BOOK BUG

VINAY DATTA PINNAKA

COSC 5590 COMPUTER VISION AND IMAGE PROCESSING FALL 2015

Final Report

Instructor: Dr. Maryam Rahnemoonfar

ABSTRACT

Recognition of text from digital images is one of the primitive way to interact with real world objects. In this project text detection from a digital image is done with an efficient algorithm which can find the maximum contour and also determine other attributes of text in the image. This application project performs text recognition and retrieval, the main intention of this application is to bookmark a book digitally. A windows based console application is developed to demonstrate the algorithm that is proposed in this project. The output obtained after the text detection is passed to Tesseract OCR engine, an open source text detection software supported by google. Final result i.e., the text obtained is stored to ensure the book is bookmarked digitally. The performance of the approach is summarized based on the experimental results obtained.

1. INTRODUCTION

1.1 PROBLEM STATEMENT

Reading is not only a habit but it is an addiction, Reading addicts read number of books every day, but remembering the page you were left reading last time is not an easy task. We generally use bookmarks for hard copies of the book, In case of e-books we have option to make a bookmark which is provided by the e-book reader application. But in case, the reader may be having both hard copy and soft copy of the book and he may be switching between both now the problem arises how to coordinate between hard copy and soft copy. Book Bug is simple solution which bookmarks hard copy of the book digitally. The concept behind this application is interacting with real world objects from a digital object [like mobile device, Tablet, PC].

1.2 MOTIVATION

The basic implementation of Book Bug is optical character recognition (OCR). User takes the image of the page number and the book front cover. Details like book title, edition number, and authors name is extracted from the front cover as well as the page number is also extracted. The data is stored and when user wants to read that book again he again show up front cover of the book with his camera on the digital device. Application will retrieve the page number from the data stored previously and show up on screen. In addition to this if the user decided to copy text from the hard copy of book he can simply take the image of text the application will extract the digital copy of the text.

1.3 SUMMARY

This report is organized as follows, Section 2 presents the functionality of the entire application Section 3 presents the individual steps that are used in the algorithm to implement this project. Section 4 describes my contribution towards the project. Section 5 and 6 show and discusses results obtained. Finally the future work and conclusion

2. FUNCTIONALITY

In this section, the basic functionality of the application is discussed. The user has to select one option from the menu, either to 'bookmark the book' or to 'find the book'. If the user selects to bookmark a book then application asks for image of the book cover. Image will be taken from the web camera in the present application. In complete application the image will be obtained from the client app which runs on a smart phone. Process that is implemented to take an image is discussed in later sections. Next application asks for the image of the page number to bookmark. Both the images are processed and the data obtained is stored. If the user selects the option 'find the book'. Application will take only the image of the book cover and it will search in the data file for the page number. Block representation of the functionality is shown in Fig 1.

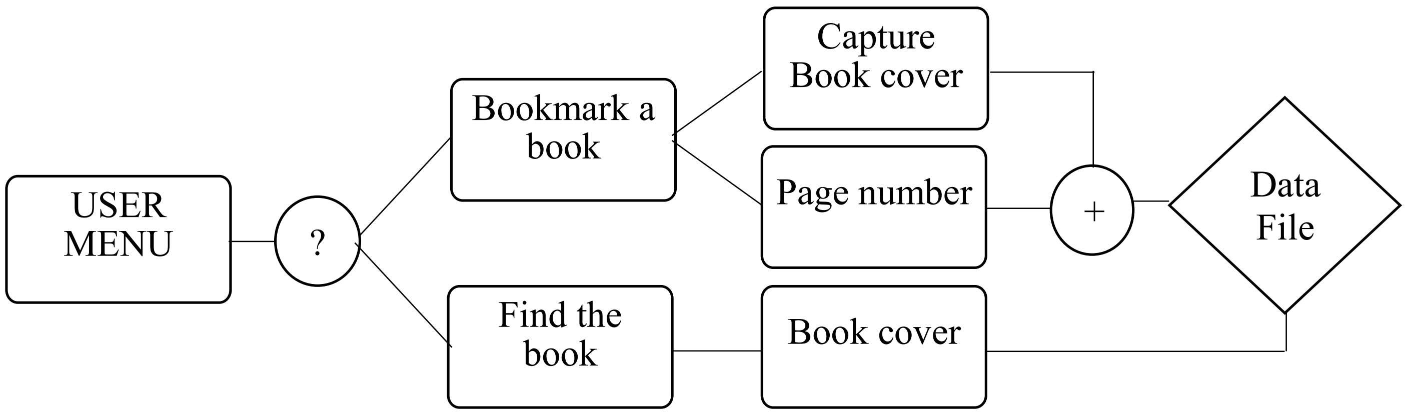


Fig 1: Functional block of Book Bug application.

3. THE PROPOSED APPROCH

This section describes all the processing steps involved in text detection and storage. The application is written in C++ using openCV libraries and the entire application is built in visual studio 2015. The following requirements are assumed to be fulfilled before running this application.

(a) A web cam is connected to the system.
(b) User is not using hand written text.
(c) Text books used to bookmark are with normal font (no graphic logos).

The steps involved in this application are:

Step 1: Input Image

User has to take the image of the book cover in order to store the title, edition and other details. So, a frame from the video captured by the camera is taken when user requests it.

OpenCV provides the function a class VideoCapture, This class captures video from video files cameras , image sequences. Here is how the class is used.

VideoCapture cap(0); *// open the default camera*

When mouse click event is occurred next frame is write to a Mat object using following code

VideoWriter::write(const Mat& image)

Step 2: Grayscale Image

The image read from the web camera is in RGB format so, before using it for processing convert it to gray scale.

Use opencv function cvtColor to convert RGB image to gray scale.

cvtColor(originalImg, grayImg, CV_RGB2GRAY);

Step 3: Gradient Image

To find the gradient of the image first design a structuring element of ellipse type, specifically ellipse because the text obtained is of type rounded rectangle. The opencv function used for generating a structuring element is

getStructuringElement(MORPH_ELLIPSE, Size(3, 3));

Once the structuring element is obtained then the morphological gradient operation is applied using the ellipse kernel. Opencv provides a morphological function

```
morphologyEx(grayImg, gradientImg, MORPH_GRADIENT, ellipseSE);
```

Step 4: Threshold

The gradient image obtained after morphological operation is to be converted into binary using a Threshold function. Threshold value is not set manually but OTSU algorithm determines the threshold value based on the histogram of the image. General implementation of the threshold function is

```
threshold(gradientImg, binaryImg, 0.0, 255.0, THRESH_BINARY | THRESH_OTSU);
```

Step 5: Closing

To connect all the letters of the word and make a large contour we need to apply closing operation on the binary image. Closing will connect all the letters, and for this functionality morphological close operation is performed using the line structuring element.

```
Mat closingK = getStructuringElement(MORPH_RECT, Size(9, 1));
morphologyEx(binaryImg, connectedImg, MORPH_CLOSE, closingK);
```

Step 6: Find contours

In the connected image all the contours has to be identified and among all the contours largest contours has to be separated which are basically text that are to be extracted. *FindContours* is a function in opencv which does this for you.

```
findContours(connectedImg, contours, hierarchy, CV_RETR_CCOMP,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
```

Step 7: Large contour

Once all the contours are found in the connected image then the next step is sort all the contours based on the area, the algorithm is designed such that indices of all the contours are sorted in an array based on their size. The area of the contour is obtained using the opencv function.

```
area = cv::contourArea(contours[i]);
```

The algorithm used to sort the contour index based on the area is as follows

```
If area greater than max_area then
    max_area = area
    index[max] = present contour index
```

Step 8: Draw contours

Using the array that is sorted based on the contour size algorithm draw contours on mask image. Each contour is drawn on separate mask such that while extracting the text it is obtained in order.

```
drawContours(mask, contours, max_idx, cv::Scalar(255), -1);
```

Step 9 : Mask

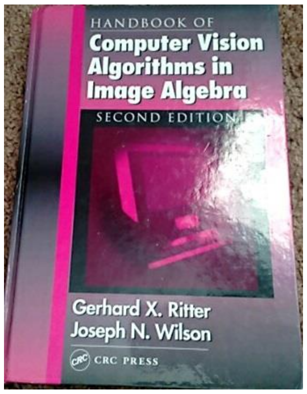
An output image is created using gray image and the mask that is generated in the previous step. This mask contains only the images that are identified as text. The condition used to differentiate the text from other contours is bright pixel ratio should be greater than 50 percent, width and height of the bounding rectangle should be greater than 10 and the width should be two times the height.

```
if (r > .50 && (rect.height > 8  && rect.width > 10) && (rect.width > 2* rect.height) )
```

Step 10: Tesseract

The final output obtained is ready to pass to the OCR engine, here in this case Tesseract an open source OCR engine is used. First initialize the tesseract engine and then set variables, data obtained from each final image is stored in data file.

The algorithm above discussed applies both for extracting text from book cover as well as the page number, but the difference is while extracting the page number some of the steps are skipped. For simple understanding the steps skipped in extracting page numbers are closing of the image, since we have very less contours in page number image closing is not required, apart from this finding large contours is unnecceary as we require all the contours in the page number image. Sample set of images used are shown below.



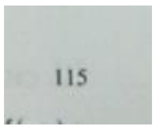


Fig 2: Book cover and Page number image

Block diagram representation of all the steps involved in the algorithm for extracting text from the book cover is shown below.

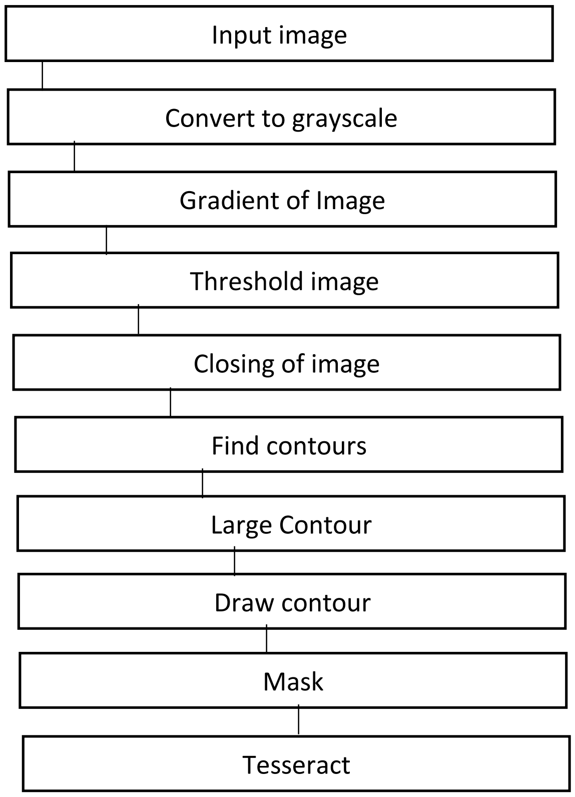


Fig 3: Algorithm used for book cover extraction

Block diagram representation of all the steps involved in the algorithm for extracting text from the Page number is shown below.

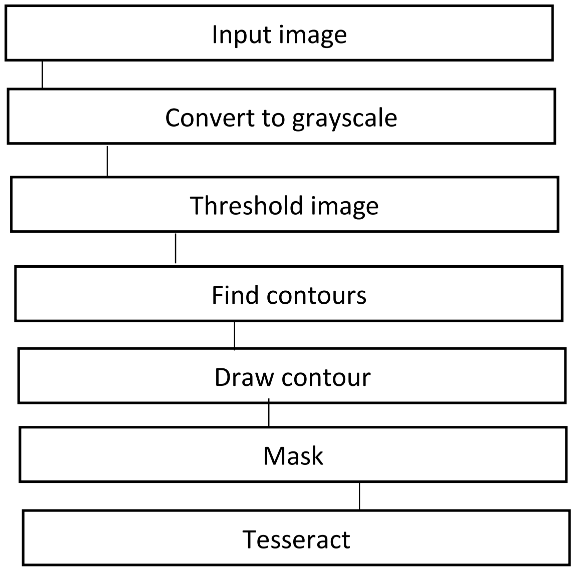


Fig 3: Algorithm used for Page number extraction

4. CONTRIBUTION

Many algorithms for extracting text from an image are in use but the approach used for book bug is different because it is basically designed to extract the features of the book like title, author, edition number and page number. The following are the contributions that are done to develop the specific algorithm for book bug.

(a) Works for both bright color background images as well as for dark color background images. This is implemented in the algorithm by finding the gradient first and processing is done on this gradient, therefore background and foreground color does not have much significance while extracting data.

(b) Concept of finding large contour is implemented in the algorithm which is able to sort the contours based on their area, sorting always good to obtain only the specific information from the image. Since in book cover the large contour represents the title of the book, this approach is apt for extracting important information from the image.

5. RESULTS

Results obtained intermediately while performing step by step algorithm on book cover are presented here.

Step 1:

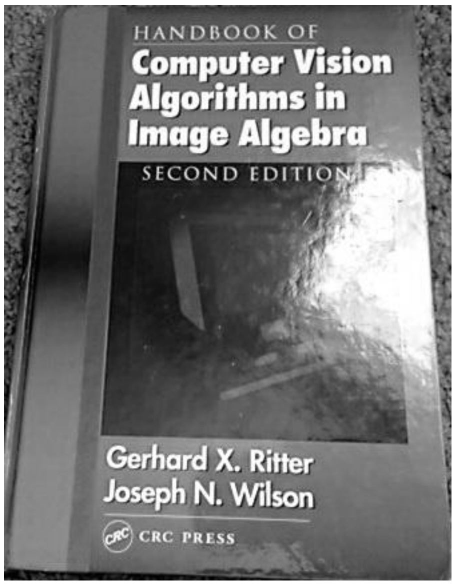


Fig 5a: Gray scale image of original book cover

Step 2:

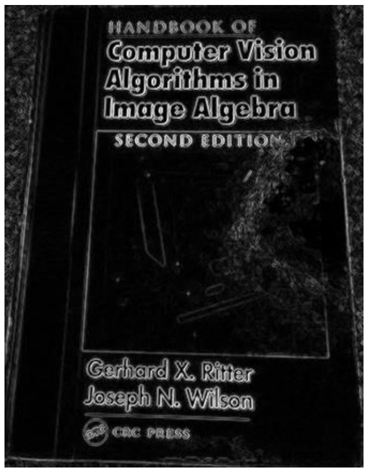


Fig 5b: Gradient image using ellipse as structuring element

Step 3:

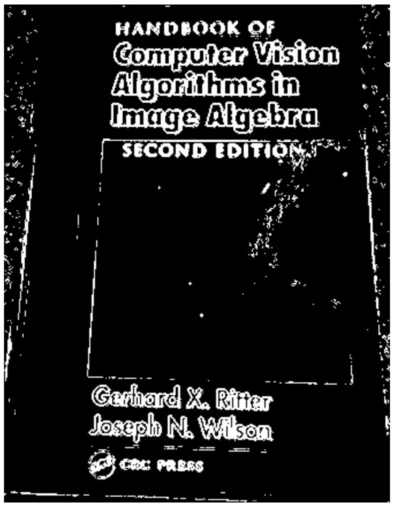


Fig 5c: Binary image obtained after thresholding using OTSU method

Step 4:

Fig 5d: Connected image obtained after closing operation

Step 5:

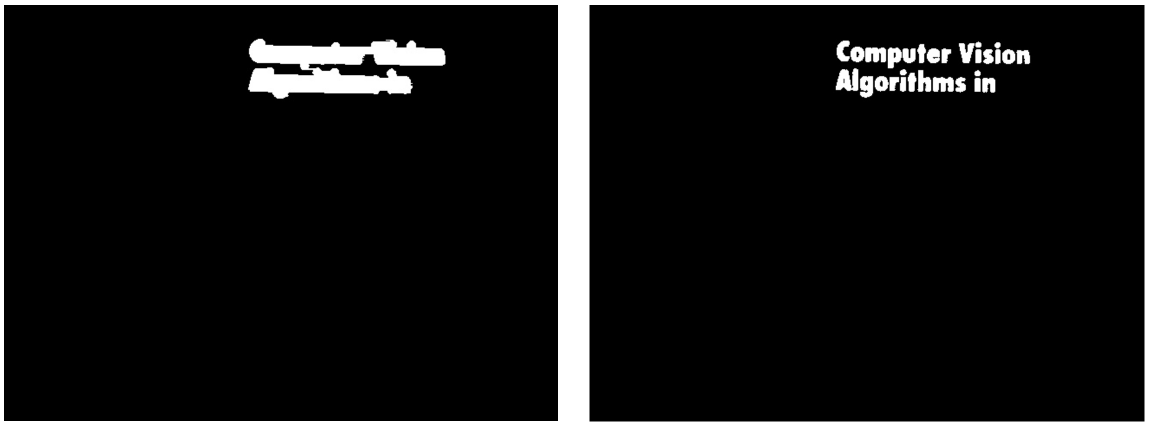


Fig 5e: Image with large contour and the final image that to be passed to tesseract.

Results obtained intermediately while performing step by step algorithm on Page number image are presented here.

Step 1:

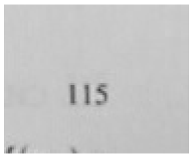


Fig 6a: Gray scale image of page number

Step 2:

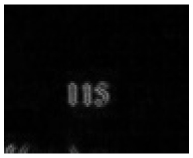


Fig 6b: Gradient of gray scale image

Step 3:

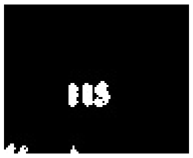


Fig 6c: Binary image of original gray scale

Step 4:

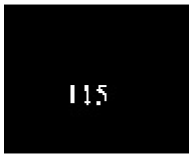


Fig 6d: final mask obtained that to be passed to tesseract.

6. RESULTS DISCUSSION

6.1 Book Cover

The gray scale image obtained in step 2 is used as a input for morphological gradient using 3x3 ellipse structuring element. The gradient of the image can be obtained either by sobel or laplacian filter but both of them have their own implications, so a morphological gradient which gives more accurate result is used.

After applying threshold the resultant image is used to perform morphological close operation with 9x1 line structuring element. Closing makes it easy to find contours and sort them based on their size.

In step 5 large contour is extracted and corresponding mask obtained is shown.

6.2 Page Number

The page number image is converted to gray scale and the corresponding gradient image is generated similar to the step 3 in book cover extraction.

The gradient image is thresholded but no closing operation is applied because every contour in page number is important so directly find the contours and draw the contours as in Fig 6c.

Now generate final image from the mask and the grayscale image as in Fig 6d. and pass it to tesseract to obtain the page number.

7. FUTURE WORK

The algorithm implemented is not 100% accurate, but it designed to not only to extract text but also to differentiate the attributes of the image. Future work involves increase in efficiency of the algorithm by considering all the patters of the image, calculating the font size and predicting text alignment.

An android application would be a better option to run this kind of applications, so the processing speed should be increased by multithreading concepts. Paragraph text extraction is next challenge that to be solved for complete fulfilment of Book bug.

8. CONCLUSION

Book bug is one of the simplest way to interact a hard copy of the book digitally. All the steps involved in implementing this application are image processing techniques. The results obtained are almost accurate except some few exceptions. The windows console based application written in c++ in visual studio IDE demonstrates the prototype for the actual application. Finally this application has to be ported to android platform which will be the future work of this application.

9. REFERENCES

[1] Derek Ma, Qiuhau Lin and Tong Zhang. Mobile Camera Based Text Detection and Translation.

[2] Julinda Gllavata1, Ralph Ewerth1 and Bernd Freisleben. A Robust Algorithm for Text Detection in Images

[3] Ayatullah Faruk Mollah, Nabamita Majumder, Subhadip Basu and Mita Nasipuri. Design of an Optical Character Recognition System for Camera-based Handheld Devices

[4] http://opencv-code.com/tutorials/how-to-read-the-digits-from-a-scratchcard/#more-807

[5] http://www.samhaskell.co.uk/blog/reading-real-books-aloud-using-free-software

[6] http://blog.ayoungprogrammer.com/2013/01/equation-ocr-part-1-using-contours-to.html