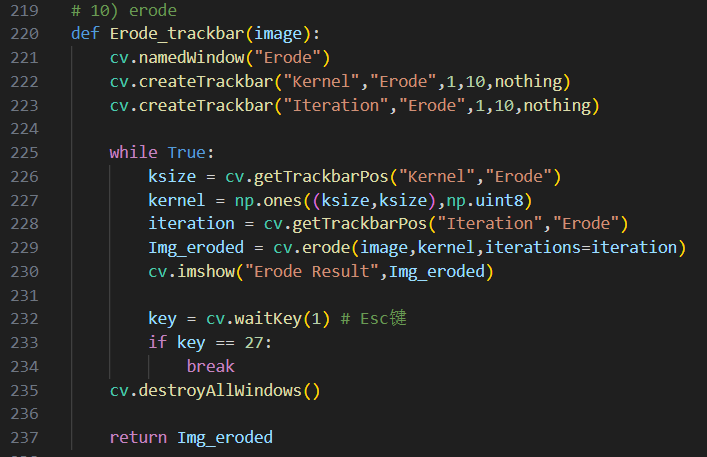


I added a trackbar for other image preprocessing subfunctions too.

It is important to notice that the value from the trackbar’s position cannot be used in the GaussianBlur function directly. The kernel size and standard deviation need to be calculated before being used in the function.

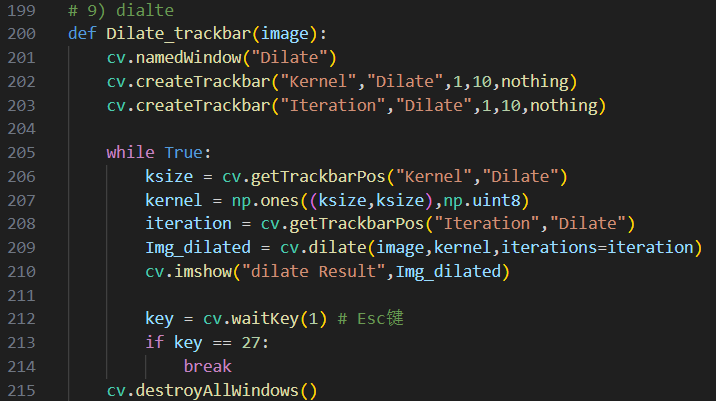


Version 2 has erode and dilate functions:



erosion can eliminate white noise, and disconnect 2 closed objects. The function for erosion is cv2.erode(src, kernel, dst=None, anchor=None, iterations=None, borderType=None, borderValue=None)

The dilate function:



Dilate enlarges the white part of the image.

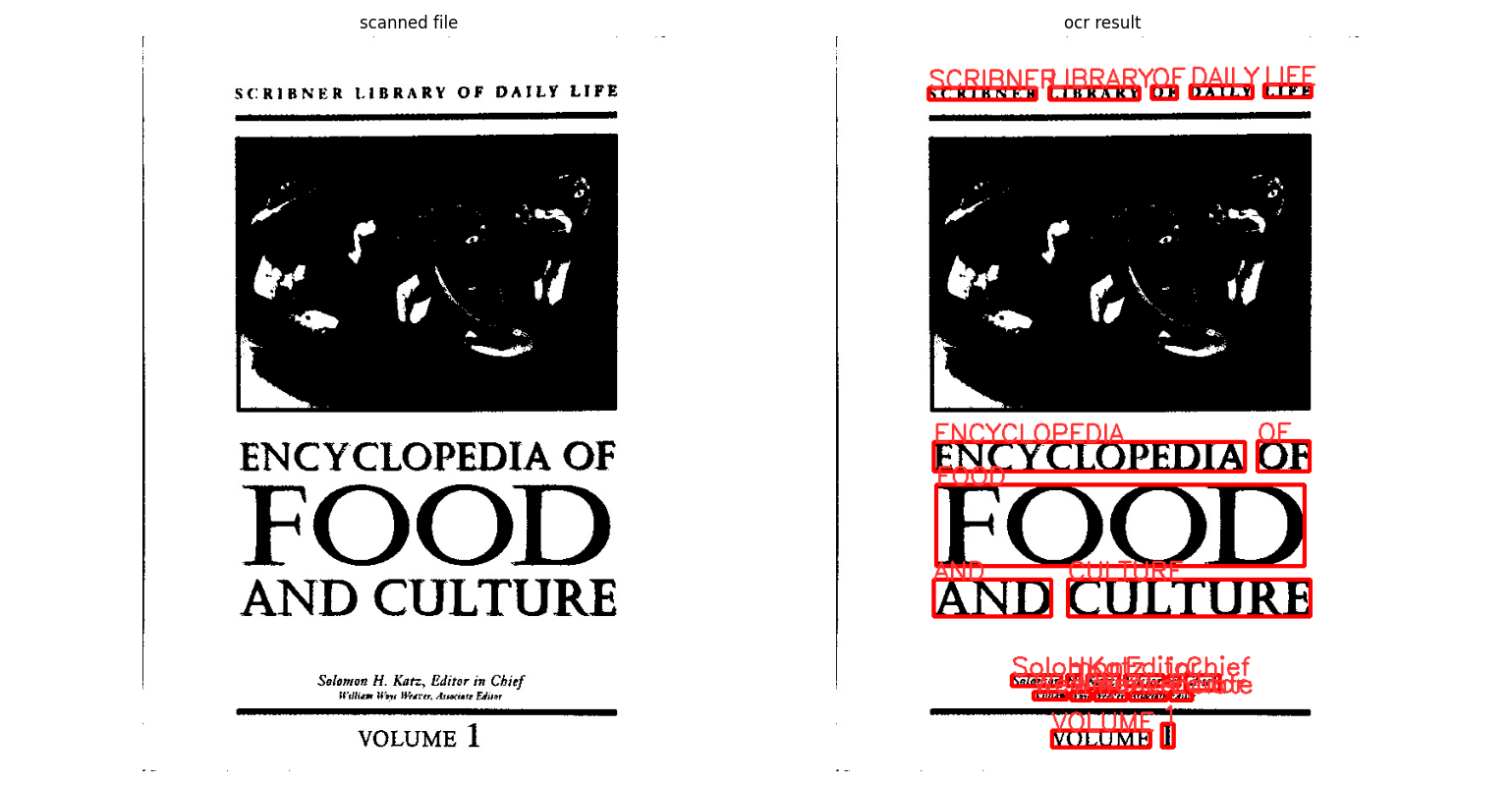
I use the erode function first and then the dilate function to minimize the noise.

1729189901500

The overall image preprocess result is:



The OCR result:



Print the output text file:

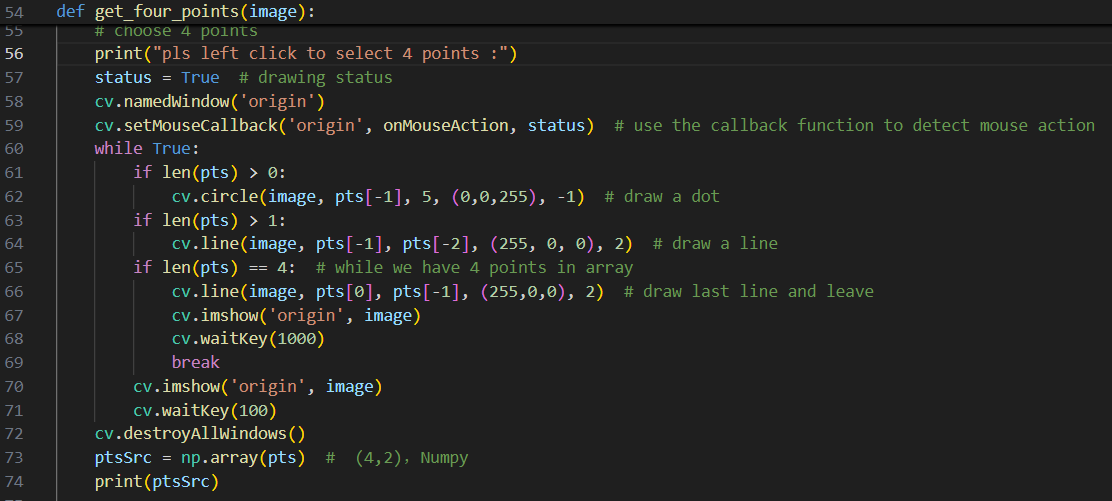


The result is similar to version 1, but the operation of variables is much more convenient.

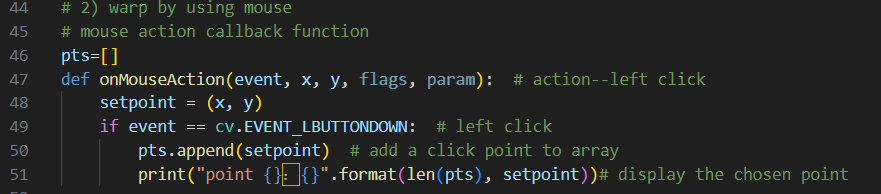
**Version 3**

In version 2 code, we can use the trackbar to adjust the variables of the image preprocess subfunction. But this program is still not so practical when we want to use it on other sample pictures. So, in version 3, the ROI selection and perspective transform by mouse selection have been added to the code.

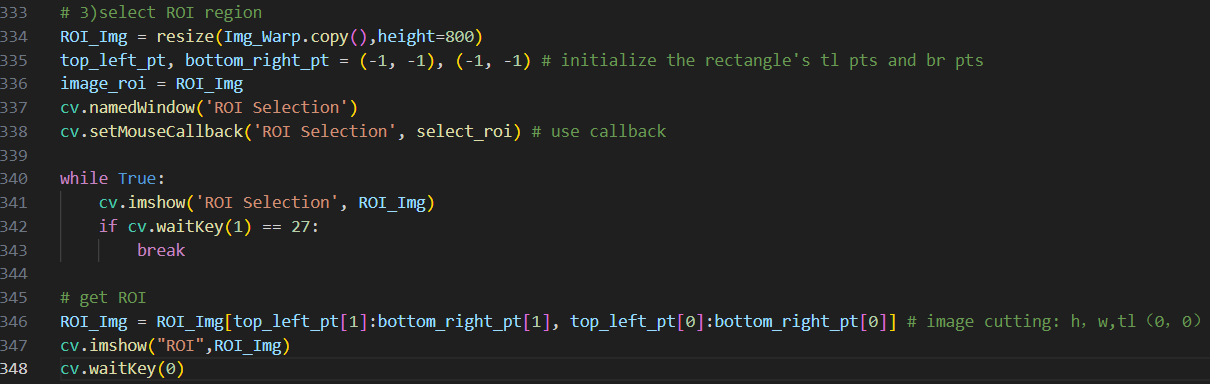
In the perspective transform part, the user can use the mouse to get the four points for transformation instead of using the find contour function.



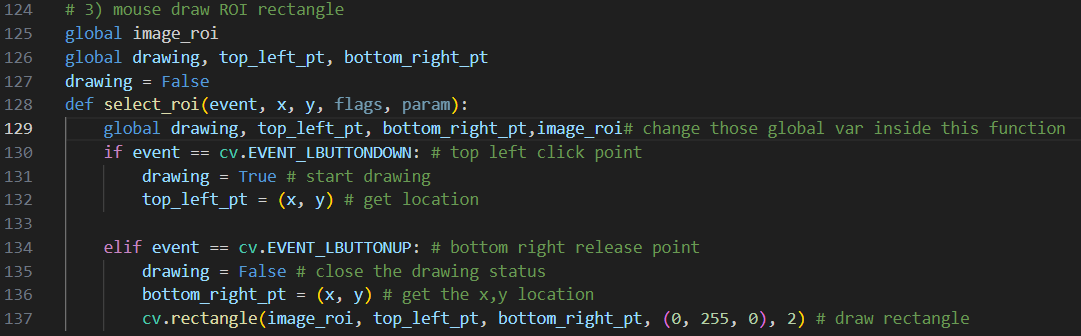
This is the mouse callback function for the left click to get four points:



The ROI selection part:



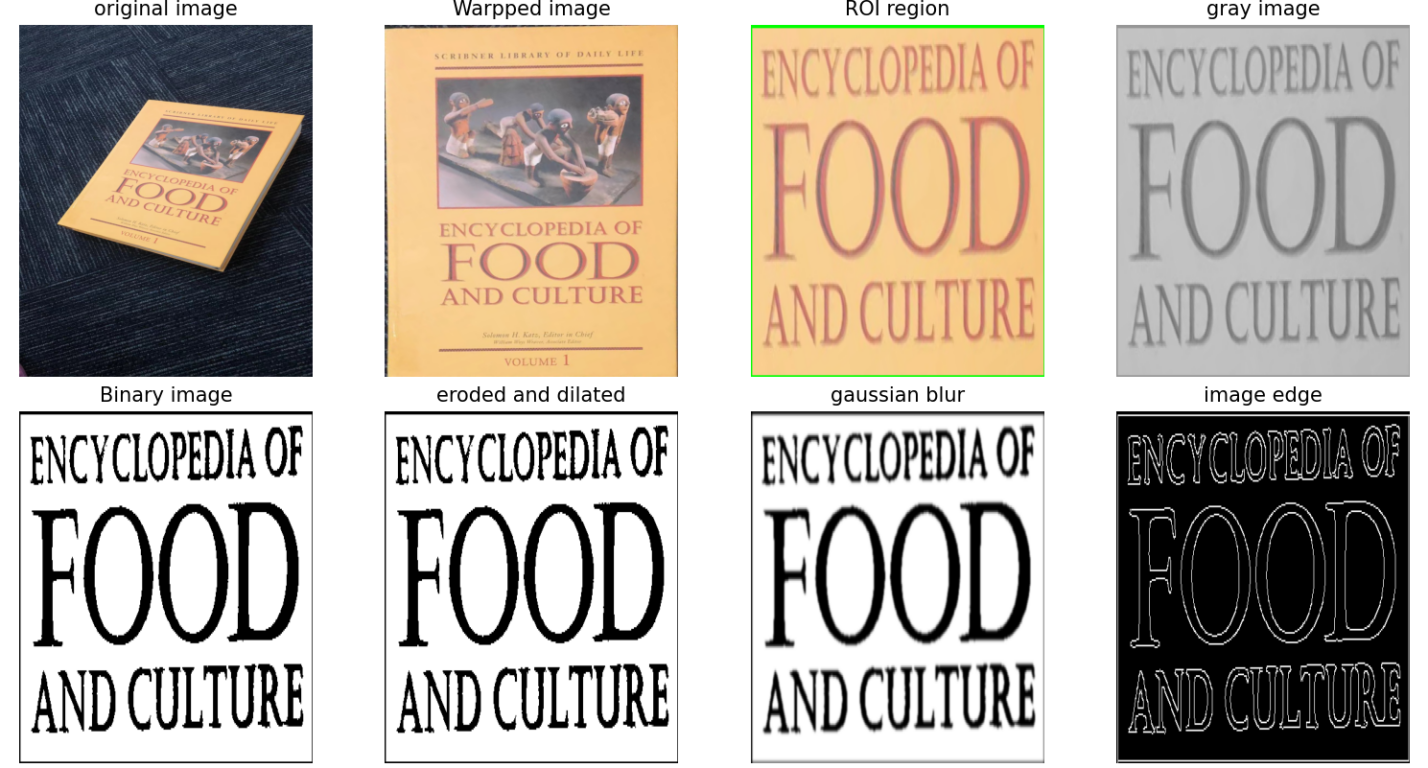
The mouse call-back function for pressing the left click and releasing the left click



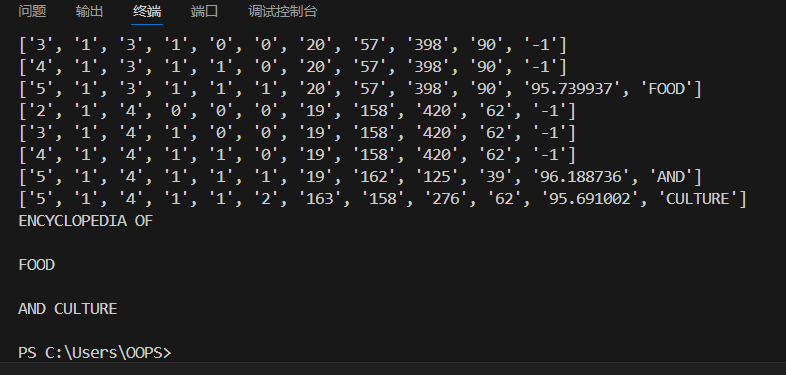
So, the overall steps in version 3 are:



And the result is:



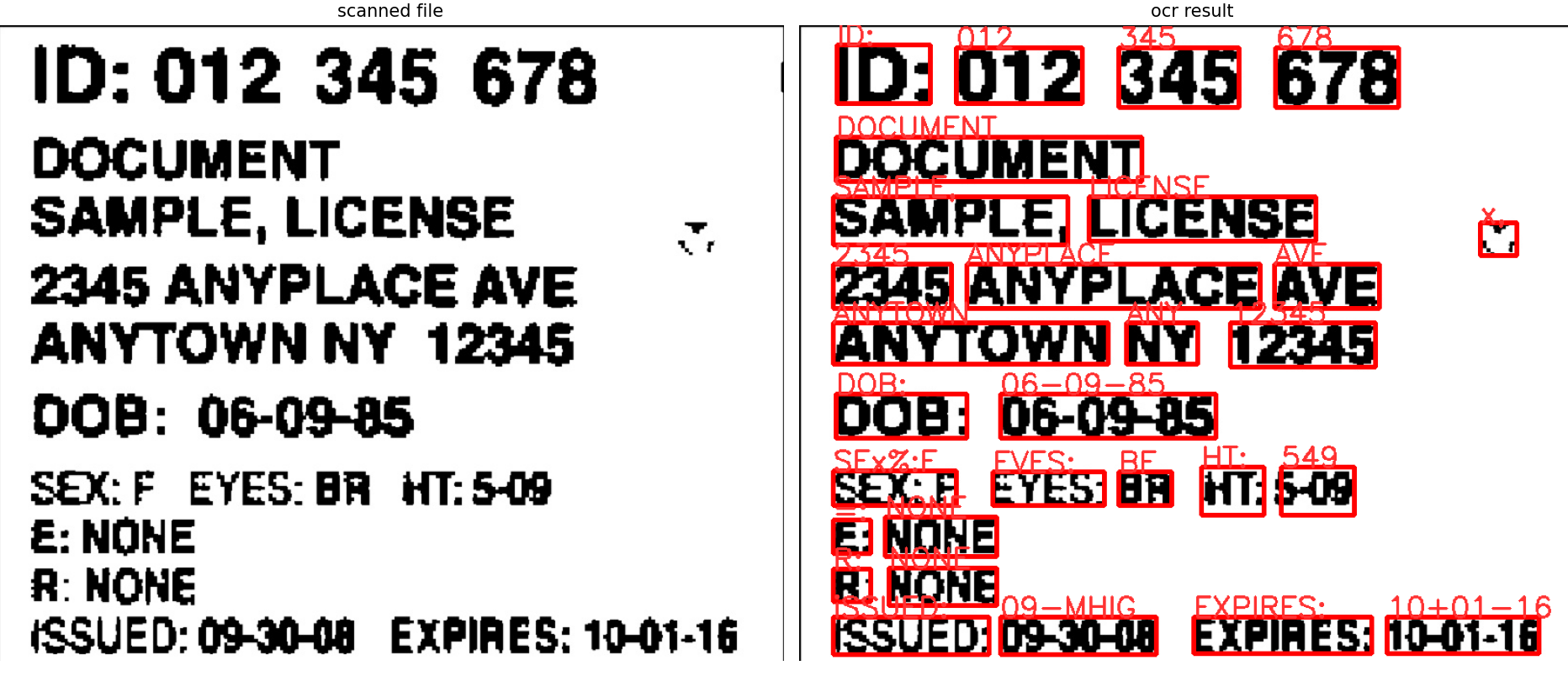


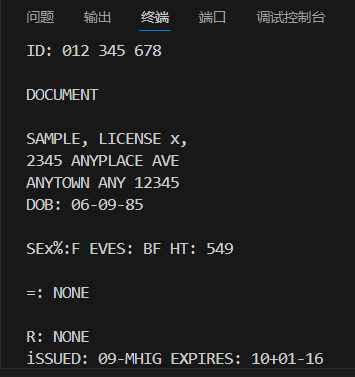


The result is accurate, and the boxes are in the right places. The third version is more flexible than the first two, but it needs the user to adjust the variables, to choose the ROI, and to click the points for perspective transformation.

Here is the version 3 test on other image samples:







Version 3 code can be used on many sample images, the result depends on the variables and points the user chooses to apply. If the User handles the trackbar and mouse precisely, the result will be better. Due to the multiple times of resize and the perspective transformation, the image lost so many pixels that some words became too blurry to be detected. There are some ways to improve the result, for example, using the pdf image, or taking a clearer picture. Use Qt to make a better OCR program so you can choose different combinations of functions to process the image.

**Conclusion:**

In this assignment, I wrote a program to preprocess an image of a book cover and use the OCR tool to detect the text on it. I use Python and OPENCV, Pytesseract to get the result. The main idea is to get rid of the noise and make the image clear enough to get detected by OCR. First, I use the displaying function in the OPENCV library to load the source image, then use the resize subfunction to resize the source image, so that it can be displayed in a suitable size. Then I use some image processing functions from OPENCV to smooth the image to get the clear edge of the image. The largest contour can be found based on the edges and so are the four points for perspective transformation. I apply the points into the perspective transformation function to get the transform matrix and the image after transformation. The new image needs to be processed to be smooth and clear enough that the OCR tool can read the text from it. Finally, I use the OCR tool to detect the text and draw boxes around each word, output the result into a txt file, and print the txt file in the terminal. The speed of this program is around 0.8 seconds, and the result of OCR detection is mostly accurate. The boxes are drawn in the right place with text labeled nearby. In conclusion, version 1 fulfills the basic function.

In version 2, I add trackbars to adjust the variables in the image preprocessing part to view the result while adjusting the variables. In this way, it is very easy to know whether the variables are suitable or not. Also, the erode and the dilate function are added in this version. The result of version 2 is similar to the result of version 1.

In version 3, I used mouse action to do the perspective transformation instead of using the contours. Mouse action is also used in ROI selection. In this way, version 3 can be used on more sample images, so it is more practical. However, the downside is the decrease in efficiency and accuracy. Because of the resize part and transformation part, the image loses a lot of pixels and becomes unclear.

The system can be improved in many ways. Making a better UI that users can use conveniently, and choose the combination of different image processing methods they like.