SIT789 – Robotics, Computer Vision and Speech Processing

Distinction Task 2.3: Document analysis and recognition

Objectives

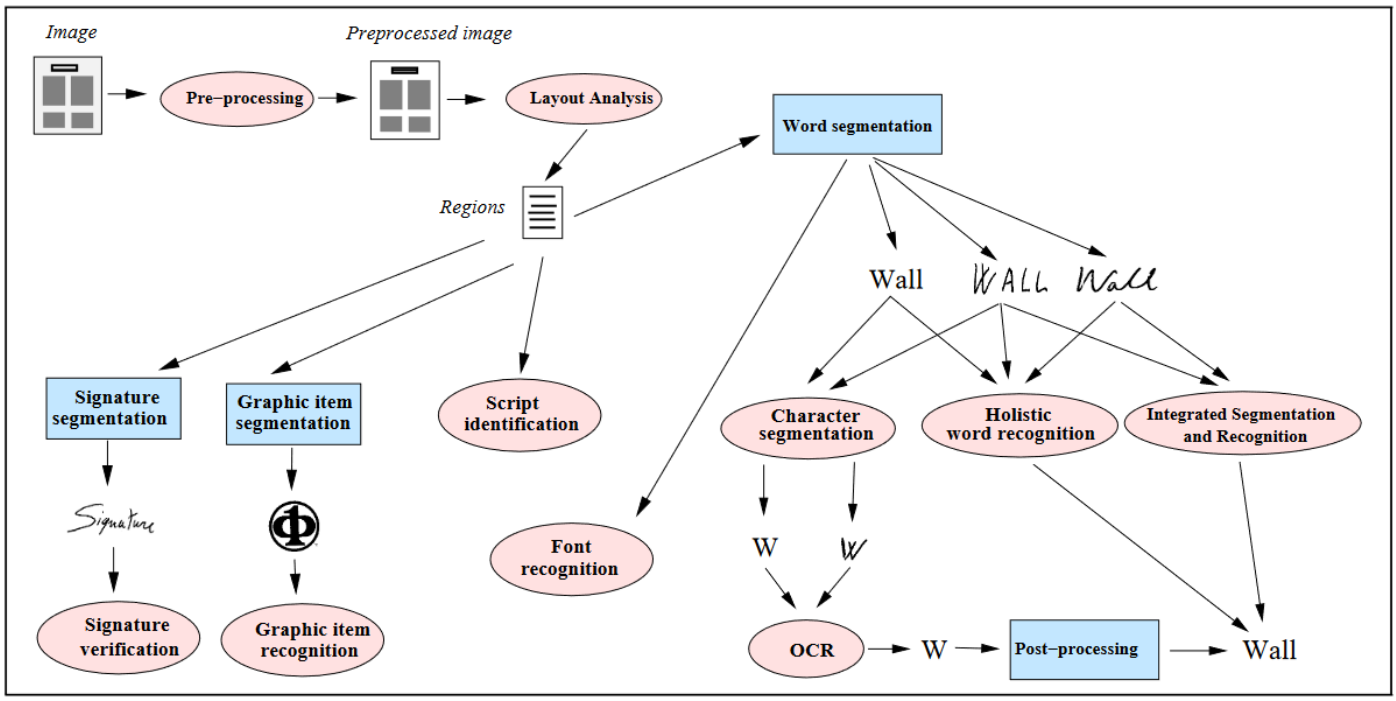
This lab illustrates how to develop a simple document analysis and recognition application in which:

- The Hough transform is applied to deskew document images
- pytesseract library is used to extract and recognise the text in deskewed images

---

Introduction

Document analysis and recognition aims to convert document images from image-based form to text-based form which enables many functionalities such as content retrieval, content searching and indexing, document compression, etc. A general framework of document analysis and recognition is presented in the following figure.



This figure is from M. Simone, "Introduction to Document Analysis and Recognition", Machine Learning in Document Analysis and Recognition, pp. 1-20, 2008.

In this workshop, you will implement a simple document analysis and recognition application. In particular, you will implement document deskewing using the Hough transform and will apply pytesseract, an OCR (Optical Character Recognition) library, for text recognition.

Tasks

1. Hough transform for document skew estimation

Like Task 2.2, in this task, we will investigate the problem of document skew estimation. However, we will explore how to solve the problem using the Hough transform. You can use the supplied document image doc.jpg in Task 2.2 to test your implementation.

Document skew estimation using the Hough transform can be summarised as follows.

Step 1. Load an input image from file and binarise the image using a threshold, e.g., 200 (see Section 2.1 in Task 2.2C for reference).

Step 2. Get negative version of the binarised image by subtracting the binarised image from 255 (see Step 1 in Section 2.2 in Task 2.2C for reference).

Step 3. Extract connected components from the negative image. Note that morphology is NOT used here.

Step 4. Select *candidate points* on the negative image using one of the following strategies

```
a. All foreground pixels in the image are considered as candidate points.

b. All connected components' centres are considered as candidate points.
The centre of a connected component has x- and y-coordinate as the means of
the x- and y-coordinates of all pixels in that connected component.

c. The point which has maximum y-coordinate in each connected component is
chosen as a candidate point.
```

Step 5. Remove all pixels which are not candidate points from the negative image.

Step 6. Create a parameter space, called Hough space. Recall that Hough space has two dimensions representing the closest distance from a line to the image origin and the angle of that line (see week 2 handouts). There are three parameters used to detect lines in Hough space. Those parameters include distance resolution (in pixels), angular resolution (in radian), and density threshold (unit free). Initialise those parameters as follows:
distance_resolution = 1
angular_resolution = $\pi/180$ (i.e., 1^{o}),
density_threshold = 10 (i.e., a line is considered if it goes through at least 10 candidate points). This parameter is set based on the number of text elements in your document image. You will need to vary this parameter for the best performance (e.g., 10, 15, 20).

Step 7. Apply the Hough transform on the negative image using the following command

```
lines = cv.HoughLines(negative_image, spatial_res, angular_res, threshold)
```

cv.HoughLines returns a list of lines. To access the distance and angle of the k-th line, you can use the following code,

```
distance, angle = lines[k][0]
```

Step 8. Create an array to store the angles of all lines detected by Hough transform.

Step 9. Import statistics (see Step 4 in Section 2.2 in Task 2.2C). Then apply statistics.median to the array of angles created in Step 8. This will result in the median angle. However, this angle is NOT the document's angle (see week 2 handouts). The document's angle is orthogonal to the median angle.

Step 10. Deskew the image in doc.jpg with the angle calculated in Step 9 (see Step 5 in Section 2.2 in Task 2.2C).

2. Performance analysis

2.1. Candidate point selection

As described in Step 4, there are different strategies to select candidate points. You are to apply each strategy to doc.jpg and compare all the strategies in terms of accuracy and computational speed. You are required to measure the computational speed of each candidate point selection strategy. To measure the computational speed, you can use time.time(). To use time.time(), you need to import time package.

2.2. Parameter setting

Another factor that also affects the performance of the skew estimation algorithm is the "density_threshold" parameter (see Step 6). You need to run an ablation experiment by varying the values of this parameter (you can try with 3 different values), then analyse the corresponding deskewing results and computational speed.

3. Other test cases

In this task, you are required to validate the skew estimation algorithm on various inputs. To do so, you need to collect a small dataset with at least 5 document images (excluding the supplied ones in Task 2.2). The collected document images must capture documents (e.g., papers, books, magazines, etc.) with bright background and dark foreground (like the supplied document images). The collected document images need to cover possible variations of document images, e.g., different skew angles, text in different font sizes, different languages, and with the presence of noise and non-textual elements such as graphics. You are free to add in any challenges. The images can be created by scanners, built-in cameras in smartphones, or collected from the Internet. If you use any images from the Internet, make sure that have adequate references.

Based on your observations from the deskewing results of doc.jpg, choose the most accurate candidate point selection strategy and the best parameter setting, then test the skew estimation algorithm with your chosen candidate point selection strategy and parameter setting on your collected data. Observe and discuss the results. Does the Hough transform accurately work in every case? If not, what could be the reason and how to address it?

4. Text recognition using pytesseract

pytesseract is a python wrapper of Google Tesseract-OCR. This library can be used to extract text regions from an image and recognise them in different languages. You can find more details of this library at https://pypi.org/project/pytesseract/

To use pytesseract, you will need to install a working version of pytesseract (e.g., pytesseract 0.3.8) and Google Tesseract OCR (please also read through the license of the library at [https://github.com/tesseract-ocr/tesseract])

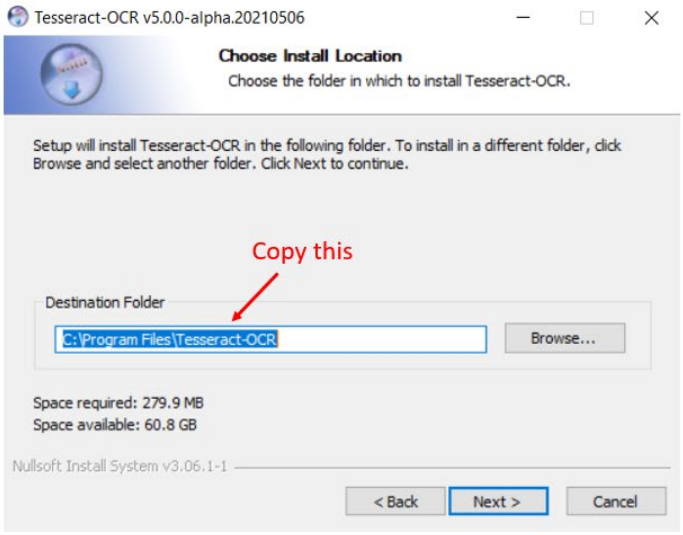
Install pytesseract

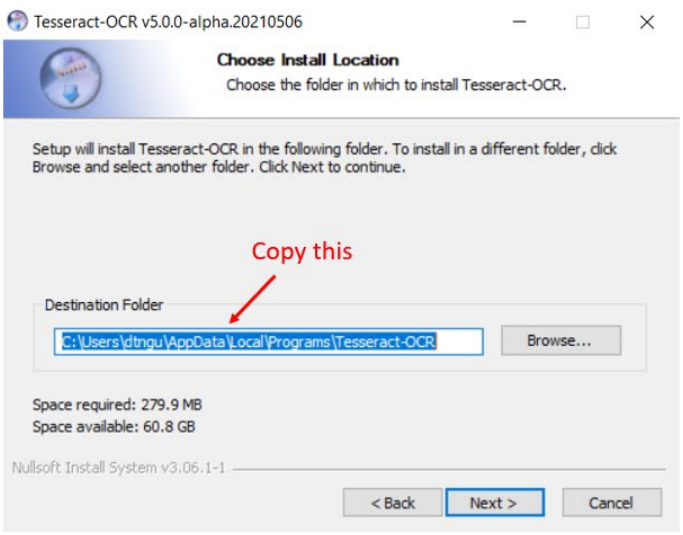
Open a terminal and execute the following command: `pip install pytesseract`

Install Google Tesseract OCR

For Windows

- Access https://github.com/UB-Mannheim/tesseract/wiki
- Download the installer from tesseract-ocr-w32-setup-v5.0.0-alpha.20210506.exe or tesseract-ocr-w64-setup-v5.0.0-alpha.20210506.exe depending your operating system
- Run the downloaded installer. You are then asked to choose if you want to install this library for everyone or for only you. Depending on your choice, copy the destination folder as shown in the following figure (as you will need to use it then).





For Linux (Ubuntu)

- Execute the following commands in your terminal:

```
sudo apt-get update -y
```

```
sudo apt-get install -y tesseract-ocr
```

For Macs

- Execute the following command in your terminal:

```
brew install tesseract
```

Now you are ready to apply the Tesseract-OCR library for text recognition. You first need to import pytesseract into your notebook by doing:

```
import pytesseract
```

If you are using Windows, you then need to run the following command:

```
pytesseract.pytesseract.tesseract_cmd=r'DestinationFolder\tesseract.exe'
```

where DestinationFolder is the location where your Tesseract-OCR is installed. For example, if your Tesseract-OCR is installed in C:\Program Files\Tesseract-OCR (if you were to choose to install this library for everyone), your command is written as,

```
pytesseract.pytesseract.tesseract_cmd=r'C:\Program Files\Tesseract-OCR\tesseract.exe'
```

Note. If you are using Linux or Macs, you can skip this step.

Suppose that you have a document image named img, to recognise the text in img, you can run the following command.

```
text = pytesseract.image_to_string(img)
```

where text is a string containing the text recognised by the Tesseract-OCR library from img. You can print out text to visually check the recognition result.

You are to test the Tesseract-OCR library with the skewed document image give in doc.jpg and its deskewed version obtained using the Hough transform (Section 1), discuss the results, and draw conclusion.

You can also convert the results of OCR into pdf by doing:

```
pdf = pytesseract.image_to_pdf_or_hocr(img, extension='pdf')
```

```
with open('img.pdf', 'w+b') as f:
```

```
        f. write(pdf)
```

Once again, you can test this functionality with other skewed document images and their deskewed versions. If your pdf reader (e.g., Acrobat reader) has text editing tool (e.g., highlight text, underline text, strikethrough text, etc.), you can use this tool to edit pdfs. Given the pdfs of a skewed document and its deskewed version, what pdf you can make more text editing?

Finally, you are to make an entire document recognition system that takes input as a document image, deskews the input document, recognises its text, and returns a pdf of the input document. You should test your system with different document images you have collected in Section 3.

Submission instructions

1. Complete the supplied answer sheet with required results and analysis.
2. Submit your answer sheet (in pdf format) to OnTrack.
3. Submit your code (in .py format) to OnTrack.