Computer Vision **(Due: 26/04/2020)**

Implementing my own Cam-Scanner (Quiz_C1) [2 marks]

Instructor: Tamal Maharaj *Name:* Jimut Bahan Pal

#TASK: Take one images containing a rectangular object (page, book, photo, whatever!). You MUST NOT use any image found online. Write code to only extract that object and save as an image. You can identify the corners manually. Also, you can provide the size of the resultant image as per the requirement. [Example: For a photo containing a A4 sheet, you know the dimension anyway]. Write a short report containing the original image and converted image, and a description of the OpenCV methods used and why. Submit the report, your code, and the image used. This following code should be useful for creating your own 'cam-scanner'.

#SOLN: There are several scanner documents that uses the tool of image transformations to separate [1] the actual content of the image from the original image. These kinds of tools are readily available on the market to perform quick 3rd Generation scanning tasks without any manual scanner. We have used the concept of just separating the document or region of interest from the actual image by using openCV functions. The process of extracting the document is simple. The following steps are required to perform the task:

1. **Input the image along with the 4 coordinates** - First we need to take the image and the 4 coordinates which will cut out a polygon from the input image. This polygon will be converted to a rectangle according to the image width and height that will be extracted.
2. **Extract the image and convert it to grayscale** -The extracted image needs to be converted to grayscale. This will help us in proper thresholding and proper removal of noise [2] to look as far as Cam Scanner scanned document.
3. **Apply proper threshold** - The final image will be thresholded properly. Since this is a very naive version of the application, we will only need to manually set threshold and remove the noise accordingly by trial and error. The commercial version of the Cam Scanner has many closed source algorithms for noise removal and providing clear binarization. The final image is ready for conversion to pdf and will resemble a fully scanned document.

Since this is a very naive version, will will work on 3 real life examples. We have used the functions of OpenCV to perform the $Perspective\ Transform$ and $Warp\ Perspective$ by the use of $cv2.getPerspectiveTransform()$ and $cv2.warpPerspective()$. These functions perform the exact alignement of the croped image to the final resultant image.

Our results are as follows:
Firstly, we took an image of the notebook containing a document text page as shown in Figure 1a (left). We then subject this to a transformation as shown in Figure 1a (right). The cropped image is converted to Grayscale for performing thresholding and looking it as close to the original scanned document as shown in Figure 1b (left). The final image after performing a Gaussian blur with a kernel size of 5 and applying Otsu's thresholding and Binary thresholding is shown in Figure 1b (right). The result obtained is pretty close to Cam Scanner, but the effort is worth one hour!

Secondly, we took an image of the text book containing a title text page as shown in Figure 2a (left). We then subject this to a transformation as shown in Figure 2a (right). The cropped image is converted to Grayscale for performing thresholding and looking it as close to the original scanned document as shown in Figure 2b (left). The final image after performing a Gaussian blur with a kernel size of 5 and applying Addaptive Gaussian thresholding and Binary thresholding is shown in Figure 2b (right).

Lastly, we took another image of a text book containing a title text page as shown in Figure 3a (left). We then subject this to a transformation as shown in Figure 3a (right). The cropped image is converted to Grayscale for performing thresholding and looking it as close to the original scanned document as shown in Figure 3b (left).

The final image after performing a Gaussian blur with a kernel size of 5 and applying Otsu's thresholding and Binary thresholding is shown in Figure 3b (right).

References

[1] Adrian Rosebrock. How to build a kick-ass mobile document scanner in just 5 minutes, 2011. https://www.pyimagesearch.com/2014/09/01/build-kick-ass-mobile-document-scanner-just-5-minutes/ available on web, last accessed on April 23, 2020.

[2] Jimut Bahan Pal. A deeper look into hybrid images, 2020. https://arxiv.org/abs/2001.11302 available on web, last accessed on April 23, 2020.

1 Appendix: Codes

transformation

April 23, 2020

0.1 Implementing own Cam Scanner

Author : Jimut Bahan Pal

Instructor: Tamal Maharaj

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import cv2
```

[ ]:
```
def order_points(pts):
    # initialzie a list of coordinates that will be ordered
    # such that the first entry in the list is the top-left,
    # the second entry is the top-right, the third is the
    # bottom-right, and the fourth is the bottom-left
    rect = np.zeros((4, 2), dtype = "float32")
    pts = np.array(pts)
    # the top-left point will have the smallest sum, whereas
    # the bottom-right point will have the largest sum
    #print("pts => ",pts)
    s = pts.sum(axis = 1)
    rect[0] = pts[np.argmin(s)]
    rect[2] = pts[np.argmax(s)]
    # now, compute the difference between the points, the
    # top-right point will have the smallest difference,
    # whereas the bottom-left will have the largest difference
    diff = np.diff(pts, axis = 1)
    rect[1] = pts[np.argmin(diff)]
    rect[3] = pts[np.argmax(diff)]
    # return the ordered coordinates
    return rect
```

[ ]:
```
def four_point_transform(image, pts):
    # obtain a consistent order of the points and unpack them
    # individually
    rect = order_points(pts)
    (tl, tr, br, bl) = rect
    # compute the width of the new image, which will be the
    # maximum distance between bottom-right and bottom-left
```

```
    # x-coordiates or the top-right and top-left x-coordinates
    widthA = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
    widthB = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
    maxWidth = max(int(widthA), int(widthB))
    # compute the height of the new image, which will be the
    # maximum distance between the top-right and bottom-right
    # y-coordinates or the top-left and bottom-left y-coordinates
    heightA = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
    heightB = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
    maxHeight = max(int(heightA), int(heightB))
    # now that we have the dimensions of the new image, construct
    # the set of destination points to obtain a "birds eye view",
    # (i.e. top-down view) of the image, again specifying points
    # in the top-left, top-right, bottom-right, and bottom-left
    # order
    dst = np.array([
        [0, 0],
        [maxWidth - 1, 0],
        [maxWidth - 1, maxHeight - 1],
        [0, maxHeight - 1]], dtype = "float32")
    # compute the perspective transform matrix and then apply it
    M = cv2.getPerspectiveTransform(rect, dst)
    warped = cv2.warpPerspective(image, M, (maxWidth, maxHeight))
    # return the warped image
    return warped
```

[ ]:
```
img = cv2.imread("driptamj.jpg",1)
driptamj = [[316,164],[940,212],[972,1272],[148,1202]]
img1 = four_point_transform(img,driptamj)
```

[ ]:
```
plt.figure(figsize=(15,15))
plt.subplot(1,2,1)
plt.title('Original Image')
plt.imshow(img[:,:,::-1])

plt.subplot(1,2,2)
plt.title('Transformed Image')
plt.imshow(img1[:,:,::-1])
plt.show()

plt.figure(figsize=(15,15))

gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
plt.subplot(1,2,1)
plt.title('GrayScale Image')
plt.imshow(gray,cmap='gray')
```

```
blur = cv2.GaussianBlur(gray,(5,5),0)
ret,thresh2 = cv2.threshold(gray,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
#thresh2 = cv2.adaptiveThreshold(gray,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
#            cv2.THRESH_BINARY,11,2)

plt.subplot(1,2,2)
plt.title('Final Image')
plt.imshow(thresh2,cmap='gray')
plt.show()
```

[ ]:
```
img_ab = cv2.imread("arora-barak.jpg",1)
ab_coord = [[206,210],[906,154],[962,1264],[138,1236]]
img_crop = four_point_transform(img_ab,ab_coord)
```

[ ]:
```
plt.figure(figsize=(15,15))
plt.subplot(1,2,1)
plt.title('Original Image')
plt.imshow(img_ab[:,:,::-1])

plt.subplot(1,2,2)
plt.title('Transformed Image')
plt.imshow(img_crop[:,:,::-1])
plt.show()

plt.figure(figsize=(15,15))

gray1 = cv2.cvtColor(img_crop, cv2.COLOR_BGR2GRAY)
plt.subplot(1,2,1)
plt.title('GrayScale Image')
plt.imshow(gray1,cmap='gray')



blur = cv2.GaussianBlur(gray1,(5,5),0)
#ret,thresh3 = cv2.threshold(gray1,134,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
thresh3 = cv2.adaptiveThreshold(gray1,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
            cv2.THRESH_BINARY,13,5)

plt.subplot(1,2,2)
plt.title('Final Image')
plt.imshow(thresh3,cmap='gray')
plt.show()
```

[ ]:
```
img_papa = cv2.imread("papa.jpg",1)
papa_coord = [[274,120],[848,152],[996,1004],[288,1032]]
```

```
img_crop_papa = four_point_transform(img_papa,papa_coord)
```

[ ]:
```
plt.figure(figsize=(15,15))
plt.subplot(1,2,1)
plt.title('Original Image')
plt.imshow(img_papa[:,:,::-1])

plt.subplot(1,2,2)
plt.title('Transformed Image')
plt.imshow(img_crop_papa[:,:,::-1])
plt.show()

plt.figure(figsize=(15,15))

gray3 = cv2.cvtColor(img_crop_papa, cv2.COLOR_BGR2GRAY)
plt.subplot(1,2,1)
plt.title('GrayScale Image')
plt.imshow(gray3,cmap='gray')



blur = cv2.GaussianBlur(gray1,(5,5),0)
ret,thresh3 = cv2.threshold(gray3,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
#thresh3 = cv2.adaptiveThreshold(gray1,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
#            cv2.THRESH_BINARY,13,5)

plt.subplot(1,2,2)
plt.title('Final Image')
plt.imshow(thresh3,cmap='gray')
plt.show()
```

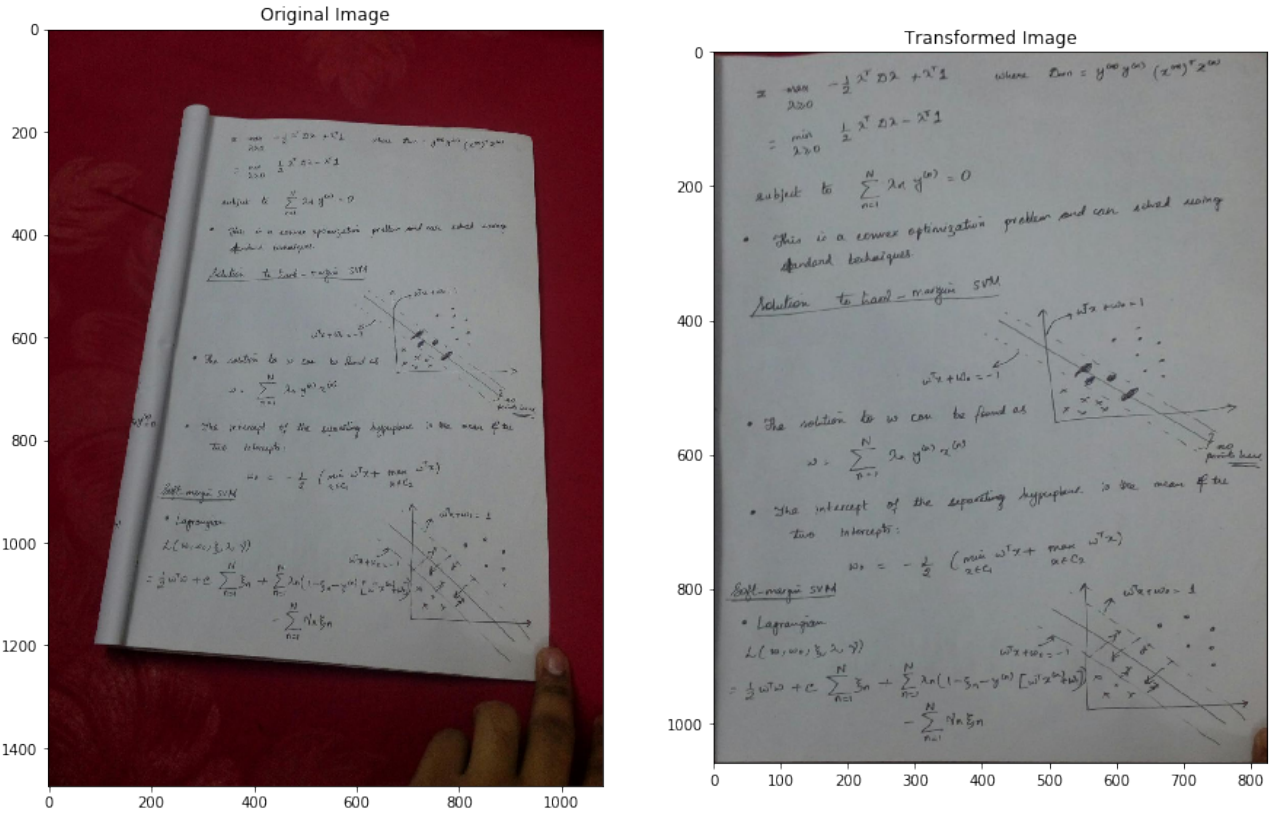
[ ]:
```
img_docwood = cv2.imread("doc_wood.jpg",1)
#doc coordinates in the following orders
# We have used GIMP to see the image coordinates, we could also have used␣
 ↪tkinter but that would be tedious at
# the moment
# upper left = 45, 25  upper right = 344, 26  lower left = 44, 265 lower right␣
 ↪= 343, 263 for the doc_wood.jpg
doc_wood = [(45,25),(344,26),(343,263),(44,265)]
img2 = four_point_transform(img_docwood,doc_wood)
```

[ ]:
```
plt.figure(figsize=(15,15))
plt.subplot(1,2,1)
plt.title('Original Image')
plt.imshow(img_docwood[:,:,::-1])

plt.subplot(1,2,2)
```

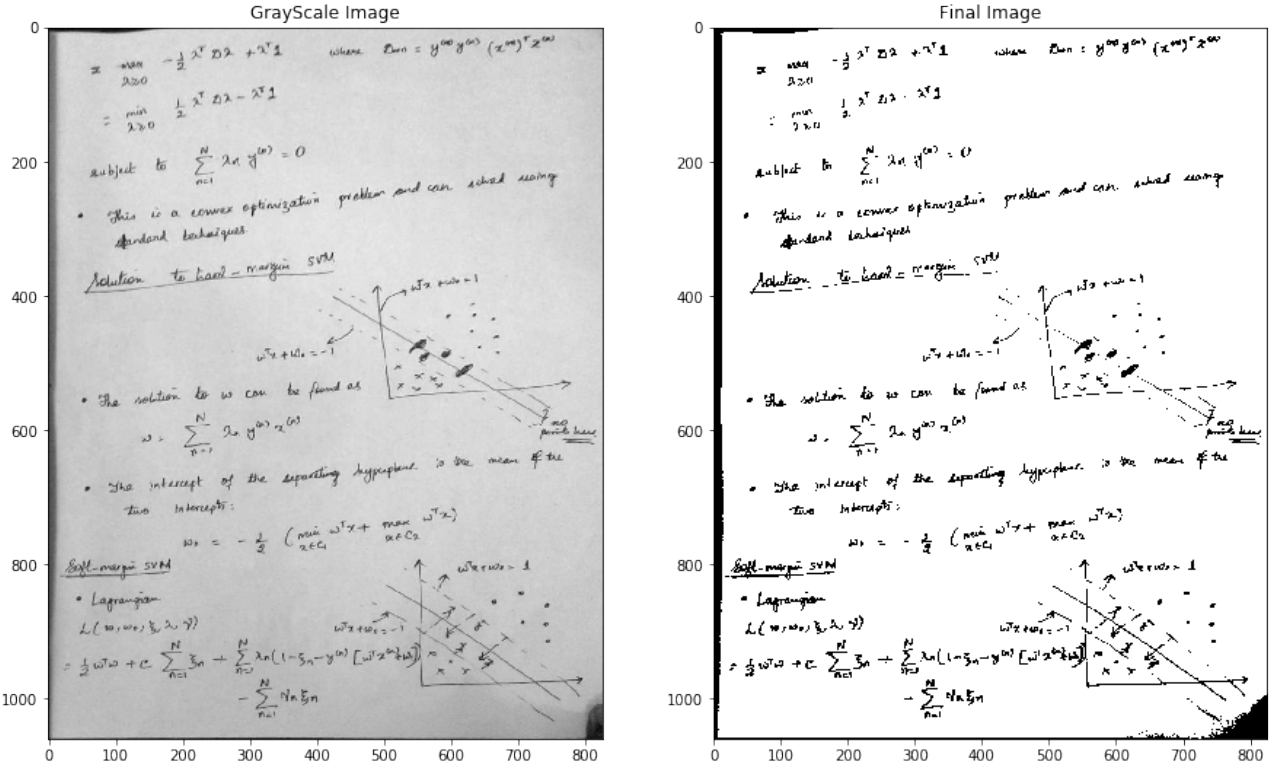
```
plt.title('Transformed Image')
plt.imshow(img2[:,:,::-1])
plt.show()

plt.figure(figsize=(15,15))

gray1 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
plt.subplot(1,2,1)
plt.title('GrayScale Image')
plt.imshow(gray1,cmap='gray')



blur = cv2.GaussianBlur(gray1,(5,5),0)
ret,thresh3 = cv2.threshold(gray1,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
#thresh3 = cv2.adaptiveThreshold(gray1,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
#            cv2.THRESH_BINARY,13,5)

plt.subplot(1,2,2)
plt.title('Final Image')
plt.imshow(thresh3,cmap='gray')
plt.show()
```

0.2 Reference Urls:

1. https://www.learnopencv.com/image-alignment-feature-based-using-opencv-c-python/

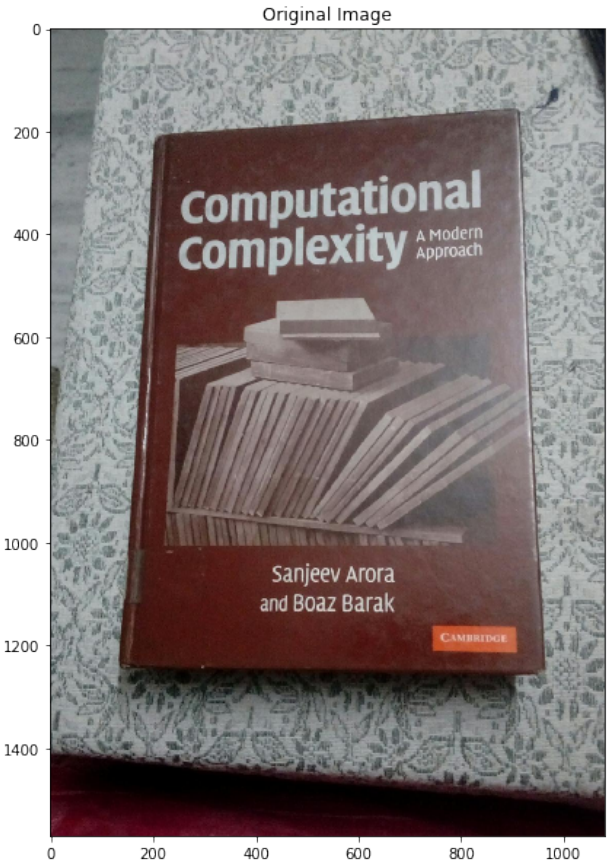


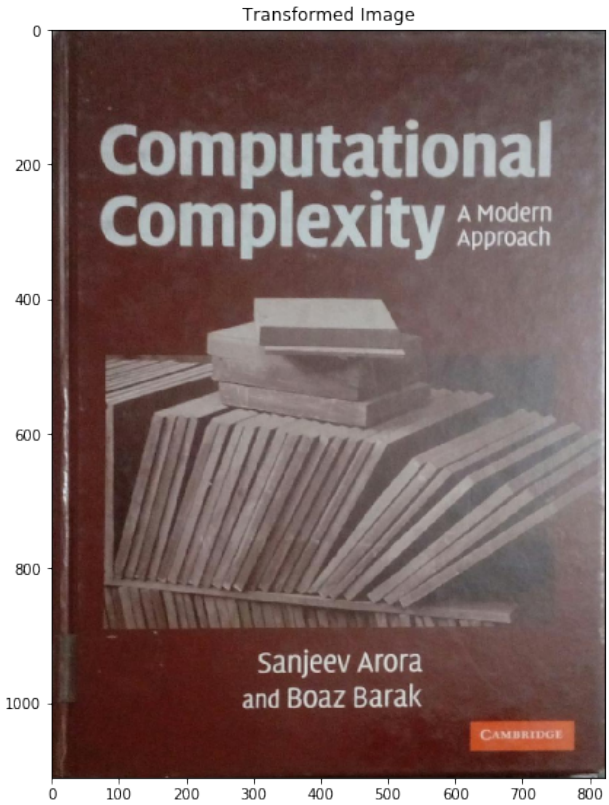
(a) The original image (left), the transformed cropped image (right).



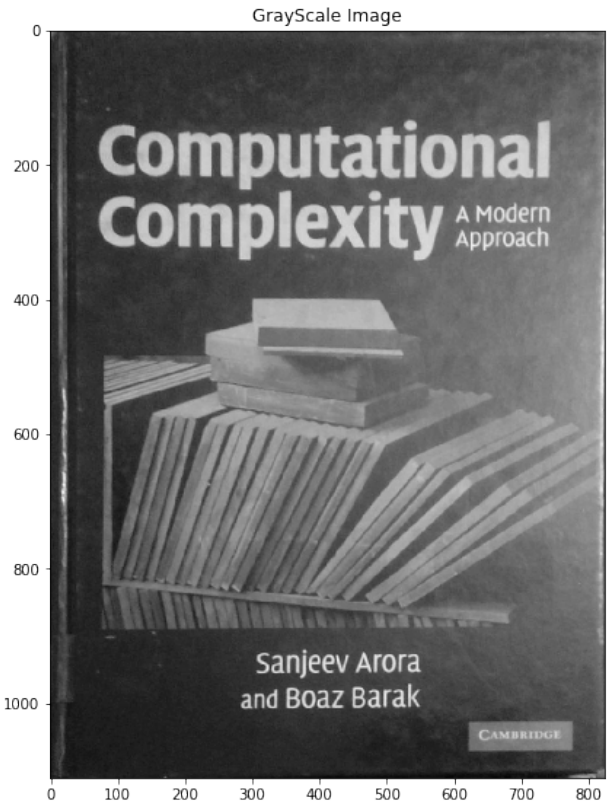
(b) The Grayscale version of the cropped image (left) and the thresholded version of the same image (right).

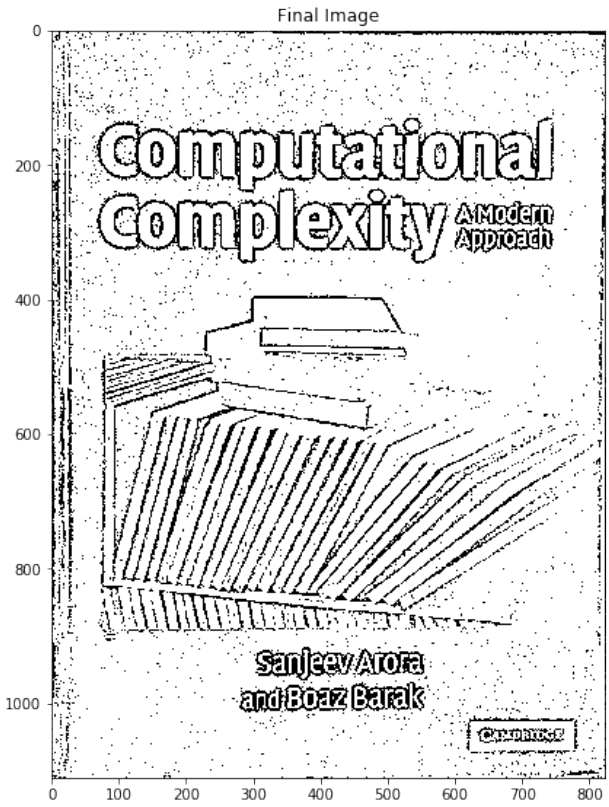
Figure 1: Image of a document when subjected to the algorithm.





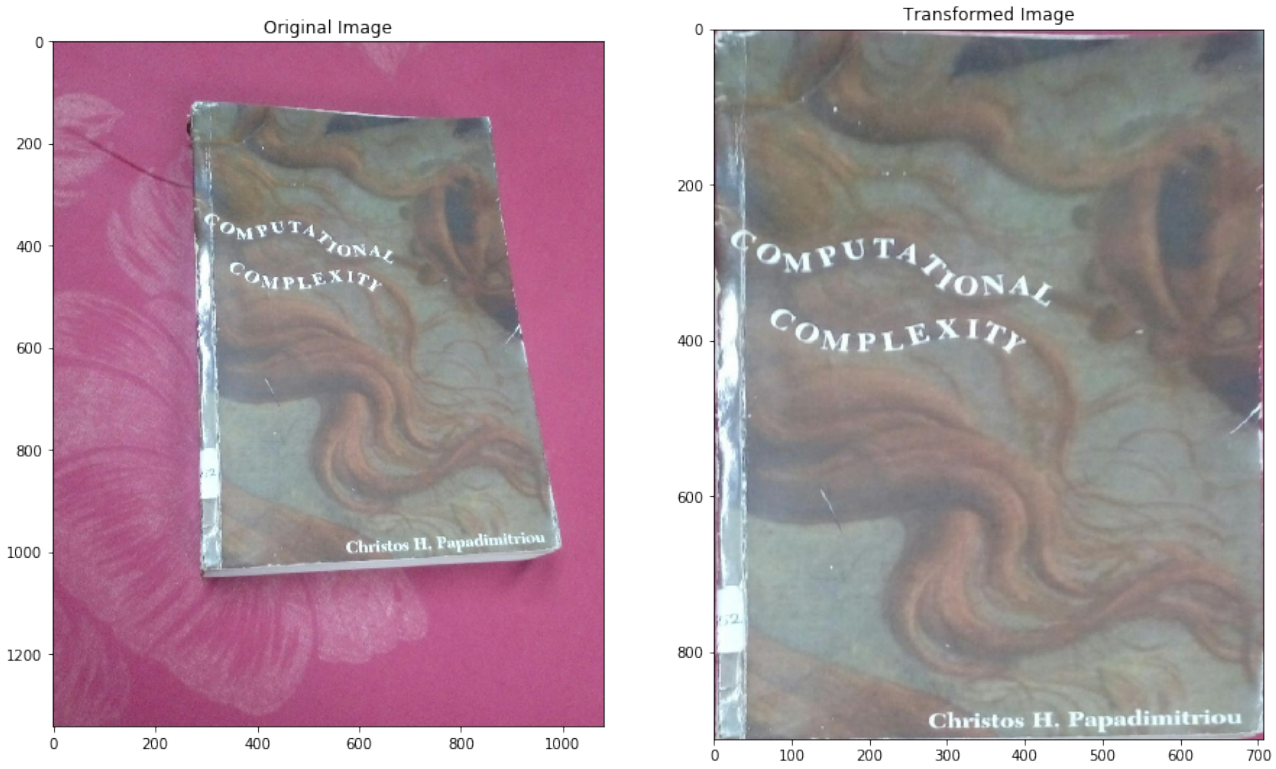
(a) The original image (left), the transformed cropped image (right).



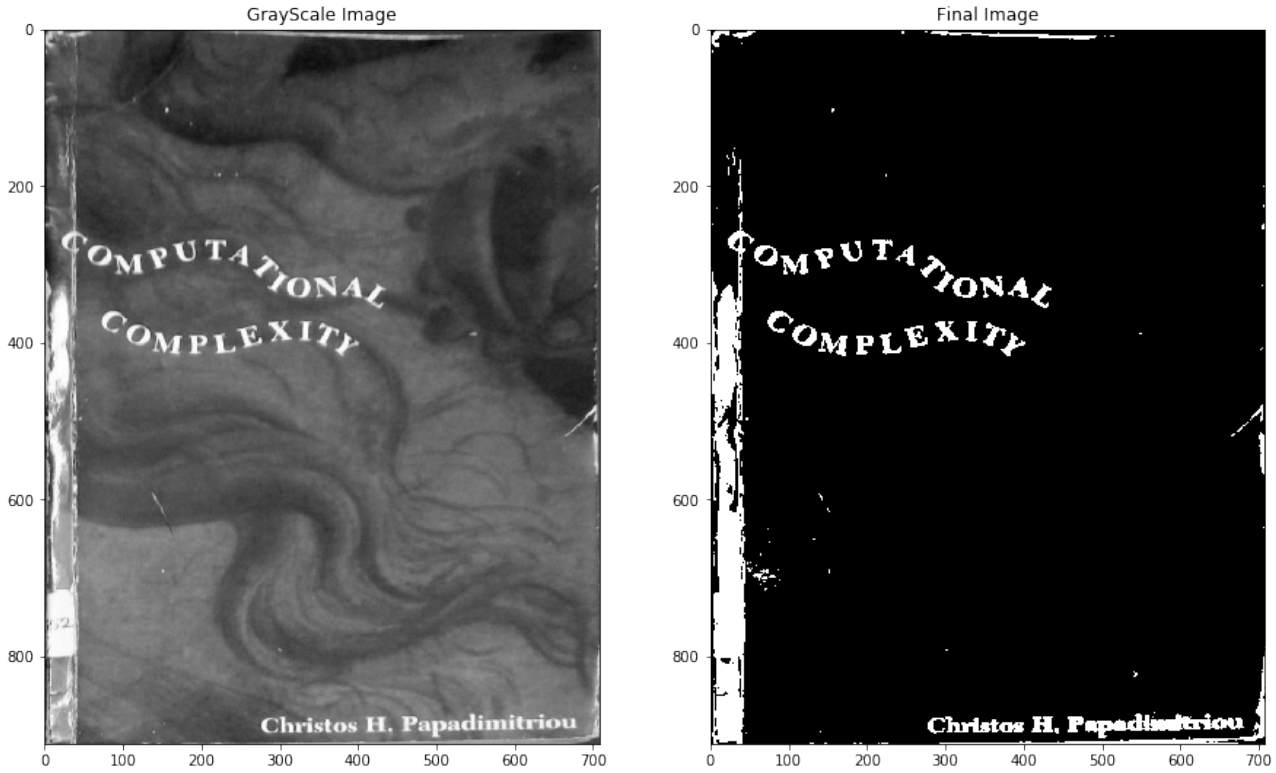


(b) The Grayscale version of the cropped image (left) and the thresholded version of the same image (right).

Figure 2: Image of a text book when subjected to the algorithm.



(a) The original image (left), the transformed cropped image (right).



(b) The Grayscale version of the cropped image (left) and the thresholded version of the same image (right).

Figure 3: Image of another text book when subjected to the algorithm.