Two input images:-

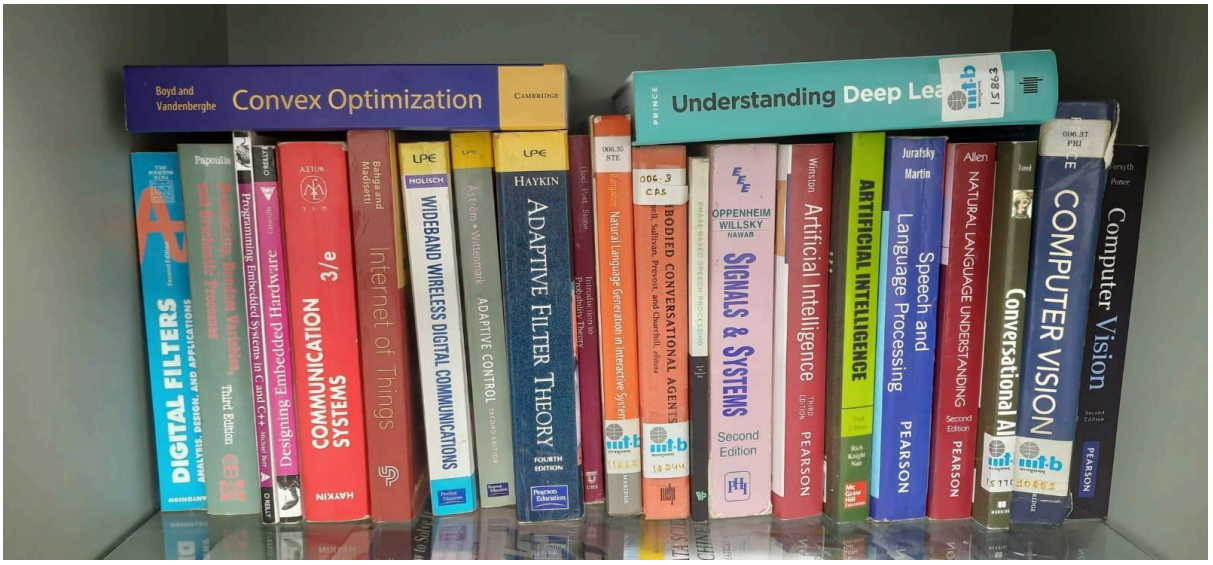


Fig 1

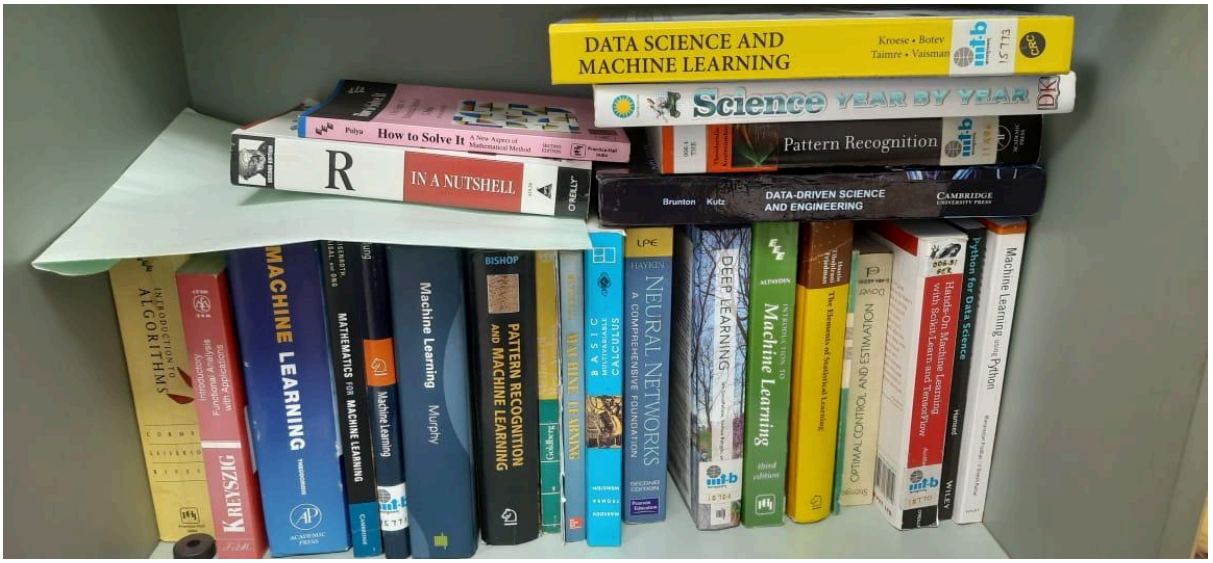


Fig 2

The problem statement was to find out number of books in the images. I have tried 2 different approaches to count the number of books in the image.

1 (i) Number of Points in Contour:

I started with importing libraries and took image as input.

```
import cv2
import numpy as np
image = cv2.imread('/content/BookCount_1.jpeg')
```

Then, I converted the input image into grayscale and performed canny edge detection on the grayscale image.

```
gray_scale_img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edge_img = cv2.Canny(gray_scale_img, 30, 200)
cv2.imwrite('/content/canny_edges.jpg', edge_img)
```

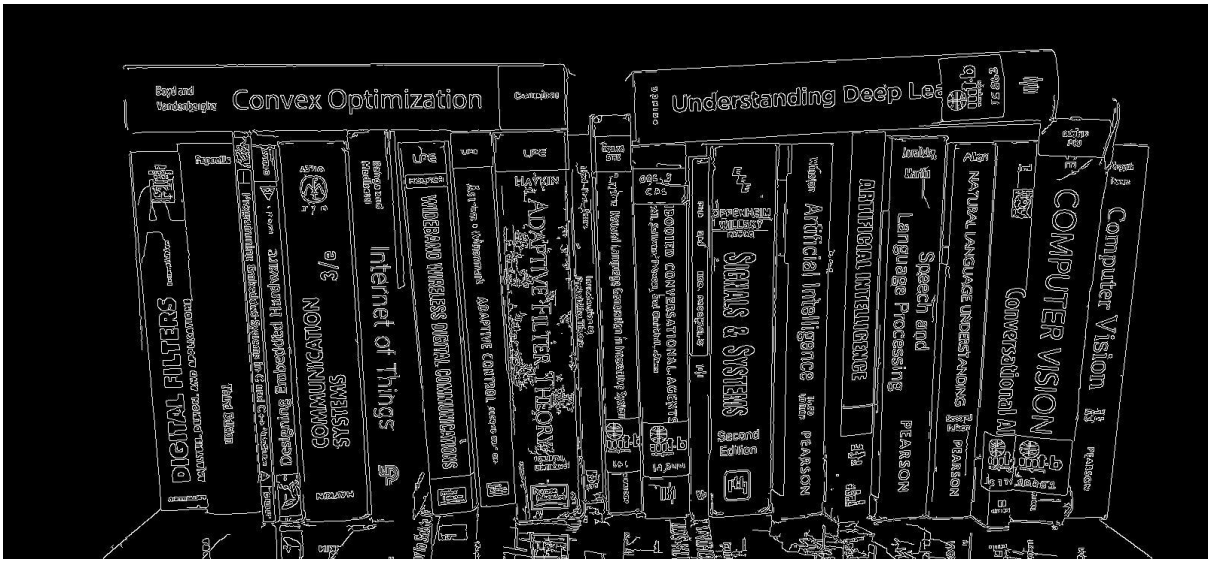


Fig 3 [Grayscale Image of Fig1]

Following this, I used .findContours() function and noticed that both edges of books and characters were figured out in the contour list.

```
contours, hierarchy = cv2.findContours(edge_img,cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
print("Number of Contours found = " + str(len(contours)))
```

```
Number of Contours found = 1499
```

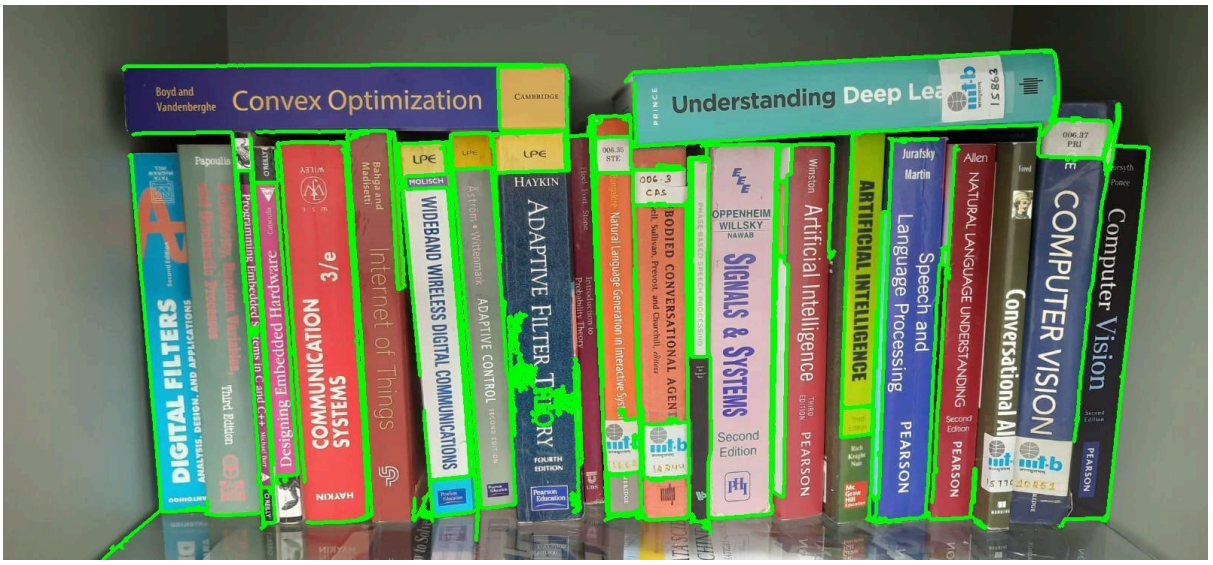
This showed that number of contours was 1499. The contour list showed that elements of it were of not same length, which can also be understood in a way that bigger the contour larger will be the element length as more points will be inside it.

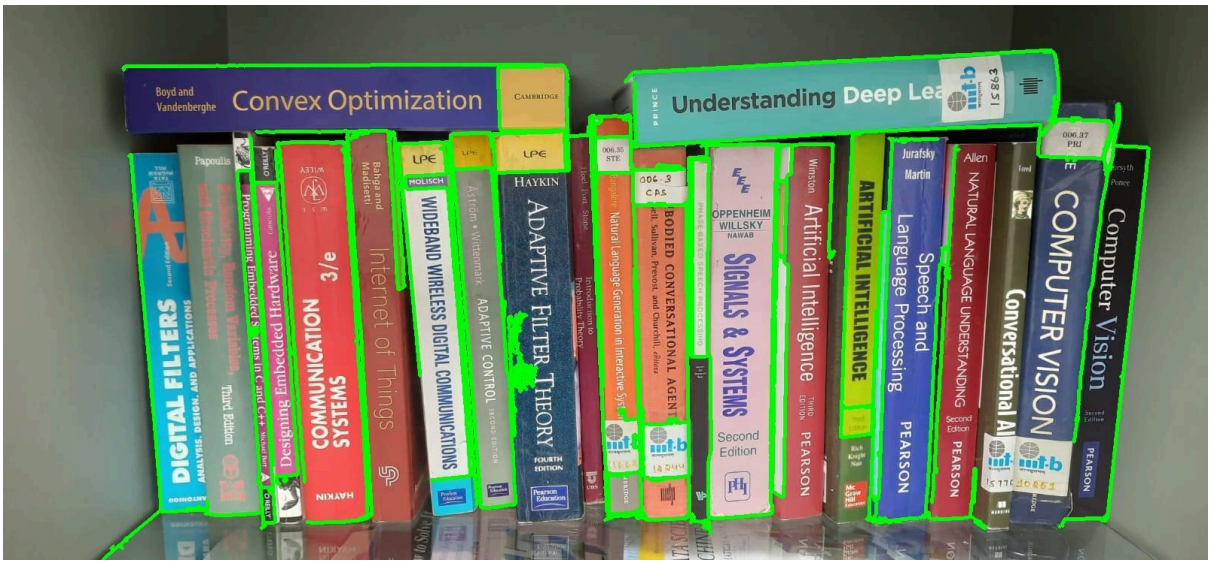
So, as character like ‘C’,’o’,’n’,’v’ etc are also counted but they have less number of points in it compared to whole outline of the book.
Finding this as distinguishing factor, I applied a threshold on this(length of elements in contour list) to remove characters and just mark the book edges.

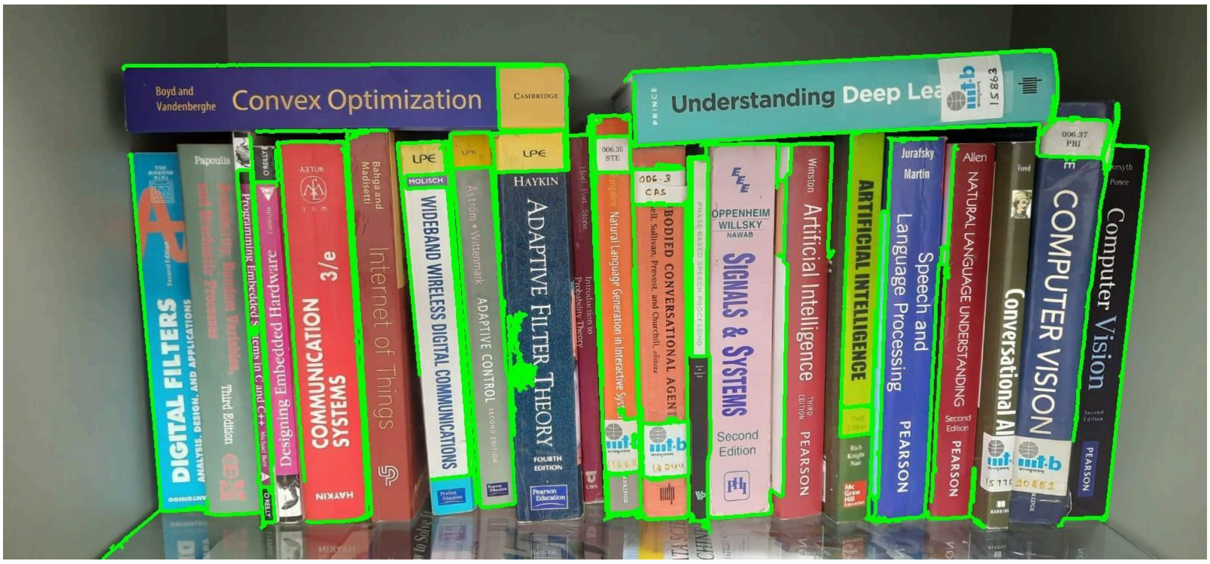
```
values=750
```

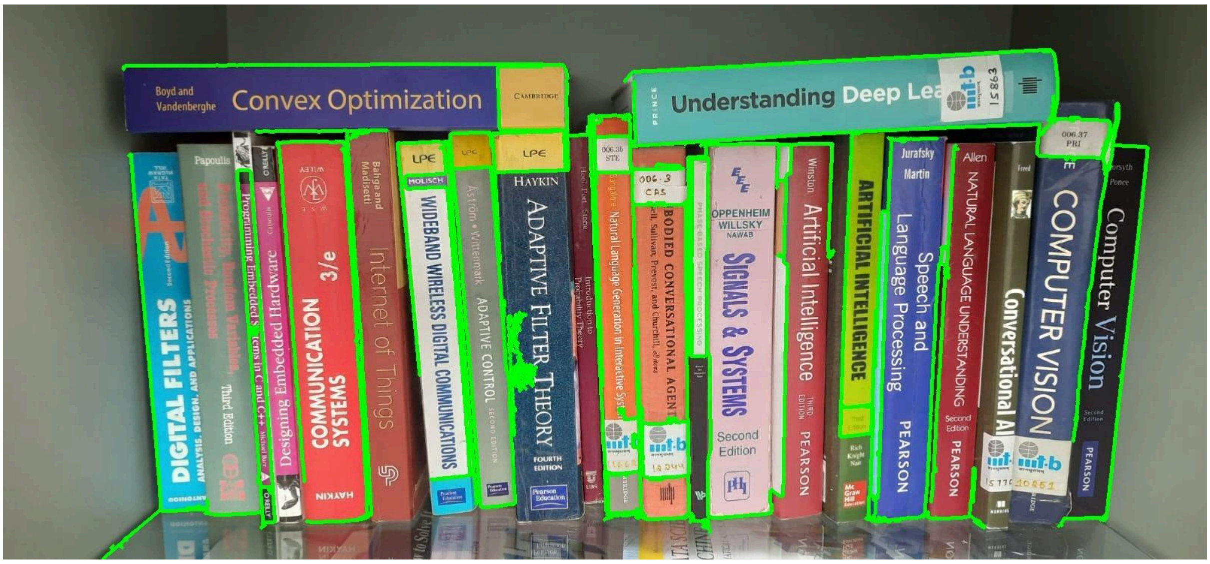
```
new_contours = []
for contour in contours:
   if len(contour) > values:
       new_contours.append(contour)

cv2.drawContours(image, filtered_contours, -1, (0, 255, 0), 3)
file_str='/content/contours_above_test_'+str(values)+'.jpg'
cv2.imwrite(file_str, image)
```

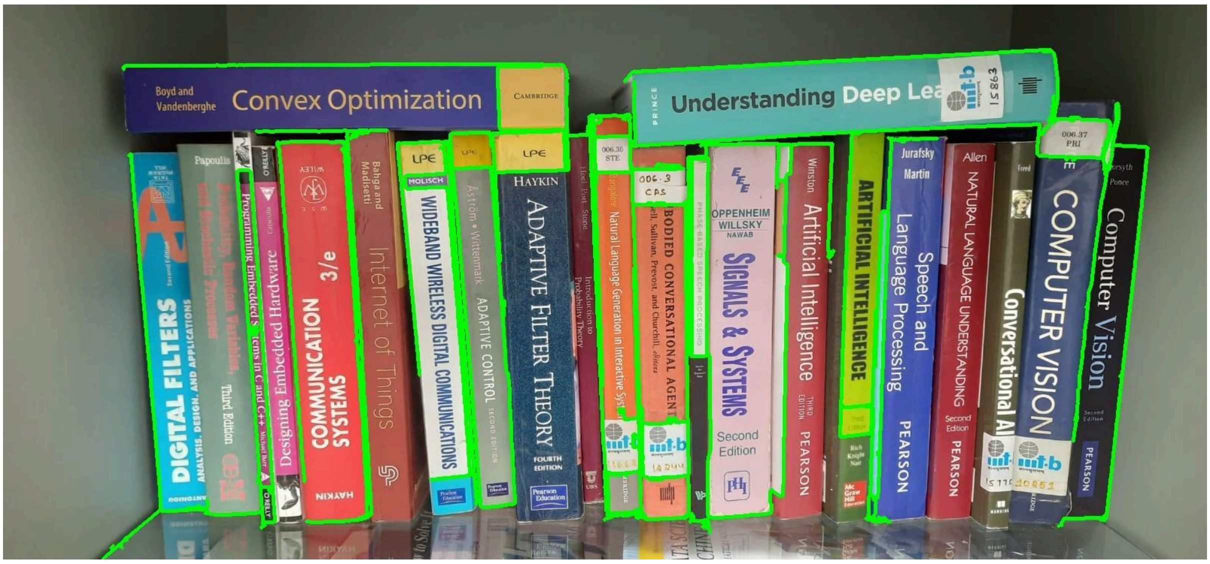
I changed the parameter *values* and tried noticing the difference in image and also the number of books.









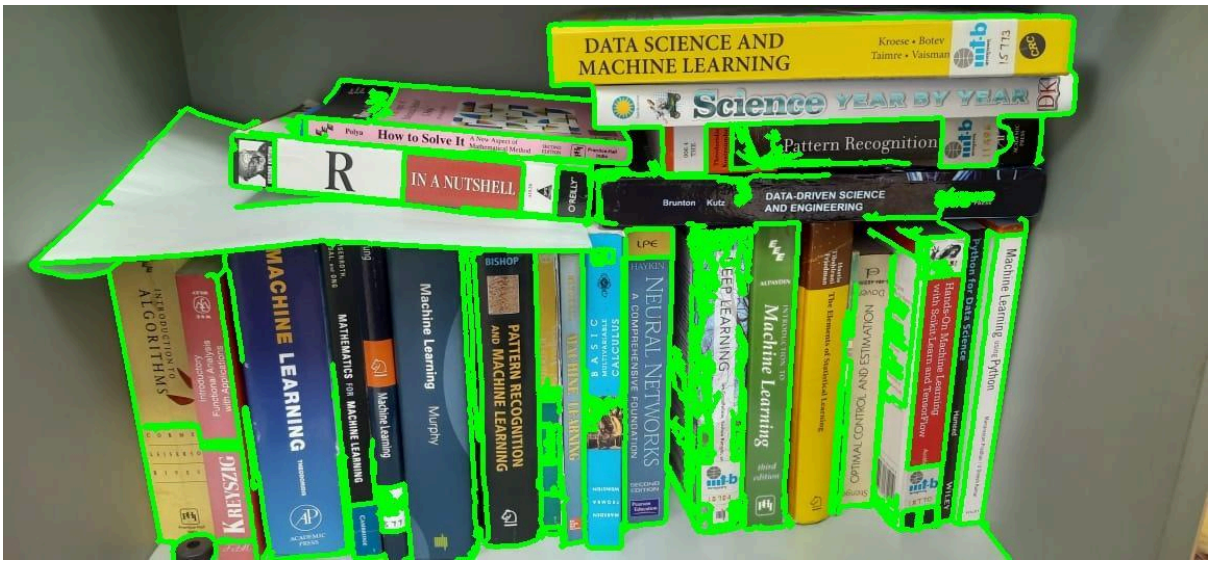


Observing this, we can say that as we increase the value of *values* the number of contours decreases and also the greenish patch which is there(

because it contains a lot of points in it) gets removed. But, we can't keep on increasing the value of *values* as book's edges are also getting removed.

I found optimal value of *values* to be **750.** There were 23 books in the first image(Fig1) and number of books detected by this method was 18.

In Fig2., number of books were 26 and the method detected 23 books.



ii) Using Hough lines:

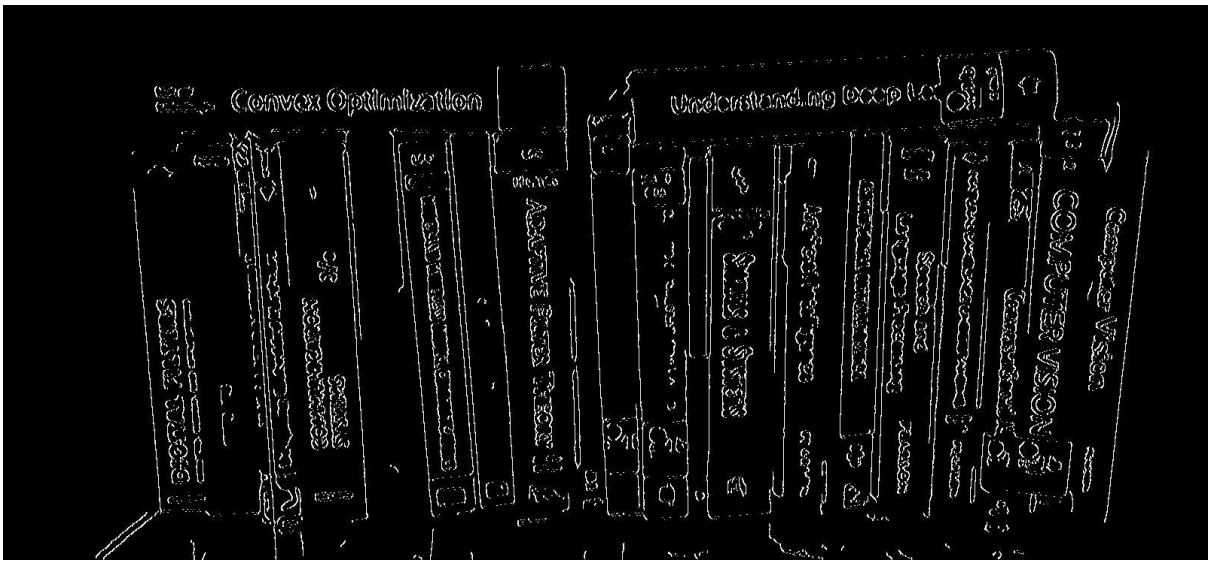
I have used Houghlines function as it helps in finding out different shapes but in this case it helps in finding lines.

Before applying houghlines function, I applied gaussian blur to reduce noise, converted this image to gray scale and finally applied canny edge detection

As there are characters also in the images so to remove them, I used erosion function. The purpose of erosion in this case is likely to thin the edges and emphasize the vertical components of the detected edges.

```
height, width, _ = img.shape
blur = cv2.GaussianBlur(img, (15, 15), 0)
gray = cv2.cvtColor(blur, cv2.COLOR_BGR2GRAY)
edge = cv2.Canny(gray, 50, 70)

vertical_kernel = np.array([
   [-1, 0, 1],
   [-1, 0, 1],
```

```
   [-1, 0, 1],
], dtype=np.int32)
img_erosion = cv2.filter2D(edge, -1, vertical_kernel)
```

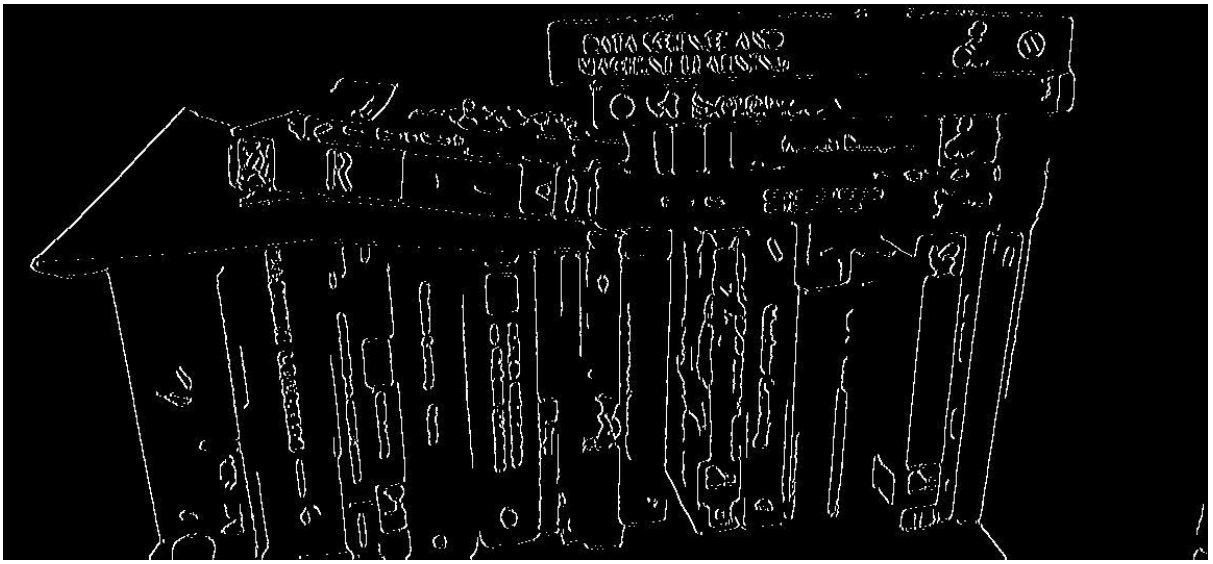


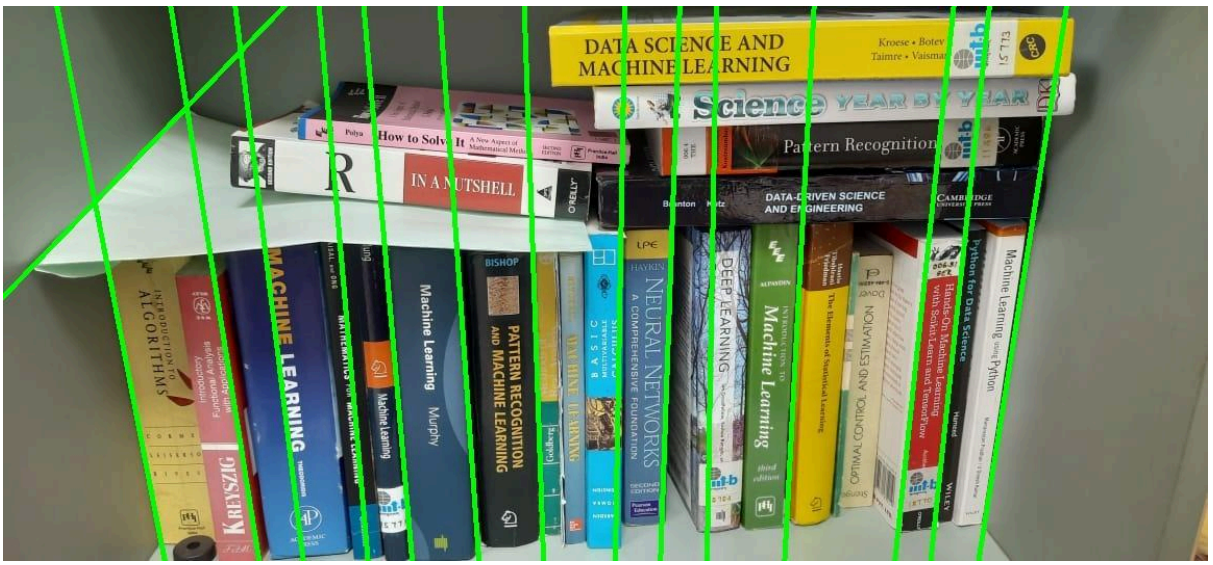

```
lines = cv2.HoughLines(img_erosion, 1, np.pi / 180, 125)

points = get_points_in_x_and_y(lines, height)
points.sort(key=lambda val: val[0][0])
non_duplicate_points = remove_duplicate_lines(points)
final_points = non_duplicate_points

for point in final_points:
   ((x1, y1), (x2, y2)) = point
   img = cv2.line(img, (x1, y1), (x2, y2), (0, 255, 0), 3)
```

This part detects lines in the edge-detected image, processes and filters these lines, and then visualizes the final set of non-duplicate lines on the original image
The cv2.HoughLines function provides lines in the parameterized form ρ,θ, where ρ denotes the distance of the closest point from the origin, and θ represents the angle.

I created a function named get_points_in_x_and_y to convert polar coordinates to x, y coordinates. Exploring an alternative approach, I generated the eroded image of the initial edge map by employing the cv2.erode function from OpenCV with a specified kernel and iterations.

This method proved effective in eliminating slender lines in the image that did not correspond to the edges of the book. The erode function operates by utilizing a kernel and determining the minimum pixel value within the kernel, contributing to the removal of unwanted features in the image.





These are the result on test image.