

Image to Speech System

Yanqiao Huang

2nd May 2022



Cardiff University

School of Computer Science and Informatics

CM3203 - Individual Project (40 credits)

Final Report

Supervisor: Bailin Deng

Moderator: Yuhua Li

Abstract

There is a large proportion of visually impaired people in the world. Population growth, ageing and the prevalence of electronic devices are expected to increase the risk of developing visual impairment in more people. To help this group of people, many smart phones are already equipped with screen assisted reading function, which helps people with visual impairments to correctly access information on the phone screen by reading it aloud. Despite the popularity of e-reading, many people still have a need to read books, newspapers or menus in paper form. Thus there is a pressing need to develop a system to help people with visual impairments to read and understand the text in images.

My project is building a system that converts the text inside an image to a speech based on OCR (Optical Character Recognition) and TTS (Text to Speech) technology, image pre-process and text post-process functions are also included. The system is optimised for people with visual impairments, they can use it without additional assistance or significant learning time. The system is expected to be deployed on a PC with a camera, and will significantly help people in need.

Acknowledgement

Firstly, I would like to thank my supervisor Dr Bailin Deng for his guidance, clear explanation and help for the problems I encountered during my project.

I would also like to thank my family for supporting my study in Cardiff.

Finally, I would like to thank the school for their support and help.

Content

Image to Speech System.....	1
Abstract.....	1
Acknowledgement.....	2
Content.....	3
1. Introduction.....	5
2. Background.....	7
2.1 Optical Character Recognition (OCR).....	7
2.2 Text to Speech (TTS).....	8
2.3 Natural Language Processing (NLP).....	8
2.4 OpenCV.....	9
3. Design and implementation.....	10
3.1 System Design.....	10
3.1.1 System Architecture Design.....	10
3.1.2 User Interface Design.....	11
3.2 System Implementation.....	13
3.2.1 Programming language selection.....	13
3.2.2 User Interface Implementation.....	13
3.2.3 Image Acquisition Module.....	14
3.2.4 Image Pre-process Module.....	18
3.2.5 Image-to-text Module.....	24
3.2.6 Text Post-process Module.....	32
3.2.7 Text-to-speech Module.....	35
3.2.8 Speech Playing Module.....	35
4. Test and Evaluation.....	37
4.1 System Functionality Test.....	37
4.2 System Evaluation.....	39
5. Conclusions and Future work.....	43
5.1 Conclusions.....	43
5.2 Future work.....	44
6. Reflection.....	46
Reference.....	48

1. Introduction

As per the World Health Organization (WHO), At least 2.2 billion people worldwide are visually impaired. In no less than 1 billion of these people, there visual impairments remain unaddressed[1]. For screen readings, there are already many proven software that can help people understand and recognized texts by reading it aloud. Android and IOS are both equipped with the ability to read aloud the contents of the screen. However, for paper readings, there is still not a proven system that can help people extract the words on the paper accurately then convert them to speech. According to a research in United States[2], 21% of over 65s still use newspapers as a source of news everyday, and older people are the ones with a greater proportion of visual impairment. Thus there is a pressing need to develop a system to help people with visual impairments to read and understand the text in images accurately.

There are already some exist image-to-speech system such as Aspose and Talkie OCR. They are all developed base on OCR and TTS technology, yet they all share some common flaws. One of the greatest concern is the issue of recognition accuracy of OCR. Due to the complexity of the realistic photoing environment and the fact that some user is visually impaired, images used for OCR recognition are often skewed, blurred, and have a level of motion. In the literature[3], the authors state that noise in image, italic Text, more colors in the image, cluttered texts can lead to a significant reduction in the accuracy of OCR recognition. In my opinion, one big reason for low recognition accuracy of the existing systems is that they do not guide the user through the process of taking a image, making it difficult for people with visual impairments to take a clear and recognisable image. Another reason is that many systems do not have pre-process function for the input image, and although some systems can do skew correction, it is limited to flat correction and cannot handle images have an skew in the Z axis. At the same time all existing systems are unable to accurately identify irregularly distributed text, due to the fact that mainstream OCR engines can only recognise regularly arranged text. Another point of concern is that for the text-to-speech function, as many software do not

post-process the text generate by OCR engine, instead they use them directly for TTS conversions. This could lead to problems of incoherence (the same sentence being read aloud in several separate paragraphs), increasing the probability of misunderstanding.

The system I developed is optimised for all of these issues. It can provide guidance for users during the photoing phase, prompting them to place the paper to be identified in the correct position for shot. It will have the image pre-process and text post-process function, improve the accuracy of image recognition and the fluency of speech reading. Also, it combine several OCR engines for text recognition, include the latest PGNet engine, makes it possible to accurately recognize irregularly arranged text. I believe this system can benefit more people with visual impairment to solve their reading problems in the future.

2. Background

2.1 Optical Character Recognition (OCR)

OCR is the progress of checking the characters printed on the paper using a scanner or a camera, and then convert them into computer text according to their shapes. This technology can be used in many scenarios such as ID card text recognition, bank card recognition, the electronicization of paper books, etc. OCR can effectively replace manual recognition, reducing labour costs and making life more efficient. OCR was first proposed by the German scientist Tausheck in 1929, its development can be roughly divided into two stages: Traditional OCR, and deep learning based OCR.

In the literature[4], authors state that traditional OCR processes include image acquisition, image pre-processing, character segmentation, feature extraction, character classification, and post-processing. Traditional OCR can generally only handle relatively simple scenes, once the scene becomes a natural scene, there is high probability that traditional OCR will fail. The deep learning-based OCR approach simplifies the process into two main steps: Text detection and text recognition. Compared to traditional OCR, deep learning-based OCR has a great advantage in recognizing text images in natural scenes. In addition, End-to-end natural scene text detection and recognition has emerged in recent years, some recognition models based on end-to-end text detection like ABCNet (Liu, Yuliang et al. 2020)[5] and PGNet (Wang, Pengfei et al. 2021)[6] can even do one stage text recognition (including detection), they avoid the usage of character-level annotations, which can improve detection accuracy and speed. Those models are capable of detecting and recognizing text of arbitrary shape.

As the system I developed is mainly for text recognition in natural scenes, I will choose deep learning-base OCR as recognition method. There are many open source deep-learning-base OCR library such as Tesseract, PaddleOCR, EasyOCR eetc. However, most of the existing text-to-speech systems do not

specified which OCR library they are using. I will compare different OCR libraries and choose the appropriate OCR library for my system according to the application scenario.

2.2 Text to Speech (TTS)

TTS is a type of speech synthesis application, It can convert the content of a document or text on an application, such as an application menu or a web page, into natural speech output. TTS helps people with visual impairments to read information on computers.

Many companies have launched their own TTS services or APIs like Microsoft, Amazon and Google, yet most of them is not free. My aim is to find a free, Multi-language support TTS library that runs in python. I have located 4 complementary TTS engine: gTTs, pyttsx3, speech and TTS.api. For the moment, little research has been done to analyse the advantages, disadvantages and compatibility of the different TTS engines. I will compare these 4 TTS engine base on their compatibility and functionality, and choose the appropriate one for reading the text result generate by the OCR engine.

2.3 Natural Language Processing (NLP)

The purpose of NLP is to develop applications or services that understand human language. The applications of NLP are Speech recognition, speech translation, understanding sentences, spelling correction, grammar correction, etc. Due to the fact that no OCR engine can guarantee 100% recognition accuracy, misidentification is very common in the image-to-text process. This is where NLP comes in handy, we can use NLP libraries such as TextBlob and ContextualSpellCheck to perform text post-processing, or to calculate the approximate accuracy of the OCR engine by checking the fluency of resulting sentences. Other applications such as sentence translation and sentence comprehension may also be useful for the future development of the system

2.4 OpenCV

OpenCV is open source and released under the Apache 2 License. It is free for commercial use. It's first open source version, opencv alpha3, was released in June 2000, now iterating to 4.7.0. It has the advantage of being cross-platform (Linux, Windows, Android, Mac OS) and supporting multiple programming languages (C++, python, java Matlab). Opencv plays an important role in image processing, it has methods for image reading, image binarization, image noise reduction, image edge detection, etc. In my project I will be using Opencv's cv2 library for image reading and writing, image processing and image capture.

3. Design and implementation

3.1 System Design

3.1.1 System Architecture Design

The system consists of 6 modules: Image acquisition module, image pre-process module, image-to-text module, text post-process module, text-to-speech module and speech playing module.

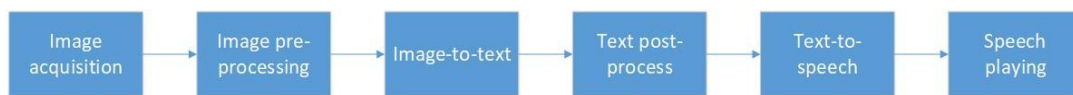


Figure1: System Architecture

Image acquisition module is responsible for image acquisition and input. Users can choose to input images locally or take images using the camera connected to the computer. The system will provide guidance if user choose to take images using the camera, and the images they take will be store locally for further processing and recognition.

Image pre-process module is responsible for processing the image acquire in the previous module before sending into the image-to-text module. This module contains methods for image super-resolution, binarisation, noise reduction, skew correction, sharpening, etc. By using these methods, the accuracy of OCR recognition can be significantly improved.

Image-to-text module is the core module of the system. It's responsible for recognizing and converting the text in the image to a string. Different OCR engines will be used in this module to improve the success rate of text recognition in different scenarios

Text post-process module is mainly for processing the text generate by OCR engine, it includes operations such as sentence connection and auto text

correction, making the result texts more reasonable before sending into the TTS engine. The presence of this module improves the fluency and correctness of the speech generate by TTS engine, and helps the user to better understand the text.

Text-to speech and speech playing module are responsible for converting the text from the previous module and play it aloud. This is the final step of the whole system. User can play, pause, restart or stop audio playback by clicking the button, they can also adjust the volume using a slider.

3.1.2 User Interface Design

As The system is designed primarily for people with visual impairments (but not blind people), meaning that it will be challenging for users to understand and learn to use the system. So the User interface design of my system is very simple, Users only need to understand the function of four key buttons (i.e. input image and play the speech) to perform the basic operations of the system. All buttons and fonts have been designed to be extra large for user to identify easily. The system will also have a tips and help function to give the user information on the status of the system, what the buttons do, etc. The main user interface of the system is as follow:

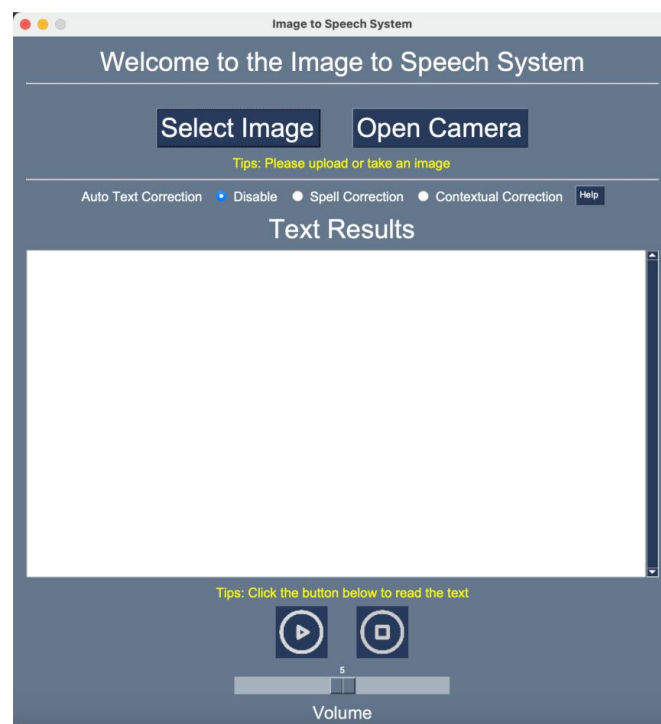


Figure2: User Interface of the system

The User Interface are designed based on Nielsen's 10 Usability principles[7]. The following principles have been applied:

1: Visibility and system status: System status is clearly state in tips function, if a user uploads a file that is not an image, the system will indicate "wrong file type" with a red warning text. If the user uploads the image file correctly, the system will indicate "image upload successfully, converting to text now." in green text.

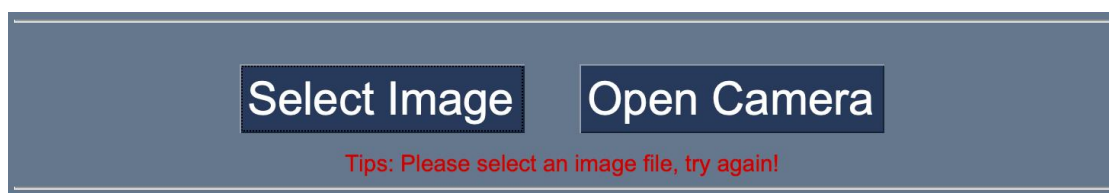


Figure3: File type warning message



Figure4: image input Successful message

2: Match between system and the real world: The play speech button, pause button, restart button and stop button are designed in a way that is consistent with real-world perception, where users can easily understand what the buttons do through association or analogy.



Figure5: Play, pause and stop button

3: Consistency and standard: The UI design is consistent, with all buttons and text centred for easy recognition and operation.

4: Flexibility and efficiency of use: The system is designed to be easy to use, while prompting and providing the necessary guidance to the user (especially in the module for uploading pictures taken with the camera, which will be mentioned later in the implementation). All beginner users will be able to master the system in short time.

5: Aesthetic and minimalist design: Compared to other existing systems, my system removes a lot of unnecessary buttons, keeping only the picture input and audio playback buttons, the spelling correction button and the button to adjust the volume level. Ensure not to make the system cumbersome for users.

6: Help and documentation: I have set up tips and help button to help users understand the system's functionality.

3.2 System Implementation

3.2.1 Programming language selection

I select Python as my programming language. The main reason I chose python is that it has good scalability, many libraries and algorithms for image recognition and processing are developed base on python. I can install many third party open source packages via pip such as Tesseract, gTTS and OpenCV to help me with the system development. The second reason I chose python is because of its simplicity and readability. In the Group Project in last semester I have learned the basic method of using python to develop a Web based software, it would be easy for me to expand those knowledge and create my own system.

3.2.2 User Interface Implementation

I use PysimpleGUI to implement the user interface. PysimpleGUI is an open source, cross platform GUI library for python. It's base on Tkinter, PyQT, and

WxPython. I chose it because PysimpleGUI is more cleaner and efficient compared to other frameworks. It integrates many basic components, such as buttons, pop-up windows, radios and sliders. It also has the ability to listen to events and values, then provide feedback base on users actions.

The user interface is divided into three main sections: Image acquisition section, text result display section, and speech playing section. Each section is separated by PysimpleGUI (known as sg) HSeparator to keep users from getting confused. Image acquisition section is implemented using sg.Button and sg.popup_get_file. Text display section is implemented using sg.Multiline, while speech playing section is implemented using sg.Button and sg.Slider. The theme I used is PysimpleGUI's classic Brown Blue.

3.2.3 Image Acquisition Module

There are two ways to acquire image in the system: Choose an image file to upload from locally or take a picture and upload it using the camera connected to the computer.

Upload an image file locally

For choosing image file locally, popup_get_file module provided by PysimpleGUI. PysimpleGUI has wrapped the entire pop-up window, including Browse, Ok and Cancel button. When the user selects the file, the system return the path to the file.

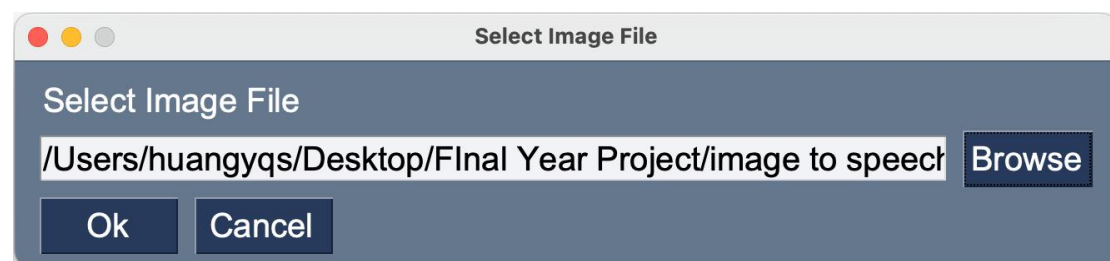


Figure6: Pop-up window for selecting file

The parameter file_types of sg.popup_get_file should be able to restrict the types of files that can be uploaded. However, according to warning from the

official documentation of PysimpleGUI[8], the `file_types` parameter will not work on Mac platform for the Tkinter version of PysimpleGUI. Considering that my system is designed to be cross-platform, to avoid the system from crashing, I have added additional judgements to restrict the file type, the system will only process forward if the user select main stream image file (i.e. jpg, png, bnp, jpeg). Otherwise, a warning message will show up to remind users to select again.

Taking an image using the camera

For taking a image using a camera connected to the computer, due to the complexity of the real world environment, it is likely that the user will not be able to take a photo that is suitable for OCR recognition. To solve this problem, I added some guidance to the camera module, which shows the position of the paper and the skew angle in real time, helping the user to take a better quality photo. The interface is as follow:

When the system can't detect any paper (The system can't detect a lion):

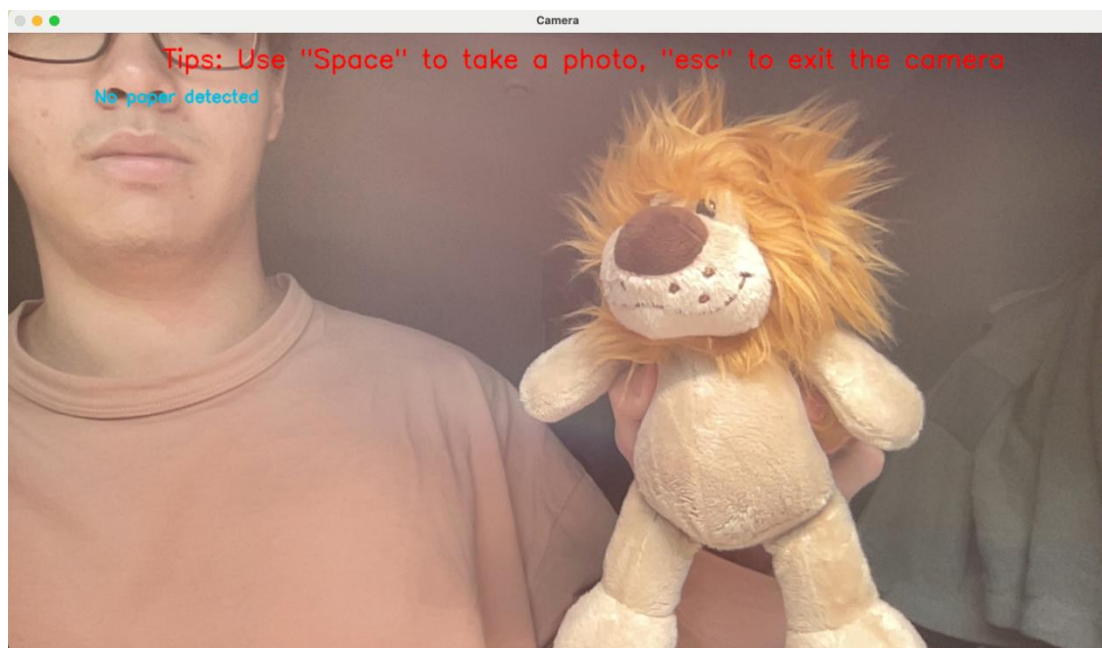


Figure7: No paper detected window

When the system detects a paper and it's in good position (Consider the paper is the article with black borders, while others are background):

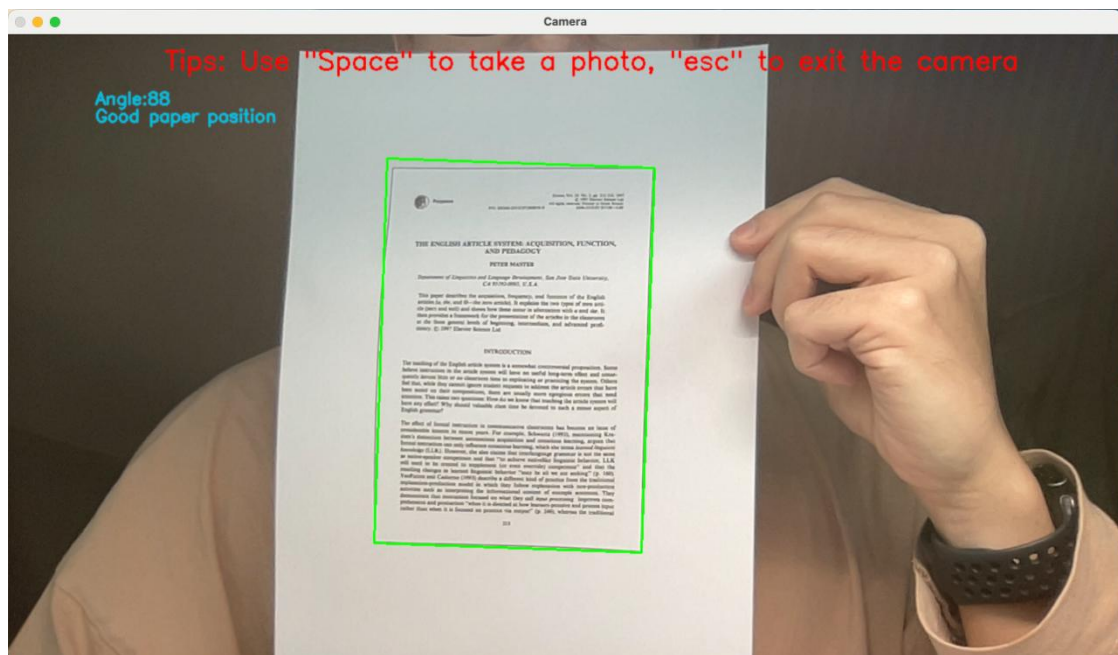


Figure8: System detects an image in good position

When the system detects a paper and it's skew for more than 25° , the color of the contour will be changed to yellow:

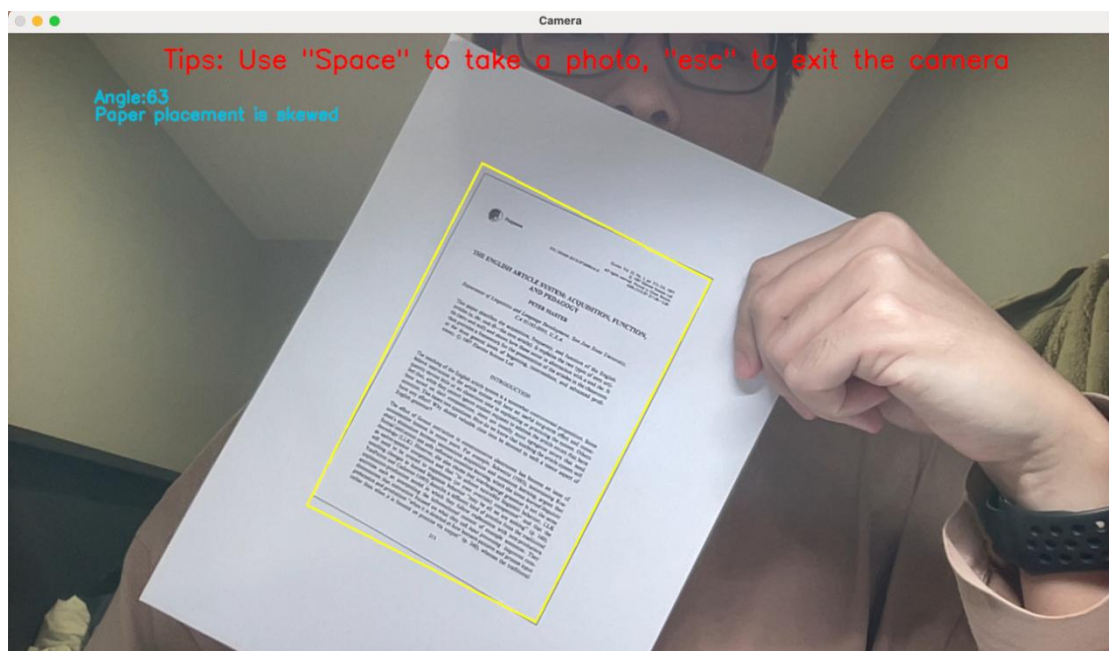


Figure9: System detects an skew image

Image Capture interface is implement using VideoCapture function of cv2 library. It is able to call up the camera connected to the computer and display each frame captured by the camera in real time. What I've done with the guide

function is essentially to process each frame, detecting the contour of the paper and putting it in a green box. Then I calculate the skew angle and use `cv2.putText` function to include the skew information in the image. For instructions of using the camera, I have added a tip at the top of the image, informing users to use space button to take an image, and esc button to exit the camera.

As the vast majority of paper is quadrilateral, the method for detecting paper is to detect the contours of the quadrilateral in the image. A method describe in [9] perform a efficient and accurate detection of the contours of quadrilateral. We first perform grey out, Gaussian noise reduction, and Canny edge detection to the image using methods from `cv2` library, then use `cv2.findContours` method to find all the counters and save them in a list. After which, we sort the list in descending order and keep the first five contours. Finally we approximate the contours using `cv2.approxPolyDP` method to get the desired quadrilateral contour. Noted that the accuracy of the contours approximation can be obtained by multiplying the contours perimeter by 0.02. After getting the contour we want, we can use `cv2.minAreaRect` method to calculate the angle from the lowest edge of the contour to the X-axis, i.e. the skew angle. The principle of the `detect_paper` method is as follow:

- 1: Image pre-process (Graying, Noise reduction, Canny detection)
- 2: Find all the contours in the image
- 3: Sort all the contours in descending order by the size of their area, and select the first five.
- 4: Use `approxPolyDP` function to fit these five contours, if the result is a contours consisting of 4 points, take it as the final result.

Figure10: The principle of `detect_paper` method

Noted that this method will be also used in the image skew correction process in 3.2.4.

3.2.4 Image Pre-process Module

As most of the users of this system are visually impaired, and the real-world photographing environment can be very complex, putting the original image directly into the OCR engine for text recognition may Resulting in a reduction in accuracy. To solve this problem, I designed an image pre-processing module image_pre_process.py to pre-process images before sending them into the OCR engine. This module can improve the accuracy of text recognition.

Image super-resolution

In literature[17], the author states that low image resolution may lead to a reduction in recognition accuracy, and the recognition accuracy will dramatically drops with image resolution below 300 dpi. Generally speaking, OCR engines work better for images with higher resolutions. To test this theory, I have made a test on the recognition accuracy of Google Tesseract (will be mention in chapter 3.2.5) of the same image under different resolution. The image was shot by iPhone12, the original resolution is 4032×3024, I used Photoshop to adjust the resolution to 1024x768 in order to make a comparison, the result is as follow:


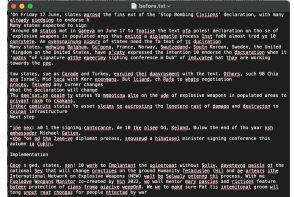
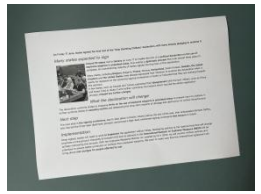
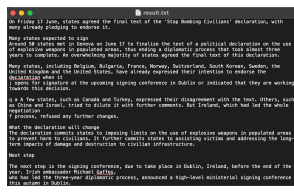
Image	Resolution	Result	Accuracy
	1024×768		165/352=46.8%
	4032×3024		347/352=98.5%

Table1: Recognition accuracy of the same image under different resolution

From the test results we can see that image resolution has a significant impact on recognition accuracy. There are two ways to increase the image resolution. The general way to simply resize the image by multiply his width and height by the magnification factor. However, this method does not help much to improve the image quality, since it only enlarges the size of the image and does not

improve the sharpness of the image. The second way is to use deep-learning based image super-resolution method. OpenCV offers four main deep-learning based image super-resolution algorithms: EDSR, ESPCN, FSRCNN, and LapSRN. In article[13], the author have made a test on each model, calculating the the mean PSNR values of the output images and the mean running time of each algorithm.

Name	2x	3x	4x	8x
Bicubic	0.859	0.794	0.728	0.552
EDSR	0.885	0.825	0.762	–
ESPCN	0.877	0.799	0.736	–
FSRCNN	0.876	0.798	0.735	–
LapSRN	0.874	–	0.735	0.554

Table2: Mean PSNR values in dB in test in[13]

Name	2x	3x	4x	8x
Bicubic	0.00099	0.00099	0.00098	0.00098
EDSR	32.501	16.718	10.224	–
ESPCN	0.049	0.032	0.018	–
FSRCNN	0.074	0.035	0.054	–
LapSRN	0.501	–	0.742	0.765

Table3: Mean time in seconds in test in[13]

From the test result we can see that EDSR seems to get the best output quality among the four algorithms, however, it runs more than 100 times slower than other methods. For ESPCN, the result PSNR value is a little lower (-0.026) than EDSR, but it's running speed is 522.4 times faster than EDSR. In this case, I will choose ESPCN as my super resolution algorithm since it's more

efficiency. This algorithm can effectively increase the resolution of the image.

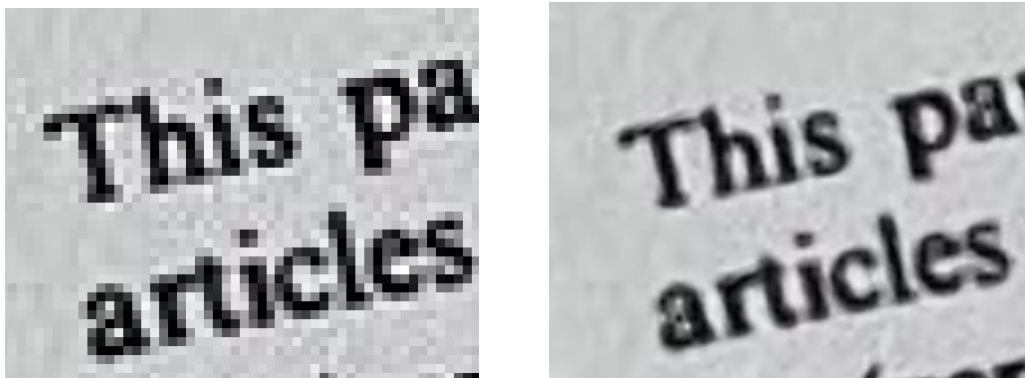


Figure11: Before-After of image super resolution

After applying the image super resolution method, I have run the image recognition process again on the low resolution image in table1 (Apply super resolution algorithm on it), the result is as follow:

Image	Resolution	Accuracy
Original image	4032×3024	98.5%
Low resolution image	1024×768	46.8%
Super resolution image	3072×2304	93.4%

Table4: Test on the super resolution image

As we can see from table 4, although the recognition accuracy of the super resolution image is still lower than the original image (98.5%), but it's a lot better than the low resolution one (46.8%), the image super resolution process can effectively increase the recognition accuracy.

Image skew correction

In literature [17], the author also states that OCR recognition is very sensitive to geometric deformations, slight rotation angle can cause OCR recognition progress fail completely. So we have to perform image skew correction before OCR progress. There are two ways to do image skew correction: 2D correction and 3D correction. 2D skew correction is suitable for images that are not skew on the Z-axis. However, we can't guarantee that users are always perpendicular to the z-axis when taking pictures, so I chose 3D correction as my main correction method.

3D skew correction went through two steps: Contours detection and Perspective transformation. For contours detection, we can again use the method

mentioned in [9], the detailed steps has already been discuss in the image acquisition module. After contours detection, I sort the four points obtained in the order of lower right, upper right, upper left, lower left, to allow the picture to be rotated at the correct angle. Then we need to do the Perspective transformation, project the target into a new view plane.

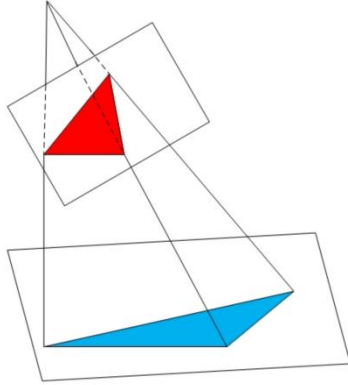


Figure12: Perspective transformation

$$\begin{bmatrix} x' & y' & \omega' \end{bmatrix} = \begin{bmatrix} u & v & \omega \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Figure13: General transformation formulas

To perform perspective transformation, we have to first obtain a 3x3 matrix of variations. To obtain this matrix, we need the four points we get in the previous step, and their corresponding positions on the output image. We first need to calculate the width and height of the output image according to those four points (using the formula for the distance between two points in a straight line), then we can get the corresponding position of lower right, upper right, upper left, lower left on the output image, which is (width, height), (0, height), (0, 0), (width, 0). After which, we can use `cv2.getPerspectiveTransform` method to calculate the transform matrix, and use `cv2.warpPerspective` method to finish the perspective transformation.

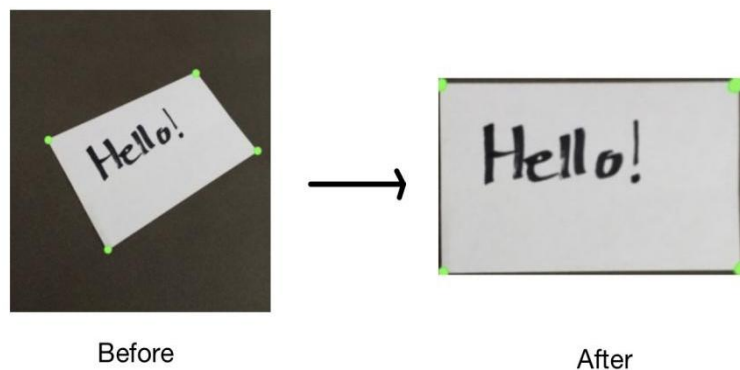


Figure14: Before-After of perspective transformation

However, there is no guarantee that we will always be able to identify the

quadrilateral contours every time. If the paper is triangular, or pentagonal, then this 3D skew correction method will not work. To solve this problem, I used 2D skew correction as an alternate method. If the system can't successfully detect 4 points as contours, it will switch to 2D skew correction. There are two mainstream 2D skew correction method: skew correction base on Hough transform (Hough correction) and skew correction base on projection profile (projection correction).

For Hough correction, in the article[10], the author states that it is the correction of skewed images using the Hough transform. The principle is to detect straight lines from the image and then determine the skew angle of the image based on the angle between the lines, and finally finish the skew correction based on the skew angle.

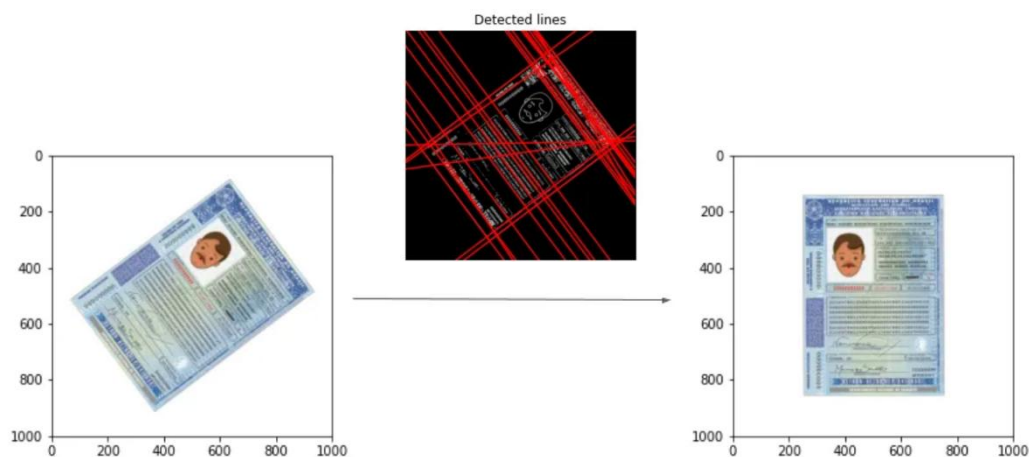


Figure15: The progress of Hough correction in [10]

An open source python library deskew can determine the skew angle of an image base on Hough transform. I first used its `determine_skew` method to detect the skew angle, then I perform affine transformation to correct the skew and get the result image.

For projection correction, in the literature[11], the author state states that the principle of projection correction is to calculate horizontal projection profiles over a range of angles. Features are then extracted from each projection profile to determine the angle of inclination. The profile with the greatest variation is the one with the best alignment to the text line. In figure 14, we can see that the

de-skewed image has a higher calculated peak and clearer spacing than the skewed image.

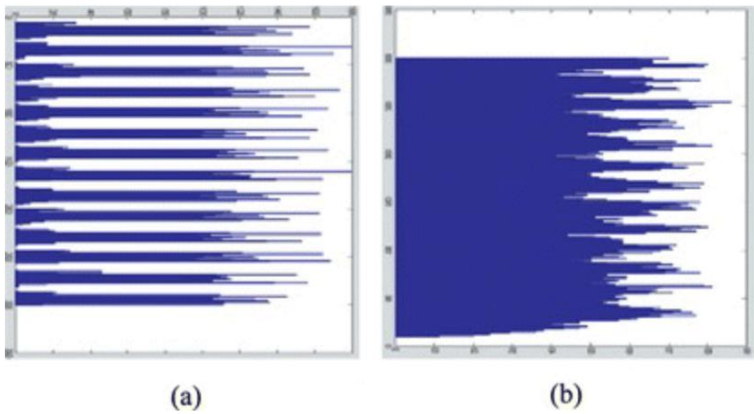
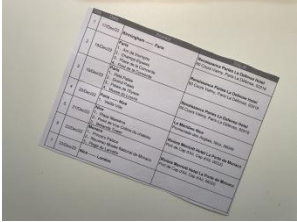


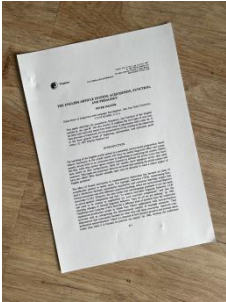
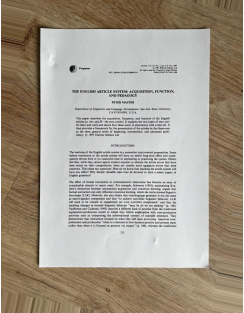
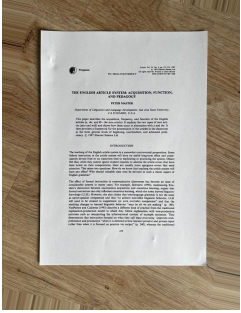


Figure16: Calculated Horizontal projection profile in [11]

The biggest drawback of skew correction base on projection profile is that it needs to calculate the projection profile for each angle within the set angle in order to get the best angle. Such calculations can be time consuming.

In order to compare the performance of these two correction methods, I used three different images, perform Hough correction and Projection correction on them respectively, and recorded the time required for correction using the time() function, as well as their running result image. The result is as follow:

Test image	Running time	Hough Result	Projection Result
	Hough: 0.58s Projection: 26.2s		
	Hough: 0.48s Projection: 26.8s		



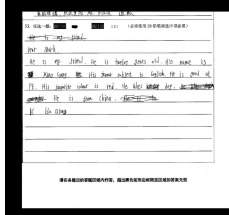
	Hough: 0.54s Projection: 35.9s		
---	-----------------------------------	--	---

Table5: Run times and results for Hough and Projection correction

According to the test result, both methods a correct result. However, skew correction base on projection profile runs more than 50 times slower than Hough correction. To improve the speed of the system, I chose Hough correction as the 2D skew correction method.

Image binarization

In literature[18], the author states that OCR engine does not work well with colour images, so we have to binarise the images to improve the recognition accuracy. The binarisation process changes the colour range of the image to only 2 values, white and black, which can increase the accuracy of OCR recognition. The binarization process is very simple thanks to the OpenCV library. I first use cv2.cvtColor to greyscale the image, then I use cv2.threshold to binarize the image. I set the threshold at 130, any pixel point with a greyscale value greater than 130 will be set to 255, while others set to 0.

3.2.5 Image-to-text Module

Tesseract

Tesseract[12] is a open source OCR engine developed by Google. The latest Tesseract 4 uses a neural network (LSTM) based OCR engine. I chose it because it supports more than 100 languages and many image formats include PNG, JPEG and TIFF. It uses a line-by-line recognition method and supports punctuation recognition, making it ideal for recognising articles.

However, Tesseract has three main problems. The first is that it is less accurate when the quality of the image is not good (e.g. poor lighting, skewed images). I have optimised the problem using the image pre-process module. The second

problem is that as Tesseract uses a per-line recognition method, it will automatically add line break in a sentence if it is spread over different lines, and the sound file generate by TTS engine will also break since it will be considered as two separate sentences. This problem will be solved by the text post-process module mentioned later. The third one is that Tesseract can only accurately recognise neatly aligned text. The accuracy of recognition of irregularly distributed text in natural scenes is very low. For the two images below, Tesseract can't recognise any text.



Figure17, 18: irregularly distributed text in natural scene

If users upload images with irregularly distributed text in natural scenes, the system can't provide an accurate result with only the Tesseract engine. According to literature[19], end-to-end text recognition methods provide much better recognition accuracy than traditional methods in natural images. So I will combine it together with Tesseract to achieve a higher accuracy.

PGNet

There are several end-to-end arbitrary shape text recognition models such as TextDragon, Mask TextSpotter, ABCNet, PGNet, etc. PGNet is a single-shot text spotter. In the literature [6], the author states that PGNet is able to avoid the use of character-level annotations and does not involve NMS and ROI operations, which can improve recognition efficiency. They also use graph refinement module (GRM) to optimise recognition and improve end-to-end performance. According to their test on the recognition performance and accuracy on Total-Text among different end-to-end models, PGNet is at least twice faster than ABCNet (Liu et al.2020). So I choose PGNet as the

alternative recognition method.

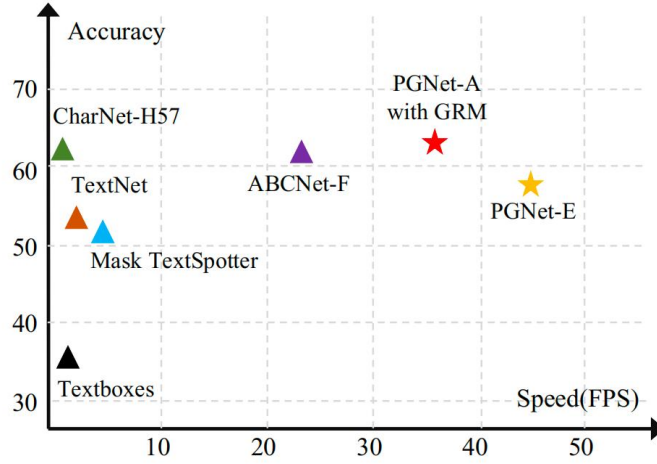


Figure19: Performance and accuracy on Total-Text among different end-to-end models in [6]

(Wang et al.) in [6] states that the recognition principle of PGNet is divided into two main sections. First is extracting features from the input image and learn TCL, TBO, TDO, TCC graphs as a multi-task problem. Then finish detection and recognition of text in once by polygon repair and PG-CTC decoding mechanism, using the sequence of center points in the text area.

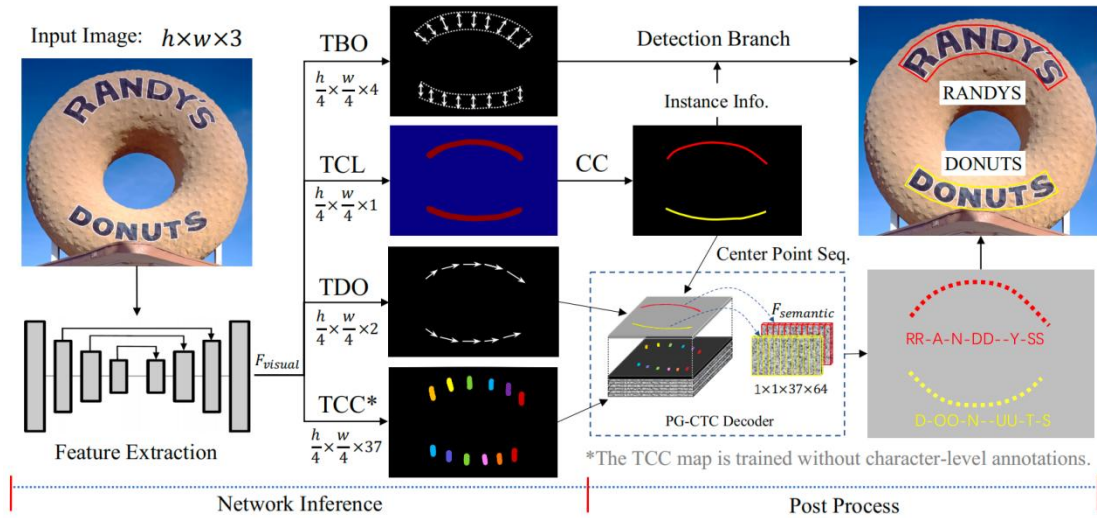


Figure21: The principle of PGNet [6]

However, if I were to apply PGNet directly to my system, few problems may occur. One of the biggest problem is that PGNet can't recognise punctuation, meaning that it cannot separate sentences from the articles correctly. In this case, I can't use PGNet as my only OCR engine. Another problem is that if an image containing multiple blocks of text, the order in which PGNet outputs

texts may sometime be incorrect. This is because PGNet output texts in the order of their Y-axis coordinates (whichever text has a larger Y-axis coordinate will be output first). Since the intention of the PGNet authors was to calculate both the texts and their coordinates, then put the text on the original picture, instead of outputting them as a sentence. So the order in which the texts are output has no effect on the result.



Figure22: Output image of PGNet



Figure23: Output image of PGNet

However, in my system, I need to read out the text in the image to help users understand their meanings. The order in which the text is read becomes very important, as the wrong order of reading may lead to misunderstandings. For figure22, the original output order of PGNet is “English heritage alan turing 1912 1954 code breaker and pioneer of science computer born was here”. This is because the Y coordinates of ‘science’ is larger than ‘computer’, and Y coordinates of ‘born’ is larger than ‘was’ and ‘here’. For figure23, the original output order is “Tea house tasting wine gastronomy”. The reason is the same as the Y coordinates of ‘Tasting’ is larger than ‘Taste’. This clearly does not fit our left-to-right reading order. So I design an algorithm to adjust the output order of the texts. The principle is as follow:

- 1: Calculate the average X and Y coordinates of the box containing the text.
- 2: Sort them base on the size of Y coordinates.
- 3: Check the Y coordinates, if their difference is less than 2% of the height, reorder them according to the size of the X coordinate.

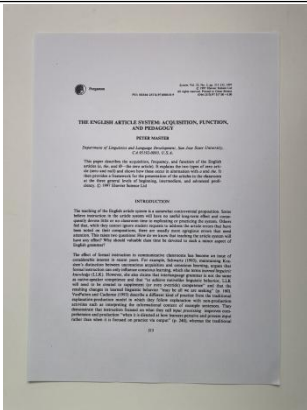

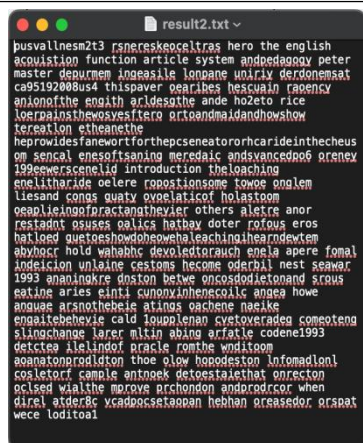
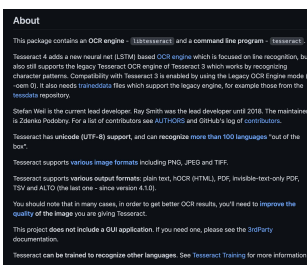
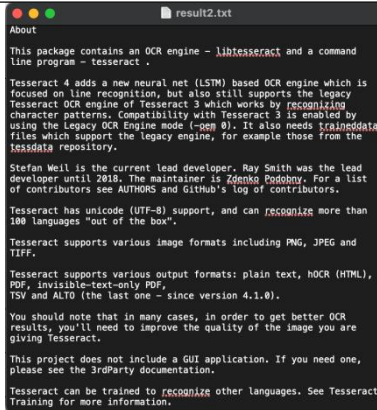
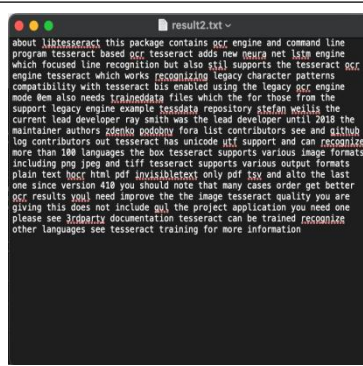

Figure24: The principle of output order of the texts

This algorithm solves the problem of text on the same line being disordered due to small differences of their Y coordinate. If the algorithm determines that

they are on the same line (The difference of their Y coordinates are less than 2% of the height), it will sort them again according to their X coordinate size, This is in line with our top-to-bottom, left-to-right reading habits. After applying my algorithm, figure22 and 23 can be recognize in a correct order.

Compare between Tesseract and PGNet

To better compare the advantages and disadvantages of the Tesseract and PGNet, I used them to recognize several images in different scenes respectively, the result is as follow. (To make the results more readable, some of the recognition results have been post-processed):

Test image	Tesseract result	PGNet result
		
		
	N/A	perfect potion

	TEA oy WINE TASTING — GASTRON f }	tea house wine tasting gastronomy
	Een, 4 MEMORI iv DTANIUNG ASAM	selamat datang memori dtanjung asam pusat mini reaktas tel 016 4670126 pak teh

Table6: Recognition result of Tesseract and PGNet

From the first two result we can see that Tesseract is much more accurate than PGNet in recognising long articles (especially those with small fonts). And since Tesseract can recognise punctuation, the results are highly readable, where PGNet's results are just irregularly arranged text. However, when recognising images containing irregularly arranged text in natural scenes such as the last three images, Tesseract basically fails to get a result, and even if it does, the result is incorrect. PGNet has a high accuracy rate for this type of image, it can get the correct result (or close to it) every time.

After this test, we learned that Tesseract is suitable for recognising long articles with punctuation and regularly arranged fonts, while PGNet is suitable for recognising short and irregularly arranged fonts without punctuation in natural scenarios.

Switch between Tesseract and PGNet

We now know the scenarios in which the two algorithms are applicable. Since our system is designed primarily for people with visual impairments, it is not appropriate to let them manually select the recognition algorithms based on the type of images they upload. The system should be able to automatically switch between Tesseract and PGNet to get the correct result.

Since Tesseract is able to recognise punctuation, and my system is primarily used to recognise articles, I will use Tesseract as my main OCR engine. One automatic switchover method is to switch to the PGNet algorithm for

recognition when Tesseract does not have any recognition results. This approach is based on the fact that Tesseract does not produce any results for the most of the recognition of irregularly arranged text in natural scenes (images similar to the third image in Table4). However, there are also some images where Tesseract will generate incorrect results, these results often contain a large number of grammatical errors, and they are not readable (images similar to the fourth and fifth image in Table4).

To deal with this situation, I used an open-source library contextualSpellCheck[14] to check the result generate by Tesseract engine. contextualSpellCheck is a deep-learning based grammar error detection and correction tool, which can detect grammatical errors in sentences according to their context and the word itself, give improve suggestions and help correct them automatically. Unless the text in the image being used for recognition is itself seriously misspelled and grammatically incorrect (Which is not common in published books and newspapers), otherwise we can assume that the spelling and grammatical errors in the results are caused by recognition errors in the text-to-speech process. In this case, with the help of contextualSpellCheck, I have designed an algorithm to calculate the approximate accuracy of the result generate by Tesseract, the principle is as follow:

- 1: Counting the number of words in the result.
- 2: Counting the number of suggestions contextualSpellCheck generates (i.e. how many words in a paragraph need to be improved)
- 3: Dividing the number of suggestions by the total number of words.
- 4: Get the approximate accuracy

Figure25: The principle of accuracy calculation algorithm

However, the accuracy obtained this way is not guaranteed to be 100% correct. ContextualSpellCheck only provides ‘suggestion’ but not ‘fact’, and if the text we identify is already grammatically incorrect, even if the OCR engine recognises them correctly, contextualSpellCheck will judge them as errors. In this case, we need to set an accuracy threshold (i.e. Tesseract results below the threshold will be considered as incorrect and need to be switched to PGNet for re-recognition). I selected 30 natural scenes images from Total-text dataset and

use Tesseract engine to extract the texts in the images, only 10 images have output result, then I have calculated their accuracy with my algorithm, the result is as follow:

image					
accuracy	0.75	0.6	0.0	0.454	0.667
image					
accuracy	0.615	0.692	0.33	0.667	0.0

Table7: Accuracy of Tesseract recognises Natural scenes images

As we can see from the result, the average accuracy of Tesseract recognises irregularly arranged text in natural scenes is 47.75%, with a maximum of 75%. To compare with the accuracy of Tesseract recognises articles with regular alignment of fonts, I have also select 10 article images and calculate their accuracy with my algorithm. For reasons of space, I have shown four of them below as examples:

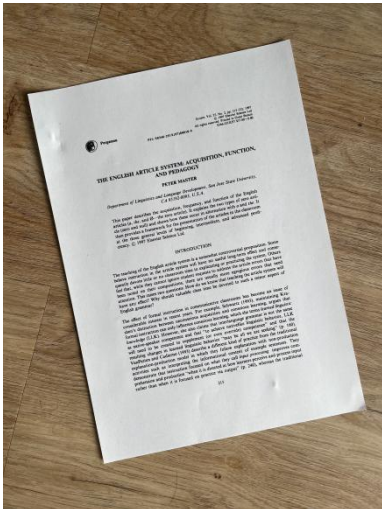
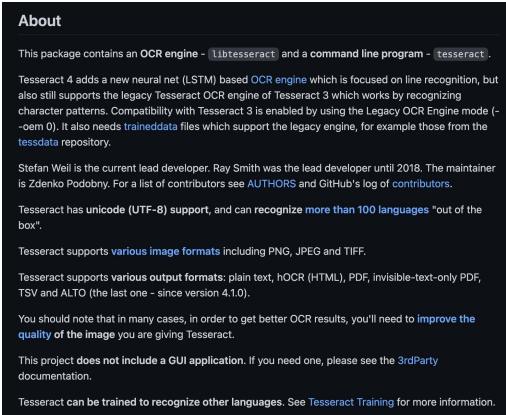
image		
accuracy	0.929	0.906

image	<div> Application Computer Vision Deep Learning Image Processing OpenCV Tutorials Paper Overview </div> <p>Introduction</p> <p>Super-resolution refers to the process of upscaling or improving the details of the image. Follow this blog to learn the options for Super Resolution in OpenCV. When increasing the dimensions of an image, the extra pixels need to be interpolated somehow. Basic image processing techniques do not give good results as they do not take the surroundings in context while scaling up. Deep learning and, more recently, GANs come to the rescue here and provide much better results.</p> <p>The image given below illustrates super-resolution. The original high-resolution image shows the best details when zoomed in. The other images are achieved after reconstruction after using various super-resolution methods. You can read about them in more detail here.</p>	<p>DESCRIPTION</p> <p>tesseract(1) is a commercial quality OCR engine originally developed at HP between 1985 and 1995. In 1995, this engine was among <i>the top 3 evaluated</i> by UNLV. It was open-sourced by HP and UNLV in 2005, and has been developed at Google since then.</p>
accracy	0.938	0.914

Table8: Accuracy of Tesseract recognises articles with regular alignment fonts

In the result (out of 10 images), the average accuracy of Tesseract recognises articles with regular alignment fonts is 92.87%, with a minimum of 90.06%. This means that articles with neatly arranged fonts are identified with a high degree of accuracy. In this case, I set the threshold to 85%. Due to the time consuming nature of each round of accuracy calculation, and the fact that PGNet is not suitable for long article recognition, to improve the efficiency of the system, I have also set a threshold to turn on accuracy detection algorithm. The accuracy detection algorithm will only be activated if Tesseracts generates a result that has less than 70 characters. The overall principle of auto-switch algorithm is as follow:

```

result = Tesseract recognition(image)
if (length(result)==0):
    result = PGNet recognition(image)
else if (length(result)<70):
    if (detect accuracy(result) <0.85):
        result = PGNet recognition(image)

```

Figure26: Principle of the switch algorithm

3.2.6 Text Post-process Module

Tesseract text post-process

As Tesseract uses a per-line recognition method, if the same sentence is spread over different lines, Tesseract will split it with a line break, and the TTS engine will read it as two separate sentences (i.e. there will be a pause in the reading

process). Also, if a word is spread over different lines and connected by a '-' sign, the word will end up being read in two parts. To deal with this situation, I have designed a text post-process algorithm to connect the breaking sentence and words generate by Tesseract, it mainly works by determining whether the sentence has ended based on the symbol at the end of the sentence, and if it hasn't then the algorithm will connect it to the previous sentence. For header lines (generally less than 25 characters), the algorithm will treat it as ended sentence. The overall principle and example running result of the algorithm is as follow (for readability, the example ignores the condition $\text{length}(\text{line}) > 25$):

```

for each line:
  if (length(current line)>25):
    if (last line ends with (". , ! ?")):
      place the line as a new line.
    else if (last line ends with("-")):
      connect the breaking words.
    else:
      connect this line with last line.
  else if (length(current line)<25):
    place the line as a new line.

```

Figure27: Principle of text post-process algorithm

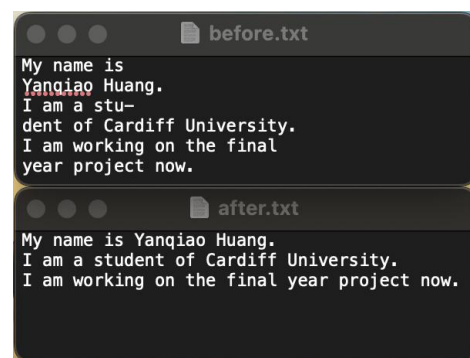


Figure28: Before-After of the algorithm

The actual code is a little more complex than the principle above, since we also need two variables `end_sentence` and `connected_word` to record if the current sentence ends or if the last word of the current line needs to be connected with the first word of the next line, with these two variables, we are able to make a correct determination of how the current line should be handle.

Auto text spelling correction

Even the most advanced OCR cannot guarantee 100% accuracy. In the process of OCR recognition, some similar letters are often being incorrectly recognized, such as "a" being recognized as "e", upper case "i" being recognized as lower

case "L", "0" being recognized as "O", etc. A single letter error can cause the TTS engine to mispronounce the entire word. To improve this situation, I have added two open-source text spelling correction algorithms to the system: Textblob and contextualSpellCheck.

In literature[15], the author propose a method to improve the accuracy of OCR recognition by using Textblob for spelling correction. TextBlob[16] is a open source Python library for processing textual data, it provides a simple API for natural language processing (NLP) tasks such as sentiment analysis, spell correction, and translation. The spell correction function of TextBlob focuses on correcting spelling errors in individual words. ContextualSpellCheck[14] has already been mention in 3.2.5, it mainly focus on correcting the spelling of words according to the context. An example of using TextBlob and contextualSpell check to correcting errors is as follow:

Original Sentence	my <u>neme</u> is Yanqiao, I am <u>working</u> on my final yaar <u>projaect</u>
TextBlob	my name is Yanqiao, I am working on my final year project
contextualSpellCheck	my name is Yanqiao, I am working on my final day.

Table9: Example of TextBlob and contextualSpellCheck

However, there are limitations to this approach, as existing spell-checks require some human intervention to ensure accuracy. The most accurate process is when a spelling error occurs, the algorithm automatically detects the error and suggests a candidate fix, then we can select the correct fix manually. However, it was impractical to use this process in my system, so I chose to let the spell-checking algorithm correct the errors to the words it thought were most likely. In general, spelling correction methods can be helpful in improving the accuracy of recognition results. But due to the fact that it can be error-prone, I leave the choice of turning on the algorithm to the user. The auto spelling correction algorithm is turned off by default when the system is initialised, users can manually turn it on and choose to use between TextBlob or

contextualSpellCorrection, the system will then remember user's choice after the selection is completed.

3.2.7 Text-to-speech Module

Under the python environment, there are 4 main free open-source TTS libraries: Google gTTS, pyttsx3, TTS.api. In order to choose the TTS library that most suit my system, I have made a comparison to both these libraries. The result is as follow:

Library Name	Network Required	Multilingual support	Speed Adjust	Sound quality
gTTS	Yes	Yes	No	Good
pyttsx3	No	Yes	Supported	Good
speech	No	Yes	No	General
TTS.api	No	Yes	No	Poor

Table10: Compare between different TTS libraries

From table7, it seems that pyttsx3 is our best choice. However, I'm developing my system using python3.8, there are compatibility issues with pyttsx3 and python 3.8. pyttsx3.say() function works perfectly fine, but when I want to use pyttsx3.save_to_file function, it will generate an empty mp3 file. My plan is to first generate and save the sound file locally before using another library to read the file (Pygame, which will be mention in chapter 3.2.8). In this case, I choose Google gTTS as my TTS engine.

3.2.8 Speech Playing Module

In this module, I used an open-source library Pygame to read and play the music file we get from the text-to-speech module. Pygame is a cross-platform set of python module designed for writing video games, the music module of Pygame can read and play audio files of different formats. It also includes advanced operations such as adjusting playing speed or playing position, which

is better than the operation provided by gTTS.

The speech playing module consists of 4 buttons: “play”, “pause”, “restart”, and “stop”, as well as the volume slider. “Play”, “pause” and “restart” button is placed in the same place. The “Play” button combines audio file loading and playing function, the logic of the button arrangement is as follows:

- 1:The “play” button will be valid only if the OCR progress is finished.
- 2:After the “play” button is pressed, the “pause” button will appear, while “play” and “restart” button will be hidden.
- 3:After the “pause” button is pressed, the “restart” button will appear, while “pause” and “play” button will be hidden.
- 4:After the “restart” button is pressed, the “pause” button will appear, while “restart” and “play” button will be hidden.
- 5:After the “stop” button is pressed, the “play” button will appear, while “pause” and “restart” button will be hidden.

Figure29: Logic of the 4 different buttons

For volume adjustment sliders, Pygame gives a volume adjustment range of 0 (minimum) to 1(maximum). To make the displayed values more in line with real-world common sense, I have enlarged the displayed numbers of each value by ten times. Users can adjust the volume value between 0 and 10, the default value is 5.

4. Test and Evaluation

4.1 System functionality test

To ensure that the system functions properly, I have tested the six major modules (Image acquisition module, Image pre-process module , Image-to-text module, Text post-process module, Text-to-speech module, Speech playing module) of the system. The test table is as follow:

Test content	Test Process	Expected result	Test result	Note
Acquire image from local folders	Click the “select image” button, first choose an correct format image file and upload it, then choose a non-image file, upload it again.	For the correct format image file, the “image upload successfully, converting to text now” tips will appear, and the system will automatically begin OCR progress. For the non-image file, the “Please select an image file, try again” tips will appear.	PASS	In windows system the non-image file won’t be available to select. The step of selecting non-image files can be skipped
Acquire image from the camera connect to the computer	Click the “Open camera” button, use “Space” button in the keyboard to take an image, click “Esc” button to exit the camera.	When entering the camera page, the system will recognise the edges of the paper and displays the angle of tilt. When the user finishes photoing and exits, the “image upload successfully, converting to text now” tips will appear, and the system will automatically begin OCR progress. The image will be save as camera.jpg in the “files” folder	PASS	The camera will stay active until users click the “Esc” button, you can click “Space” button to take an image for multiple times, and the newest image will be used for OCR recognition.
Image pre-process module	Upload a skew image with a resolution below 2180p, check the resulting image file in the ‘files’ folder	The system will super-resolves and de-skews the uploaded image, and save the image as result.jpg in ‘files’ folder	PASS	If 3D skew correction is used, the background of the image other than the article will be removed.
Image-to-text module	Upload an image of an article with regularly	For the image of an article with regularly arranged text,	PASS	For some natural scenes with neatly

	arranged text and an image of an irregularly arranged text in a natural scene for recognition. Check the recognition result in the “Text results” interface.	the system will use the Tesseract engine for recognition. For the image of an irregularly arranged text in a natural scene, The system will automatically switch to the PGNet engine for recognition. The result will be displayed in the “Text results” interface, and will be save as result.txt in the “files” folder		arranged texts, the system will not automatically switch to PGNet for recognition if Tesseract produces an close-to accurate result.
Sentence connection function	Upload an image of an article for the system to recognise using the Tesseract engine, check the before and after result of text post-process.	The system will automatically connect the same sentence spread across different lines and the same word spread across different lines	PASS	None
Auto text correction function	Turn on spell correction and contextual correction respectively, check the text results.	The system will automatically correct the spelling mistake according to the chosen method.	PASS	None
Text-to-speech module	Upload an image for recognition, check the mp3 file in the “files” folder.	The system will convert the text to speech, generate an mp3 file and save it to the result.mp3 in the files folder.	PASS	None
Speech reading function	Upload an image for recognition, after the OCR progress finished, press “play” button to play the speech, “pause” to pause playing, “restart” to restart playing, and “stop” to end playing.	The speech will be played after users click the “play” button, pause if users click the “pause” button, restarted again in the same place if users click the “restart” button. Stop and clear the playing progress if users click the “stop” button	PASS	The the speech ends, users need to press “stop” and “play” again if he wants to replay the speech.

Table11: System functionality test

Through the test result, we can see that all the six major module works as designed and expected.

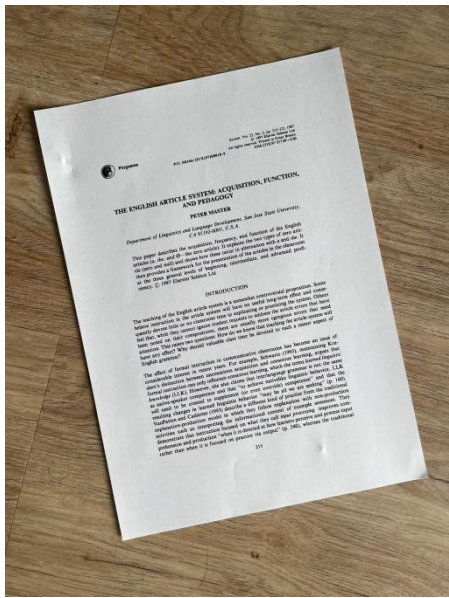
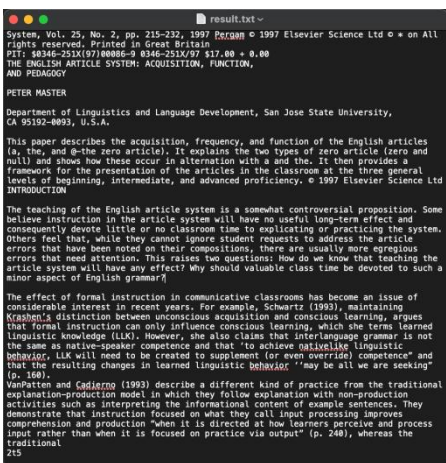
To enable the program to run properly and to make it easier to check the running result, the ‘files’ folder is created to store temporary files and result files generated during the running of the program. In the folder, camera.jpg is

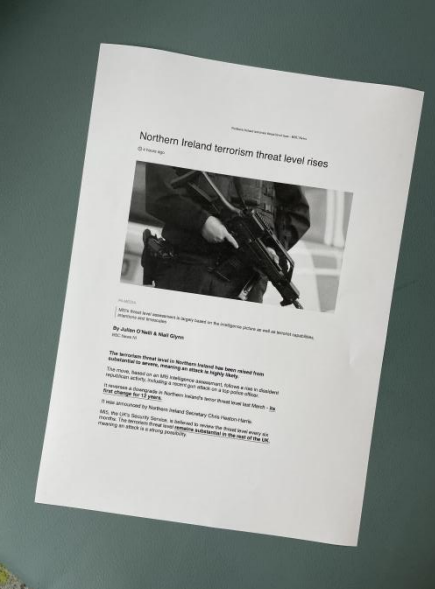


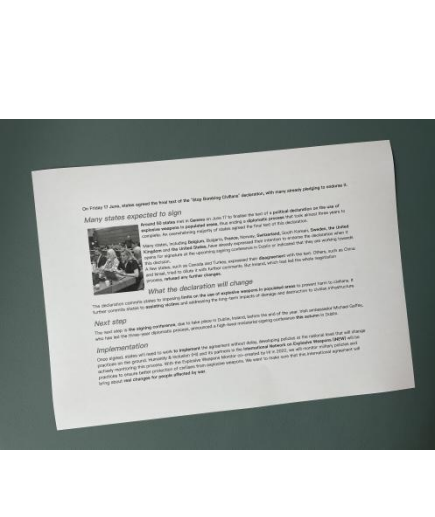




the image file that the user chose to capture with the camera, result.jpg is the image pre-processed result file, result.txt is the text recognition result produced after the text-to-speech module, PGNet-result.jpg is the default generated image file by PGNet engine (The result of the text recognition is displayed directly on the image), and result.mp3 is the audio file generate by the text-to-speech module.

4.2 System Evaluation

To evaluate the recognition accuracy (character Accuracy) of the system, I chose 3 article images and 9 natural scenes images (from Total-text), upload them to the system for recognition, and calculated the recognition accuracy. Generally, there are three ways to calculate the recognition accuracy: single word recognition rate, whole line recognition rate, whole sheet recognition rate. I choose the single word recognition rate to calculate my system's recognition accuracy, the formula and calculated result is as follow:

$$\text{Accuracy} = (\text{Number of correctly recognized characters} / \text{Total number of characters}) \times 100\%$$

Image	Recognition result	Accuracy
		403/407 = 99%

	<div>   result.txt </div> <p>Northern Ireland terrorism threat level rises - BBC News Northern Ireland terrorism threat level rises 4 hours ago</p> <p>MIS's threat level assessment is largely based on the intelligence picture as well as terrorist capabilities, intentions and timescales</p> <p>By Julian O'Neill & Niall Glynn BBC News NI</p> <p>The terrorism threat level in Northern Ireland has been raised from substantial to severe, meaning an attack is highly likely.</p> <p>The move, based on an MIS intelligence assessment, follows a rise in dissident republican activity, including a recent gun attack on a top police officer.</p> <p>It reverses a downgrade in Northern Ireland's terror threat level last March - its first change for 12 years.</p> <p>It was announced by Northern Ireland Secretary Chris Heaton-Harris.</p> <p>MIS, the UK's Security Service, is believed to review the threat level every six months. The terrorism threat level remains substantial in the rest of the UK, meaning an attack is a strong possibility.</p>	<p>154/155=99.3%</p>
	<div>   result.txt </div> <p>On Friday 17 June, states agreed the final text of the 'Stop Bombing Civilians' declaration, with many already pledging to endorse it.</p> <p>Many states expected to sign</p> <p>Around 50 states met in Geneva on June 17 to finalise the text of a political declaration on the use of explosive weapons in populated areas, thus ending a diplomatic process that took almost three years to complete. An overwhelming majority of states agreed the final text of this declaration.</p> <p>Many states, including Belgium, Bulgaria, France, Norway, Switzerland, South Korea, Sweden, the United Kingdom and the United States, have already expressed their intention to endorse the declaration when it ; opens for signature at the upcoming signing conference in Dublin or indicated that they are working towards this decision.</p> <p>q a A few states, such as Canada and Turkey, expressed their disagreement with the text. Others, such as China and Israel, tried to dilute it with further comments. But Ireland, which had led the whole negotiation f process, refused any further changes.</p> <p>What the declaration will change</p> <p>The declaration commits states to imposing limits on the use of explosive weapons in populated areas to prevent harm to civilians. It further commits states to assisting victims and addressing the long-term impacts of damage and destruction to civilian infrastructure.</p> <p>Next step</p> <p>The next step is the signing conference, due to take place in Dublin, Ireland, before the end of the year. Irish ambassador Michael Gaffey, who has led the three-year diplomatic process, announced a high-level ministerial signing conference this autumn in Dublin.</p> <p>Implementation</p> <p>Once signed, states will need to work to implement the agreement without delay, developing policies at the national level that will change practices on the ground. Humanity & Inclusion (HI) and its partners in the International Network on Explosive Weapons (INEW) will be actively monitoring this process. With the Explosive Weapons Monitor co-created by HI in 2022, we will monitor military policies and practices to ensure better protection of civilians from explosive weapons. We want to make sure that this international agreement will bring about real changes for people affected by war.</p>	<p>347/352=98.5%</p>
	<p>english heritage alan turing 1912 1954</p> <p>code breaker and pioneer of computer science was born here</p>	<p>14/14=100%</p>
	<p>Bikes direct</p>	<p>2/2=100%</p>

	<p>peak lookout the loowaut</p>	<p>$2/3=66\%$</p>
	<p>tea house wine tasting gastronomy</p>	<p>$5/5 = 100\%$</p>
	<p>selamat datang memori dtanjung asam pusat mini reaktas tel 016 4670126 pak teh</p>	<p>$11/13=84.6\%$</p>
	<p>east village shoe repaib or 1st marks customized sneakers buy sell trade new used shoes</p>	<p>$13/15=86.7\%$</p>




	india permit all tourist vehicle	4/5=80% ("all" is not in the correct position)
	constantinos market	2/2=100% (missing " ' ")
	hhatulistiwa cafe open 24hours	3/4=75%

Table12: System evaluation result

As we can see from the evaluation result, the accuracy of recognising images of articles with regular arranged texts is very high, most punctuation marks are correctly recognised and the same sentence separate on different lines is automatically connected. For images with irregularly arranged text in natural scenes, the overall recognition accuracy is relatively high, but there are still cases of incorrect recognition or incorrect texts order.

Overall, the system was able to correctly identify most of the test images, meeting the original design objectives.

5. Conclusions and Future work

5.1 Conclusions

This report design and implement an 7 stages image to speech system, which is image acquisition, image pre-process, image-to-text, text post-process, text-to-speech and speech playing. Each stage is described and justify in detail.

In the image acquisition stage, users can choose to upload an image locally or take an image using the camera on the computer. I compared the UI of many other text-to-speech software, and found their disadvantages of lacking user guidance in the image capturing progress. To deal with this problem, I have added real-time paper detection and skew detection function to my photo acquisition interface, this helps to improve the quality of the images taken by the user.

In the image pre-process stage, I discovered that many existing systems do not pre-process images, resulting in a reduction in the recognition accuracy of the OCR engine. To deal with this problem, I have made researches and analysis on different image super-resolution algorithms and skew correction algorithms, and choose an efficient super-resolution algorithm that triples the resolution of the original image, a 3D skew detection and correction algorithm, and an alternate 2D skew detection and correction algorithm for the image pre-process progress. In addition, I binarised the image to reduce the interference of stray colors.

In the image-to-text stage, I have researched and analyse on different OCR engines, and choose Google Tesseract and PGNet as my two OCR engine. They are responsible for the processing of long article images with regularly arranged text and natural scenes images with irregularly arranged text. At the same time, I designed an NLP-based auto-switching algorithm that allows the system to automatically select the most suitable OCR engine for recognition based on the image.

In the text post process stage, I have discovered the problem that Tesseract will automatically line wrapping the recognition results, which will reduce fluency in text-to-speech results. To deal with this issue, I designed an algorithm to rejoin same sentences separate by Tesseract in different lines. At the same time, in order to solve the problem of misrecognition of similar-looking letters in the image-to-text process, I researched applied two different word spelling correction algorithms to improve the accuracy of OCR recognition results. Users can choose whether or not to enable these two automatic spelling correction algorithms.

In the text-to-speech stage, I compared and analysed different TTS engines and finally chose the Google gTTS engine that best suited my system. And for Speech playing stage, I used Pygame's music playing module instead of the default playing function provided by gTTS to implement more complex speech playing functions.

Overall, my system solves many of the shortcomings of existing software. Compared to the them, my system has full user guidance on the image taking interface, better image pre-process function, the ability to recognise irregularly arranged text in natural scenes, and better text post-processing processes. These improvements can deliver a better user experience. My system can not only help visually impaired people to read, but also help to scan traditional books and turn them into e-books, or extracting useful text information from images of natural scenes, etc. However, due to various objective factors and practical constraints, the process of text extraction from the image is still not guaranteed to be 100% accurate, the OCR engines and text extraction models still needs to be improved.

5.2 Future work

Although this report implements a complete, easy to use, and accurate text to speech system, we still need to make improvements in the following parts:

In terms of the user interface, although the design of the user interface has been optimised for people with visual impairments, all buttons have been designed to be particularly large and easy to distinguish, it is still possible that people with severe visual impairments are unable to identify different buttons on their own in order to use the system. In future development I will make the system able to automatically read the font on the button when the mouse is moved over the surface of the button. This is usually achieved with the 'hover' function. However, Pysimple GUI didn't provide this function, in the future I may need to refactor the user interface design with a different library.

In terms of the image-to-text function, for Google Tesseract, in future developments I can use jTessBoxEditor to train and improve the Tesseract model, I can add more training data of similar scenarios for the user's usage scenarios, which can help to improve the text recognition accuracy. For PGNet, the order of text recognition in a small number of natural scenes still needs to be optimised. For example, if the text is arranged in a circular pattern, then the reading order is not necessarily from top to bottom. For the auto switch algorithm between PGNet and Tesseract, even if this algorithm does the switch correctly in most cases, I still need to continue to improve it for some specific scenarios. For example, if Tesseract identified only 2 out of 20 words, although both 2 words were all correct, the system should still switch to PGNet for recognition at this point.

In terms of the text post-process function, at the moment I only use paragraph length to distinguish between the title and the body of the article. (Even if the title does not end with a punctuation mark, it will not be automatically linked to the previous or next sentence). Although the correct distinction can be made in most cases, if the title appears to be too long, and the previous sentence does not end with a punctuation mark, the algorithm still judges it as a sentence and joins it to the previous or next sentence. This can lead to problems with reading headlines without pauses. In future development, I will continue to work on algorithms that can correctly distinguish between headlines and body text (maybe by NLP judgement or deep learning based algorithms).

6. Reflection

In this project, I have designed and implemented a fully function 7 stage text to speech system. To be honest, this project was very difficult for me at the beginning, since I have not learned anything about image processing and computer vision. Under the guidance of my supervisor, I start to learn about the steps of image-to-text and text-to-speech. Through research and learning on articles and books, I was able to design the architecture of my system. I was trying to do a six step image to speech system as my first version, which is image acquisition, image pre-process, image-to-text, text-to-speech, and speech playing. But as this is only a very basic and simple system, my supervisor reminded me to research and compare on other existing systems, find their shortcomings and make improvements. So I began to dig deeper into each step and trying to make innovations and improvements.

In the implementation of image acquisition module, I started out with just the basic photo upload and local upload functions. After delving into other systems, I added a user guidance function to make the interface more user-friendly. In the implementation of image pre-process module, I only did 2D skew correction, greyscaling and binarization to the images at the beginning. But with my supervisor's prompting, I discovered that images may also be skewed in the 3D plane, and that images with too low a resolution may reduce the accuracy of recognition. So I have designed a skew correction algorithm with 3D skew correction as the primary and 2D skew correction as a backup, as well as adding a image super-resolution algorithm to process the input images. In the implementation of image-to-text module, I started my work with finding OCR engines that could recognize regularly arranged text and located Google Tesseract as a very efficient and accurate open-source engine. But after I dig deeper in the image-to-text field, I found out that Tesseract is not performing

well when deal with images in natural scenes with irregularly arranged text, So I started working on end-to-end text recognition methods and discover PGNet as a suitable OCR engine for my system. At this point, my supervisor reminded me of the problem of text recognition order in natural scenes and the problem of switching between two text recognition engines. To deal with these problems, I wrote the automatic switching algorithm and the text sorting algorithm for natural scenes. In the process, I also discovered the problems in the output result of Tesseract, and added one more steps (text post-process) to my system to improve the recognition result.

During the implementation of this system, I found that the iterative development approach well suites me for developing small software. I first start by developing the most basic but functioning system as my first version, then I can iterate and optimise each part of the system individually based on requirements and different scenarios. But in the actual progress, I also found that I had not considered all aspects of the issue well enough. For example, I did not take into account the problem of images skewing in 3D, and the irregularly arranged text extraction problem in the early stage. Thanks for the guidance of my supervisor, I was able to identify these problems and came up with solutions. In the process of developing software in the future, I must take into account the complexity of the environment in which the software is used, and design software that can flexibly respond to various usage scenarios.

During the research process, I have also experienced the difficulties of living with a visual impairment. In my future life, as a citizen, I will definitely give them as much help as I can. And as a computer science student graduate from Cardiff University, I will definitely use my knowledge to contribute to the development of accessible software, to help more people with disabilities in need.

Reference

- [1] World Health Organization. (2022, October 13). Blindness and Vision Impairment. Who.int; World Health Organization: WHO. Available at: <https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment>
- [2] Watson, A. (2022, March 23). Newspaper consumption frequency U.S. by age 2021. Statista. Available at: <https://www.statista.com/statistics/1251242/newspaper-usage-frequency-by-age/>
- [3] Sonth, Shalini, and Jagadish S. Kallimani. "OCR Based Facilitator for the Visually Challenged." IEEE Xplore, 1 Dec. 2017, available at: <https://ieeexplore.ieee.org/document/8284628>
- [4] Chandra S, Sisodia S, Gupta P. Optical character recognition-A review[J]. International Research Journal of Engineering and Technology (IRJET), 2020, 7(4): 3037-3041.
- [5] Liu, Yuliang, et al. "ABCNet: Real-Time Scene Text Spotting with Adaptive Bezier-Curve Network." Openaccess.thecvf.com, 2020. Available at: openaccess.thecvf.com/content_CVPR_2020/html/Liu_ABCNet_Real-Time_Scene_Text_Spotting_With_Adaptive_Bezier-Curve_Network_CVPR_2020_paper.html
- [6] Wang, Pengfei, et al. "PGNet: Real-Time Arbitrarily-Shaped Text Spotting with Point Gathering Network." Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, no. 4, 18 May 2021, pp. 2782–2790. Available at: ojs.aaai.org/index.php/AAAI/article/view/16383
- [7] Nielsen, J., and Molich, R. (1990). Heuristic evaluation of user interfaces, Proc. ACM CHI'90 Conf. (Seattle, WA, 1-5 April), 249-256.
- [8] Python GUIs for Humans. Available at: <https://www.pysimplegui.org/en/latest/>
- [9] OCR text scanning, Contour detection, Perspective transformation. Available at: https://blog.csdn.net/weixin_47924038/article/details/117363977?
- [10] Schwarz, S. (2020, July 8). Correcting Image Rotation with Hough Transform. Sinch Blog. Available at:

<https://medium.com/wearesinch/correcting-image-rotation-with-hough-transform-e902a22ad988>

[11] Rehman, Amjad, and Tanzila Saba. "DOCUMENT SKEW ESTIMATION and CORRECTION: ANALYSIS of TECHNIQUES, COMMON PROBLEMS and POSSIBLE SOLUTIONS." *Applied Artificial Intelligence*, vol. 25, no. 9, Oct. 2011, pp. 769–787, Available at: <https://doi.org/10.1080/08839514.2011.607009>

[12] Tesseract: <https://github.com/tesseract-ocr/tesseract>

[13] Agarwal, Vardan, and Lipi Patnaik. Super Resolution in OpenCV. 22 Mar. 2021. Available at: learnopencv.com/super-resolution-in-opencv/

[14] ContextualSpellCheck: <https://github.com/R1j1t/contextualSpellCheck>

[15] Sreekanth. "Improve Tesseract OCR Accuracy with Spellchecking." *Medium*, 19 Dec. 2021, Available at : medium.com/@rr_42830/improve-tesseract-ocr-accuracy-with-spellchecking-d69806fb32e

[16] TextBlob: <https://github.com/sloria/TextBlob>

[17] W. Bieniecki, S. Grabowski and W. Rozenberg, "Image Preprocessing for Improving OCR Accuracy," 2007 International Conference on Perspective Technologies and Methods in MEMS Design, Lviv, UKraine, 2007, pp. 75-80, doi: 10.1109/MEMSTECH.2007.4283429. Available at : <https://ieeexplore.ieee.org/abstract/document/4283429>

[18] M. Brisinello, R. Grbić, M. Pul and T. Anđelić, "Improving optical character recognition performance for low quality images," 2017 International Symposium ELMAR, Zadar, Croatia, 2017, pp. 167-171, doi: 10.23919/ELMAR.2017.8124460. Available at : <https://ieeexplore.ieee.org/abstract/document/8124460>

[19] S. Qin, A. Bissaco, M. Raptis, Y. Fujii and Y. Xiao, "Towards Unconstrained End-to-End Text Spotting," in 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Korea (South), 2019 pp. 4703-4713. doi: 10.1109/ICCV.2019.00480