Document Scanning OCR

Group Member Names

Aryaman Jain 18BCE2037

Ankit Yadav 18BCE2026

Rakshit Kumar 18BCE0999

Faculty Name

Dr. CHIRANJI LAL CHOWDHARY

Associate Professor, SITE, VIT Vellore

Winter Semester

December 2019 - June 2020

18BCE2037

We’ve uploaded our video review on the link given below.

https://drive.google.com/drive/folders/1-XwURA1z4X2TaHK48UWAZGgEL3asvdZ9?usp=sharing

SYNOPSIS

This project is about Optical Character Recognition. it's a process of classifying optical patterns with reference to alphanumeric or other characters. Optical character recognition process includes segmentation, feature extraction and classification. Text capture converts Analog text based resources to digital text resources, then these converted resources are often utilized in several ways like searchable text in indexes so on identify documents or images. because the primary stage of text capture a scanned image of a page is taken. And this scanned copy will form the thought for all other stages. The very upcoming stage involves implementation of technology Optical Character Recognition (OCR) for converting text content into machine understandable or readable configuration. OCR analysis takes the input as a digital image which is printed or hand written and converts it to computer readable digital text format. Further OCR processes the digital image into smaller components for analysis of finding text or word or character blocks or occurences. And again the character blocks are further broken into components and are compared with a dictionary of characters. Thereafter, the text within the image is then displayed & the user gets the output which can be used for further requirements. Tools used are python libraries like tesseract , opencv & numpy. Where Tesseract is an optical character recognition engine for various operating systems which makes the work easier & eases the problems . in conjunction with that OpenCV (Open Source Computer Vision Library) which is used as an open source computer vision and machine learning software library which makes

the image processing part easier because it can read & write images, detect shapes and texts like reading number plates. It also supports multiple languages which makes it effective & easier to implement. Optimized algorithms are used to process the image & give the results.

18BCE2037

WHAT YOU OR YOUR GROUP HAS DONE IN REVIEW 0 (FIRST)?

For review 0(first) the title of the project that we submitted was “EFFICIENT CHARACTER SEGMENTATION APPROCH FOR MACHINE-TYPED DOCUMENTS” along with a base paper,at that time we had 4 members in our team , but in the final we were only 3 left.

General Character Segmentation Approach for Machine-Typed Documents

Vladan Vučković, Boban Arizanović

***Abstract*—In this paper one important entirety in most OCR systems is discussed. The main goal is to explain the approach for segmentation of the individual characters from machine-typed documents, which is based on different image processing techniques in spatial domain and exploits the features of a document structure. The paper gives the overview of the proposed technique by the phases, starting from the grayscale conversion and binarization, through the line, word, and character segmentation of a document image using the modified projection profiles technique, as well as the description of the problems which can appear. The approach is semi-automatic since it requires adjustment of the thresholds for the different segmentation parameters. The numerical results which show the proposed approach performances from the perspective of the time complexity and segmentation accuracy are provided. Results show that this approach outperforms the state-of-the-art methods in all aspects. Furthermore, it is shown that the proposed character segmentation technique is suitable for real-time tasks.**

***Index Terms*—Image Processing, OCR, Character Segmentation, Machine-Typed Documents.**

I. INTRODUCTION

Character segmentation still represents very challenging task in image processing and other related fields. It is a crucial pre-processing part in most OCR systems [1,2], and combined with character recognition [3] represents an important subject of research for a long period of time [4]. The process of character segmentation is usually unduly underestimated comparing to the process of character recognition [5,6]. Some approaches deal with extraction of characters from natural and other non-document images [7,8], and others deal with character segmentation on document images. The second group is usually divided into machine-printed documents [5,9,10,11], where the shape of document elements is regular, and handwritten documents where character segmentation is challenged due to irregular document structure [6,12,13,14,15]. Machine-typed documents are especially important because of historical documents [3,16,17].

All stages in character segmentation have been a subject of research. One of the most important problems which can appear in the process of character segmentation is a document skew and a variety of works deal with this problem. Many skew estimation approaches are modifications of the Hough transform [18,19], and some of them are based on correlation functions or straight line fitting [20,21]. Analyses on image binarization parameters proved the superiority of Otsu method and other Otsu-based methods [22]. Gaussian low-pass filter and innovational Laplace-like transform can be used for character segmentation [23]. Segmentation can be also performed using the Bayes theorem, in order to ensure the usage of the prior knowledge [24]. One character segmentation approach for typewritten historical documents is based on the usage of the horizontal projection profile of each word segment [16]. Adaptive run length smoothing (ARLSA) algorithm for segmentation of historical machine-printed documents proved to be better than state-of-the-art methods [10]. Diverse approaches for segmentation of handwritten documents are proposed [13]. A clustering based method can be exploited in the process of segmentation [25]. Gabor filter can be exploited for feature extraction and Fisher classifier for feature classification [26]. In order to solve the problem with touching characters, self-organizing maps, SVM classifiers, and Multi-Layer Perceptron are used [12,14,27]. For natural images, some approaches are based on tensor voting and on the usage of the three-color bar code for segmentation [8,28].

This paper presents general character segmentation approach for machine-typed documents and provides experimental results for time complexity and segmentation accuracy. This technique can be also used for processing of machine-printed documents. The proposed approach uses the modified projection profiles technique which exploits the spatial features of a document structure in the segmentation process. Provided numerical results show that this approach generally gives better results than state-of-the-art methods. For the purpose of testing the approach performances, historical ground-truth machine-printed documents are used. It should be mentioned that this approach is designed as a part of a real-time OCR system for the needs of the “Nikola Tesla Museum” [29].

This paper is organized as follows: in section 2 the complete representation of the proposed character segmentation approach is provided. In section 3 the approach performances are analyzed using the set of ground-truth historical machine-printed documents and comparison with the state-of-the-art approaches is given. Finally, the discussion about the performances and the future work is given in section 4.

II. CHARACTER SEGMENTATION APPROACH

The approach consists of several steps and each step is explained in details. Image processing techniques in spatial domain for image binarization and noise reduction [30,31] and histogram processing methods [15,16,32], are used in the development of the proposed approach. The original

Vladan Vučković is with the Faculty of Electronic Engineering, University of Niš, 14 Aleksandra Medvedeva, 18000 Niš, Serbia (e-mail: vladanvuckovic24@gmail).

Boban Arizanović is with the Faculty of Electronic Engineering, University of Niš, 14 Aleksandra Medvedeva, 18000 Niš, Serbia (e-mail: bobanarizanovic@hotmail.com).

Proceedings of 4th International Conference on Electrical, Electronics and Computing Engineering, IcETRAN 2017, Kladovo, Serbia, June 05-08, ISBN 978-86-7466-692-0

pp. RTI2.2.1-6

The problem statement that we wanted to solve was OCR (Optical Character Recognition) is a technology that can easily scan and extract the text from document images or business cards images, then convert them into editable and searchable files. You can get information from captured images and solve the data entry problem without having to retype it.

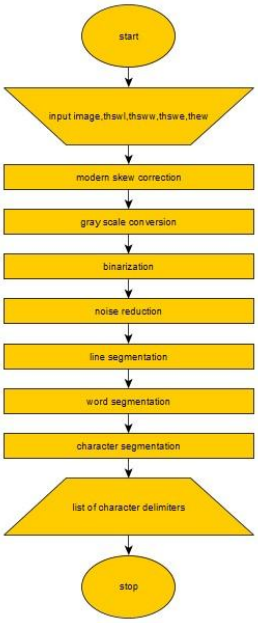
And for the same we proposed the following initial method:

The algorithm consists of three main stages: 1) Manual skew correction, using a new general, ultra-fast architecture for geometrical image transformations, 2) Image filtering, which represents a pre-processing stage and uses literature algo- rithms for image binarization, and noise reduction 3) Segmentation logic, which uses histogram processing in the process of determining character positions in the given document image. The inputs are a document image and threshold values which are used in order to control the segmentation process, since the proposed character segmentation approach is semi- automatic

18BCE2037

WHAT YOU OR YOUR GROUP HAS DONE IN REVIEW 1 (SECOND)?

For review 1(second) building on our project by this time we were able to convert images into grayscale images and using those grayscale images for edge detection using the following algorithm.



After we concluded the review we were asked a bunch of questions reguarding the functioning of the machine and how are these images converted as so, we were also asked to add some extra features in the machine so that it ways out from the others that are already existing in the market. After this we all decide that instead of providing the image to the machine why not we make it in such a way that we are able to scan the document and then convert it into text.and making it flexible to every situation

18BCE2037

REVIEW 3 (LAST)

TABLE OF CONTENTS OF PROJECT

1. **INTRODUCTION**

Optical character recognition (OCR) is the electronic identification and digital encoding of typed or printed text by means of an optical scanner and specialized software. Using OCR software allows a computer to read static images of text and convert them into editable, searchable data. OCR typically involves three steps: opening and/or scanning a document within the OCR software, recognizing the document within the OCR software, then saving the OCR-produced document during a format of our choosing. OCR are often used for a spread of applications. In academic settings, it's oftentimes useful for text and/or data processing projects, also as textual comparisons. OCR is additionally a crucial tool for creating accessible documents, especially PDFs.

a.MOTIVATION:

The paper presents a solution of re-writing the text composed in an image. Implementation of the work is done using Python & it’s libraries such as tesseract & openCV. Since, it is time-taking to write down text from an image our

application scans the image, saves it & turns it into text therefore relieving the user of the hassle. Hence, the innovativity & speed is the motivation behind this.

Contribution by RAKSHIT KUMAR

- Took the liberty to create some functions required for image processing & implementing them in the main code , starting from converting the original image into grayscale so that we have to deal with less pixels. Functions created by Aryaman were implemented by me in the main code and were called at the required moments. It took a great deal of time to implement the function of finding the biggest contour as errors kept popping up , so the function was re-defined by me to solve the complications & make it subtle.Further I removed the extra unwanted pixels from the sides to ensure that only the document is selected & not any part of the surface it is kept on. The labels for display is then called when the image is processed to show us the stages of digital image processing using OCR.

Contribution by ARYAMAN JAIN

- I and Rakshit have implemented the document scanning section of the project
- I have made all the functions that are later called by rakshit for the image conversion and saving the image.
- The following functions are as follows:
- 1) stacking function : as we have may outputs in the project all when executed cluttered the screen so using this all the individual results are collected in one single output.
- 2) biggestcontour: as the name suggest this function is used to find the biggest contour(rectrangle in this case) in the image , it uses area as its measure
- 3)draw rectangle: this function basically assigns the coordinates to the biggest contour

- 4) initializetrackbars:this function is used to get a trackbar that allows is to sdjust the threshold values as required.
- 5)valtrackbar: this function saves the trackbar values and return them for further processed

Contribution by ANKIT YADAV

- I have implemented the saving function, when the key S is pressed, the final image will be saved in a scanned folder (named Scanned) and displays a text message that is scan saved on result screen. Then this scanned image is sent to the tesseract library.
- Then I have implemented the tesseract library, when the saving function is called the image that is stored in the scanned folder is called by Tesseract library. And all the internal processes like segmentation and all occurs and finally the output result is shown.

2. LITERATURE SURVEY

a. ALSO A TABLE OF YOUR LITERATURE SURVEY

Reference	Methods Used	Evaluation	Merits and Demerits
https://docs.opencv.org/4.3.0/d0/d05/group__cudaimgproc.ht	CannyEdgeDetecter TemplateMatching Segmentation Color Processing	Finds edges in an image using [36] algorithm Converts Colored to Grayscale Enhances text	Fast but occupies lots of space

ml			
https://numpy.org/doc/stable/reference/arrays.ndarray.html	ndarray	Arranges pixel in an array Is used for further calculations	Fastest, no demerits
https://tesseract-ocr.github.io/tessapi/4.0.0/	setRectangle setThresholder tesseract()	Creates contours Thresholds the image to enhance the text	Errors can occur such as 2% undetected characters Sometimes characters are disfigured.

3. BACKGROUND OF YOUR PROJECT WORK

The whole project is divided into 2 segments first being the scanning of the image and second being the conversion of image to text:

Segment one:

Here the original image is being converted into a scanned document this process in further divided into sets , first the image is converted into a grayscale image , then this gray scale image

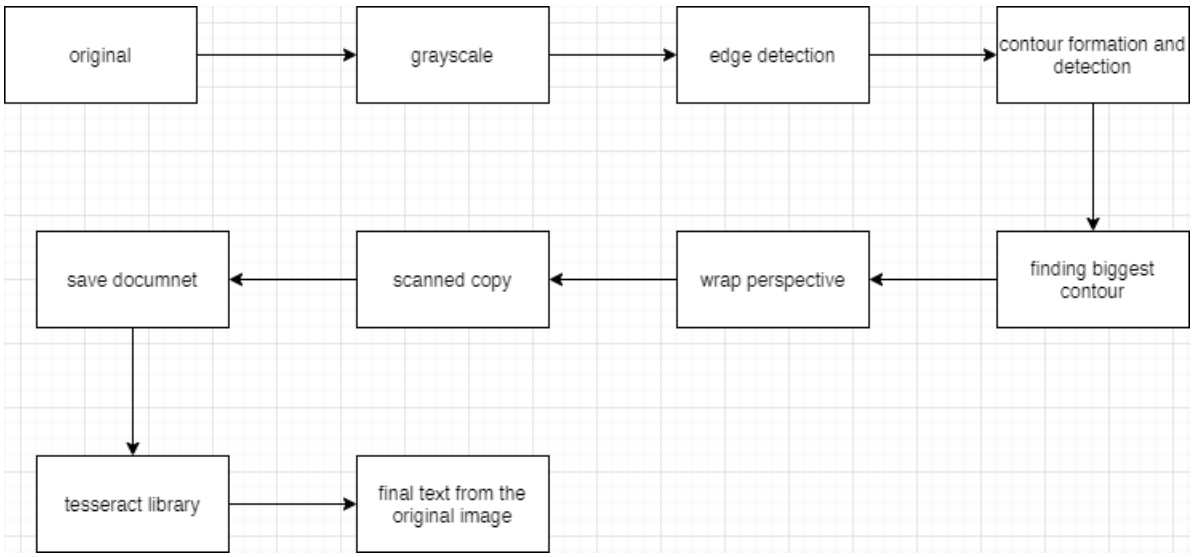
is used for the edge detection (using the canny edge detection function) around the text here we use threshold values which are adjustable manually to get better results , next we need to detect the size of the paper so we use contour detection further we find the biggest contour in the image hopefully that is the paper you want to scan ,then coordinates are assigned to the corners of the paper ,then we need to apply wrap perspective so that the orientation of the document however it may have been scanned comes out as 90* next an adaptive threshold is applied so that the scanned document has a scanned doc type feel. Now we have implemented a save function in the code where if we press “s” then the image is saved in the saved folder.

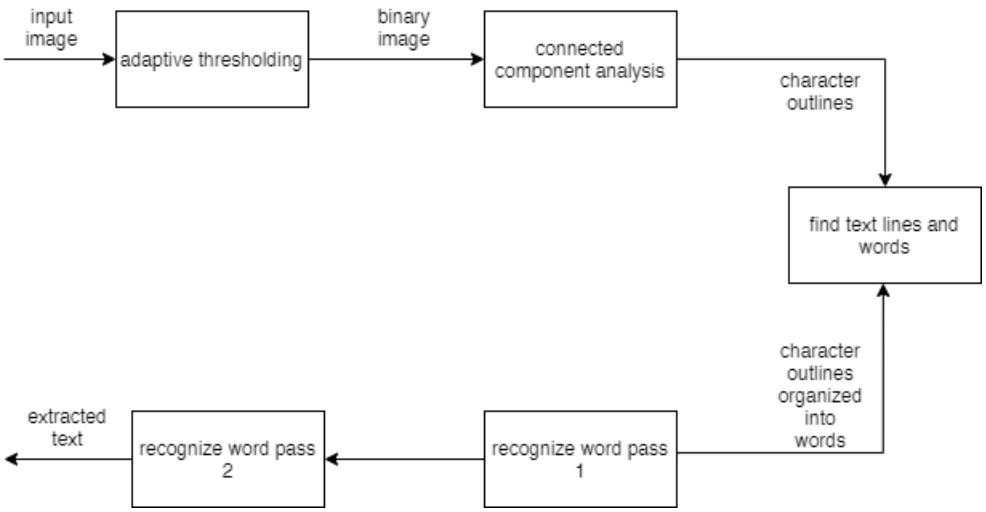
Segment two:

Here the image that we have scanned or saved will undergo the OCR process, we have implemented the tesseract library which converts the image to text.

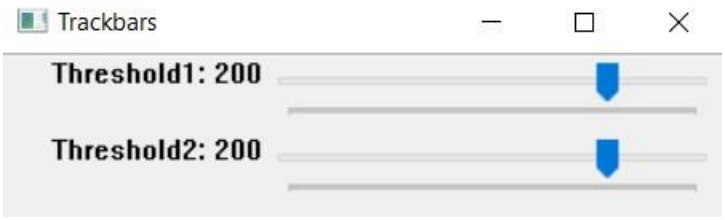
When both of the segments work as one the second the image is loaded in the machine and the thresholds are adjusted the scanned image is produced and once saved it gives out the text from the image.

4. Proposed Work

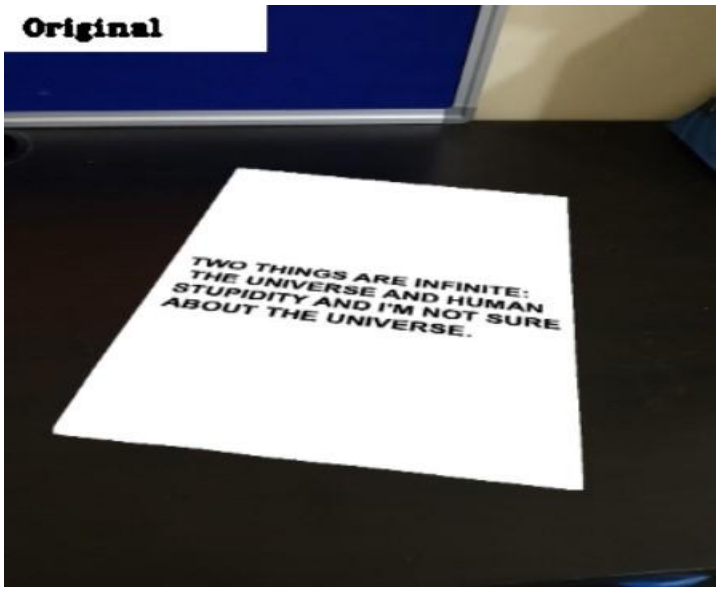




5. Evaluation and Results Analysis

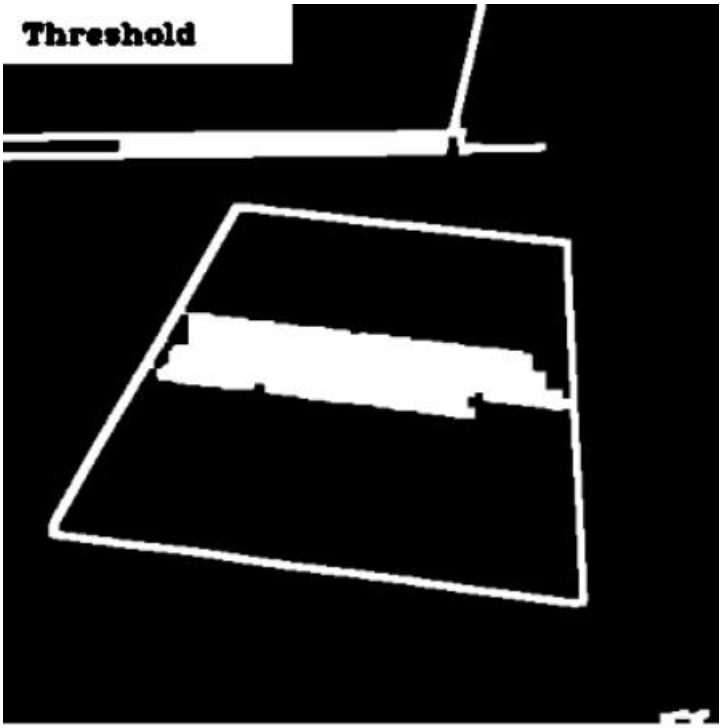


threshold trackbar

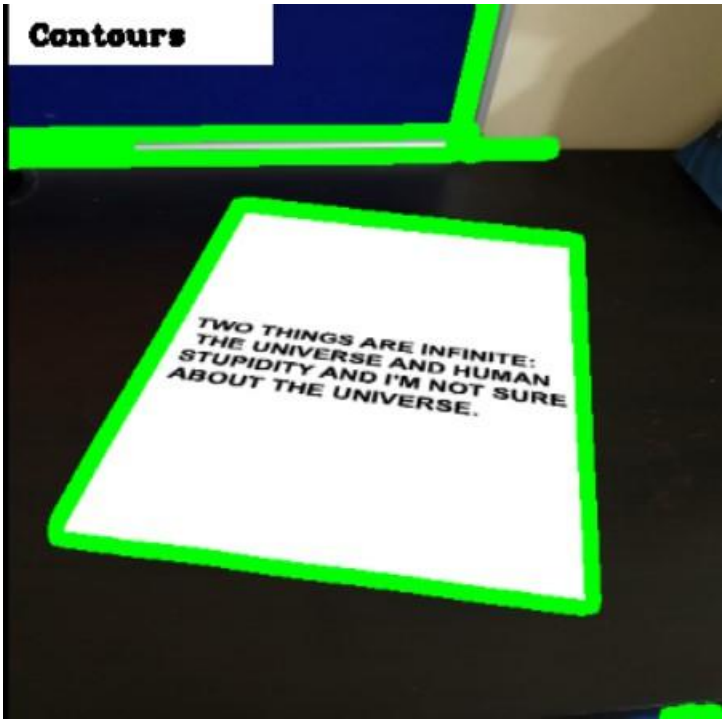


original image

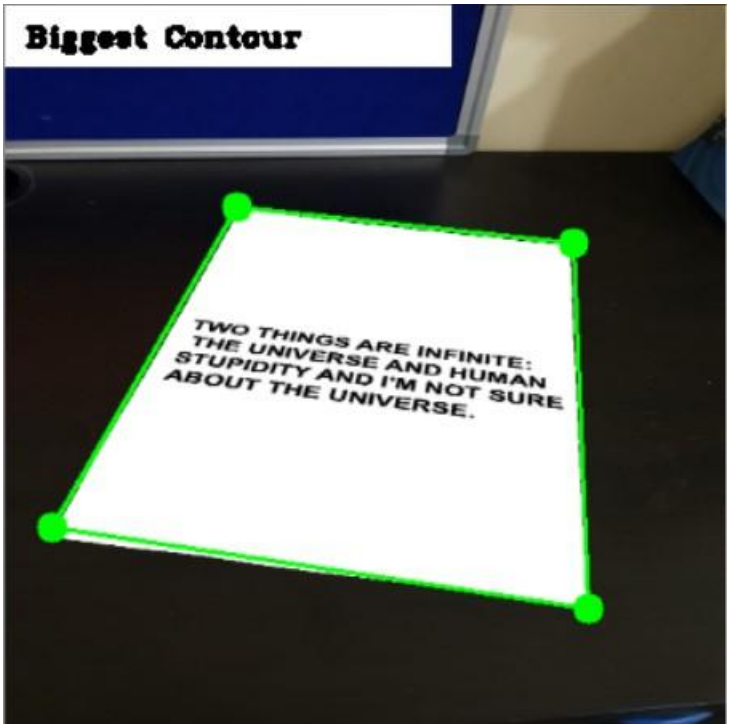
images converted to grayscale



edge detection via thresholds



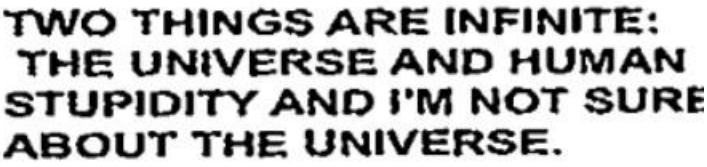
contour detection



getting the biggest contour in the image

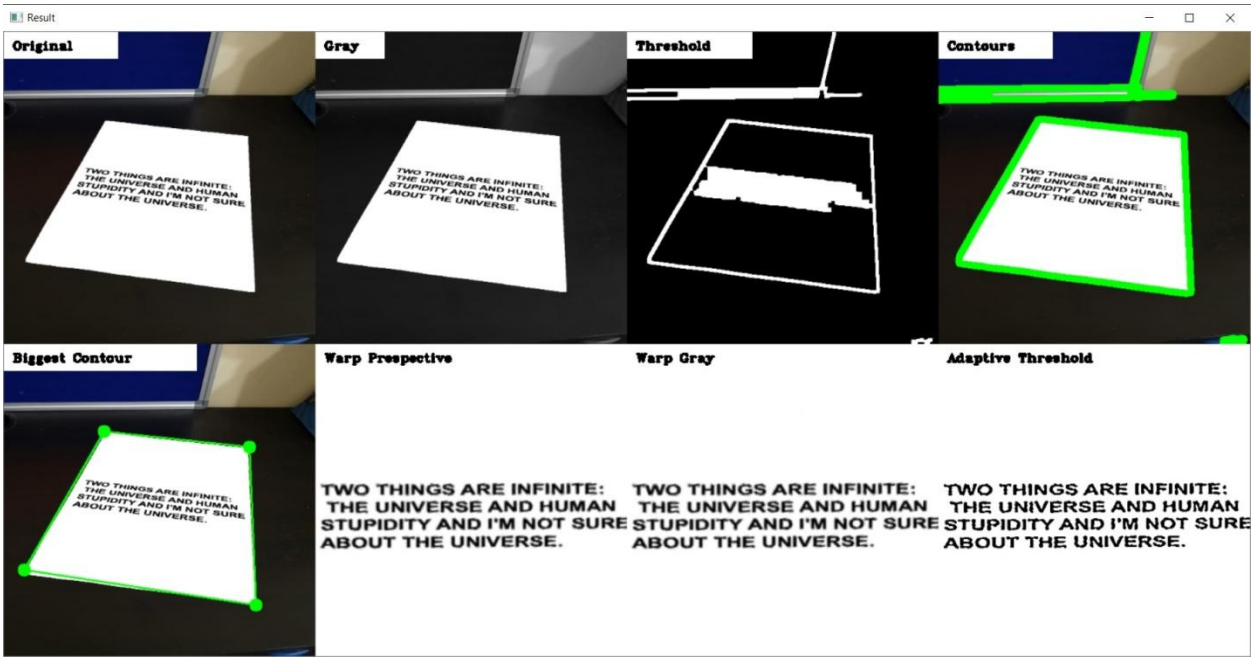
oriented image via warp prespective

Adaptive Threshold



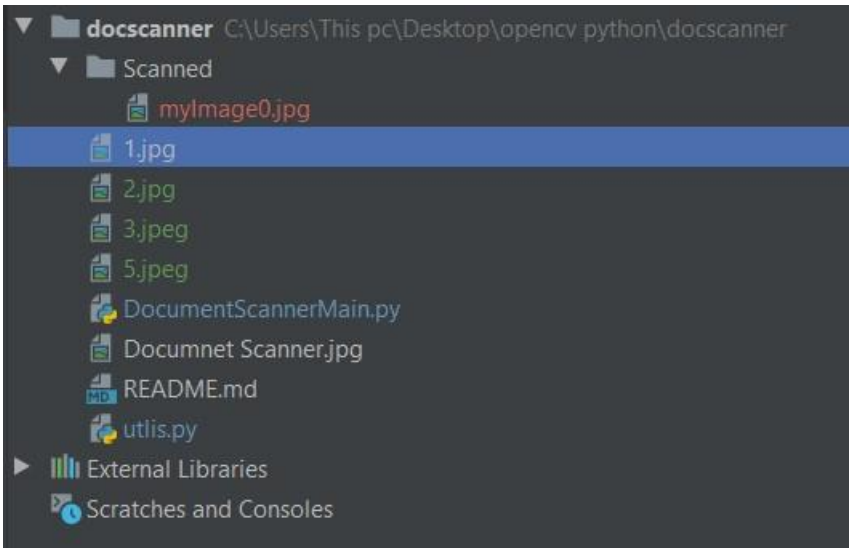
scanned copy

a

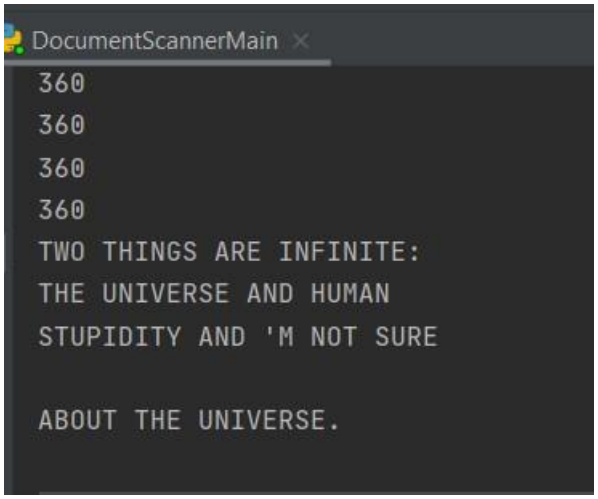


the main result (single result)

notification of saving image



saving of image in folder



final result of text

6. Code:

Main document scanner:

```
import cv2
import numpy as np
import utlis
import pytesseract

webCam = True
imageLocation = "5.jpeg"
cap = cv2.VideoCapture(1)
cap.set(10,160)
heightImg = 480
widthImg = 480

utlis.initializeTrackbars()
count=0

while True:

    imgBlank = np.zeros((heightImg,widthImg,3),np.uint8)

    if webCam:success, img = cap.read()
    else:img = cv2.imread(imageLocation)
    img = cv2.imread(imageLocation)
    img = cv2.resize(img,(widthImg, heightImg))
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #convets the image to grayscale
    imgBlur = cv2.GaussianBlur(imgGray, (5, 5), 1)  #converts the grayscale image to blured image
    thres=utlis.valTrackbars() #using this function we get the trackbars which help us adjust the threshold
    imgThreshold = cv2.Canny(imgBlur,thres[0],thres[1])
    kernel = np.ones((5, 5))
    imgDial = cv2.dilate(imgThreshold, kernel, iterations=2) # APPLY DILATION
    imgThreshold = cv2.erode(imgDial, kernel, iterations=1)  # APPLY EROSION

    ## FIND ALL COUNTOURS
    imgConts = img.copy()
    imgBigConts = img.copy()
    contours, hierarchy = cv2.findContours(imgThreshold, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE) # FIND ALL CONTOURS
    cv2.drawContours(imgConts, contours, -1, (0, 255, 0), 10) # DRAW ALL DETECTED CONTOURS


    # FIND THE LARGEST COUNTOUR IN THE FRAME
    big, maxArea = utlis.biggestContour(contours) # FIND THE BIGGEST CONTOUR
    if big.size != 0:
        big=utlis.reorder(big)
        cv2.drawContours(imgBigConts, big, -1, (0, 255, 0), 20) # DRAW THE BIGGEST CONTOUR
        imgBigConts = utlis.drawRectangle(imgBigConts,big,2)
        pts1 = np.float32(big) # PREPARE POINTS FOR WARP
        pts2 = np.float32([[0, 0],[widthImg, 0], [0, heightImg],[widthImg, heightImg]]) # PREPARE
POINTS FOR WARP
        matrix = cv2.getPerspectiveTransform(pts1, pts2)
        imgWarpColored = cv2.warpPerspective(img, matrix, (widthImg, heightImg))
```

```
        #REMOVE EXTRA UNWANTED PIXELS FROM THE SIDES
        imgWarpColored=imgWarpColored[20:imgWarpColored.shape[0] - 20,
20:imgWarpColored.shape[1] - 20]
        imgWarpColored = cv2.resize(imgWarpColored,(widthImg,heightImg))

        # APPLY ADAPTIVE THRESHOLD
        imgWarpGray = cv2.cvtColor(imgWarpColored,cv2.COLOR_BGR2GRAY)
        imgAdaptiveThre= cv2.adaptiveThreshold(imgWarpGray, 255, 1, 1, 7, 2)
        imgAdaptiveThre = cv2.bitwise_not(imgAdaptiveThre)
        imgAdaptiveThre=cv2.medianBlur(imgAdaptiveThre,3)

        # Image Array for Display
        imageArray = ([img,imgGray,imgThreshold,imgConts],
                      [imgBigConts,imgWarpColored, imgWarpGray,imgAdaptiveThre])

    else:
        imageArray = ([img,imgGray,imgThreshold,imgConts],
                      [imgBlank, imgBlank, imgBlank, imgBlank])

    # LABELS FOR DISPLAY
    lables = [["Original","Gray","Threshold","Contours"],
              ["Biggest Contour","Warp Prespective","Warp Gray","Adaptive Threshold"]]

    stackedImage = utlis.stackImages(imageArray,0.75,lables)
    cv2.imshow("Result",stackedImage)

    # SAVE IMAGE WHEN 's' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('s'):
        cv2.imwrite("Scanned/myImage"+str(count)+".jpg",imgWarpColored)
        cv2.rectangle(stackedImage, ((int(stackedImage.shape[1] / 2) - 230), int(stackedImage.shape[0] / 2)
+ 50),
                      (1100, 350), (0, 255, 0), cv2.FILLED)
        cv2.putText(stackedImage, "Scan Saved", (int(stackedImage.shape[1] / 2) - 200,
int(stackedImage.shape[0] / 2)),
                    cv2.FONT_HERSHEY_DUPLEX, 3, (0, 0, 255), 5, cv2.LINE_AA)
        cv2.imshow('Result', stackedImage)
        pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"

        img = cv2.imread("Scanned/myImage"+str(count)+".jpg")
        grey = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        adaptive_threshold = cv2.adaptiveThreshold(grey, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 111,
                                                   11)

        text = pytesseract.image_to_string(adaptive_threshold)
        print(text)
        # cv2.imshow("grey",grey)
        # cv2.imshow("adaptive_th",adaptive_threshold)
        cv2.waitKey(0)
        cv2.waitKey(300)
        count += 1
```

utilities(here all the extra functions are declared like stacking function , trackbars,etc)

```
import cv2
import numpy as np

## TO STACK ALL THE IMAGES IN ONE WINDOW
def stackImages(imgArray,scale,lables=[]):
    rows = len(imgArray)
    cols = len(imgArray[0])
    rowsAvailable = isinstance(imgArray[0], list)
    width = imgArray[0][0].shape[1]
    height = imgArray[0][0].shape[0]
    if rowsAvailable:
        for x in range ( 0, rows):
            for y in range(0, cols):
                imgArray[x][y] = cv2.resize(imgArray[x][y], (0, 0), None, scale, scale)
                if len(imgArray[x][y].shape) == 2: imgArray[x][y]= cv2.cvtColor( imgArray[x][y],
cv2.COLOR_GRAY2BGR)
        imageBlank = np.zeros((height, width, 3), np.uint8)
        hor = [imageBlank]*rows
        hor_con = [imageBlank]*rows
        for x in range(0, rows):
            hor[x] = np.hstack(imgArray[x])
            hor_con[x] = np.concatenate(imgArray[x])
        ver = np.vstack(hor)
        ver_con = np.concatenate(hor)
    else:
        for x in range(0, rows):
            imgArray[x] = cv2.resize(imgArray[x], (0, 0), None, scale, scale)
            if len(imgArray[x].shape) == 2: imgArray[x] = cv2.cvtColor(imgArray[x], cv2.COLOR_GRAY2BGR)
        hor= np.hstack(imgArray)
        hor_con= np.concatenate(imgArray)
        ver = hor
    if len(lables) != 0:
        eachImgWidth= int(ver.shape[1] / cols)
        eachImgHeight = int(ver.shape[0] / rows)
        print(eachImgHeight)
        for d in range(0, rows):
            for c in range (0,cols):

cv2.rectangle(ver,(c*eachImgWidth,eachImgHeight*d),(c*eachImgWidth+len(lables[d][c])*13+27,30+eachImgHei
ght*d),(255,255,255),cv2.FILLED)

cv2.putText(ver,lables[d][c],(eachImgWidth*c+10,eachImgHeight*d+20),cv2.FONT_HERSHEY_COMPLEX_SM
ALL,0.7,(0,0,0),2)
    return ver

def reorder(myPoints):

    myPoints = myPoints.reshape((4, 2))
    myPointsNew = np.zeros((4, 1, 2), dtype=np.int32)
    add = myPoints.sum(1)

    myPointsNew[0] = myPoints[np.argmin(add)]
    myPointsNew[3] =myPoints[np.argmax(add)]
```

```
    diff = np.diff(myPoints, axis=1)
    myPointsNew[1] =myPoints[np.argmin(diff)]
    myPointsNew[2] = myPoints[np.argmax(diff)]

    return myPointsNew


def biggestContour(contours):
    biggest = np.array([])
    max_area = 0
    for i in contours:
        area = cv2.contourArea(i)
        if area > 5000:
            peri = cv2.arcLength(i, True)
            approx = cv2.approxPolyDP(i, 0.02 * peri, True)
            if area > max_area and len(approx) == 4:
                biggest = approx
                max_area = area
    return biggest,max_area
def drawRectangle(img,biggest,thickness):
    cv2.line(img, (biggest[0][0][0], biggest[0][0][1]), (biggest[1][0][0], biggest[1][0][1]), (0, 255, 0), thickness)
    cv2.line(img, (biggest[0][0][0], biggest[0][0][1]), (biggest[2][0][0], biggest[2][0][1]), (0, 255, 0), thickness)
    cv2.line(img, (biggest[3][0][0], biggest[3][0][1]), (biggest[2][0][0], biggest[2][0][1]), (0, 255, 0), thickness)
    cv2.line(img, (biggest[3][0][0], biggest[3][0][1]), (biggest[1][0][0], biggest[1][0][1]), (0, 255, 0), thickness)

    return img

def nothing(x):
    pass

def initializeTrackbars(intialTracbarVals=0):
    cv2.namedWindow("Trackbars")
    cv2.resizeWindow("Trackbars", 360, 240)
    cv2.createTrackbar("Threshold1", "Trackbars", 200,255, nothing)
    cv2.createTrackbar("Threshold2", "Trackbars", 200, 255, nothing)


def valTrackbars():
    Threshold1 = cv2.getTrackbarPos("Threshold1", "Trackbars")
    Threshold2 = cv2.getTrackbarPos("Threshold2", "Trackbars")
    src = Threshold1,Threshold2
    return src
```

8. Overall Discussion

To actualize our code and make it work we have utilized tesseract. Tesseract is an open source content acknowledgment (OCR) Engine. It tends to be utilized legitimately, utilizing an API to extricate printed content from pictures. It bolsters a wide assortment of dialects. Tesseract doesn't have built in GUI, however there are a few accessible from the 3rdParty page. Tesseract is good with many programming languages and structures through coverings that can be found here. It tends to be utilized with the current format investigation to perceive message inside a huge record, or it very well may be utilized related to an outside book finder to perceive content from a picture of a solitary book line.

9. Conclusion

Despite the fact that an enormous number of algorithms have been proposed in the literature, no single method can give agreeable execution in all the applications because of the huge varieties in character text style, size, texture, colour, font and so on. Through this project we are in the stream of determining the palatable outcomes by improving the contribution by calibrating the picture and inferring the ideal degrees of precision from TESSERACT.

References

[1]https://repository.iiitd.edu.in/xmlui/bitstream/handle/123456789/556/Aditi%20Mithal_2013122.pdf?sequence=1&isAllowed=y

[2] https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/33418.pdf

[3] https://www.academia.edu/35938272/Text_Extraction_from_Image_using_Python

[4] https://www.analyticsvidhya.com/blog/2020/05/build-your-own-ocr-google-tesseract-opencv/

[5] https://ieeexplore.ieee.org/document/8721372

[6] https://www.ijtsrd.com/computer-science/simulation/2501/text-extraction-from-image-using-python/tgnana-prakash

[7] http://www.ijsrd.com/articles/IJSRDV5I80423.pdf

[8] https://arxiv.org/pdf/1907.04917.pdf

Appendix for Acronyms

Each Appendix appears on its own page.

OCR	Optical Character Recognition
DIP	Digital Image Processing