Paper To Speech

ENEE408G Final Report

Cherif Aidara, John Baur, Zeting Luan

5/13/2021

System Design

Paper to Speech is an accessibility tool that uses a standard webcam to read text from a document. Design of our system required many topics related to multimedia signal processing. The first task is to analyze the webcam feed to take a picture when a document is detected. The next task is to detect and crop the characters on the page so that they can be converted to text through an Optical Character Recognition (OCR) system. The OCR system processes images letter by letter and requires the letters and spaces to be properly cropped into multiple successive images. Following the cropping of each letter from the paper, the OCR algorithm will process letter by letter, and write the output to a text file as well as the command line for the user to see the current output.

Motivation and Market Survey:

Our motivation to create the Paper to Speech system for our capstone project was to help people with visual impairments. In recent years, many documents that have traditionally been printed on paper have become digitized. Despite the amount of paper in our lives steadily decreasing, there are still many important documents that are distributed physically. While there are many features on computers and smartphones that increase accessibility for viewing digital documents, we found only a few solutions that increased accessibility for physical documents. These solutions were prohibitive because they were very expensive, costing thousands of dollars. One of the machines that we found costs $2,195 [

People with visual impairments struggle with any task that involves reading. Many blind people currently hire people to read for them [1]. With our system, they would gain independence in their lives by being able to view important documents by themselves.

We did not attempt a market survey, but some of us on the team have a personal connection to someone who has visual impairments. We think that if we can lower the cost of our system, we would be more competitive than the other products on the market and there would be demand for it. Additionally, we anticipate the need for our system to go up as the amount of blind people will increase over time. It is estimated that from 2015 to 2050, the amount of people who are legally blind will double, resulting in a total of 2 million people [2].

Detailed Design:

We initially planned to divide our project into three modules. These modules would be Word Detection and Cropping, Optical Character Recognition, and Text to Speech. As the project went on it was clear that we would need to add two more modules. The first was a Graphical User Interface to assist the user in running the program. The second was creating a program to detect when documents were placed under the webcam. The following sections discuss our design of each module of our project.

I. User Interface

The graphical user interface (GUI) was designed to make the Paper to Speech system user friendly. The GUI was made using the Matlab App Designer. The code behind the GUI is fairly simple. The "Scan Page" button runs a Matlab script which runs the scanning program. The "Replay Previous Recording" button runs a python script which generates the mp3 file again and plays it over the computer's speakers. Selecting an accent from the drop down menu triggers Matlab to write to a text file on the computer. This text file is read by the Python script for generating the text to speech recording. Additionally, because our system is made for users who have vision problems, we added voice recordings that will both assist in aligning the page under the webcam and output the current program state.

II. Capturing the Webcam Image

When the Scan Page button is pressed in the GUI, Matlab runs the Python script, video.py which does a variety of functions. First, using functionality provided by the OpenCV library, this script connects to the computer's webcam and displays the video feed. Then, it crops the webcam image to only be the scanning region on the desk. For this cropped image, the script performs page detection and motion detection on the image. The script runs in a loop taking webcam captures every 0.2 seconds until a page is detected and it is determined that there is no motion in the frame. Then, the python script saves this image to the computer and exits.

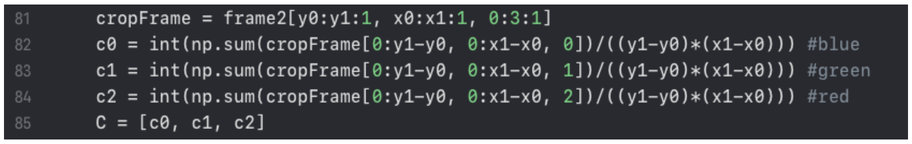
While this program is running, it continuously outputs the results of the page and motion detection algorithms on both the screen and through voice recordings. The results are outputted on the screen with the text boxes "Steady", "Not Steady", and "Page Detected". These are added to the video feed window by changing the pixel values on each frame to equal a premade image. Similarly, the program uses the playsound library to play mp3 files for the voice outputs that were made beforehand with the Google text to speech library.

In this part of the project, we needed to design both a method for page detection and for motion detection. If just page detection was used, then the program would choose the image when it first detects the page. The output image would likely be blurred as the paper would still be in motion.

There needed to be a motion detection algorithm to allow the page to be placed and ensure that the users hands are not still in the frame.

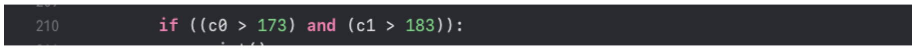
Page Detection

Page detection was accomplished by taking the average value of each of the RGB components over the cropped scanning region:



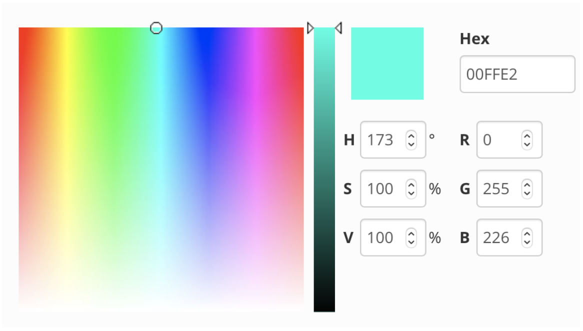
```
    cropFrame = frame2[y0:y1:1, x0:x1:1, 0:3:1]
    c0 = int(np.sum(cropFrame[0:y1-y0, 0:x1-x0, 0])/((y1-y0)*(x1-x0))) #blue
    c1 = int(np.sum(cropFrame[0:y1-y0, 0:x1-x0, 1])/((y1-y0)*(x1-x0))) #green
    c2 = int(np.sum(cropFrame[0:y1-y0, 0:x1-x0, 2])/((y1-y0)*(x1-x0))) #red
    C = [c0, c1, c2]
```

I found that the red components did not change much when paper was detected and used the blue and green components of the image to determine how much white was present in the image. I heuristically determined the cutoffs by experimenting with cut offs values with the webcam.



```
        if ((c0 > 173) and (c1 > 183)):
```

For the system used in the presentation, I determined that a page was present if the average of the region for blue pixels was over 173, and 183 for green pixels. I found this to work well on my desk. However, I realized that there are colors that are not white



Hex 00FFE2

H 173 °

S 100 %

V 100 %

R 0

G 255

B 226

Image Source: https://alloyui.com/examples/color-picker/hsv.html

which would trigger the page detection. For example, the light blue above is on the opposite side of the RGB spectrum as white but it still triggers my page detection. This system would not work well if used on a desk of that color. For a more robust system, it would be better to use the set of all RGB values that fall below the black line in the following image:

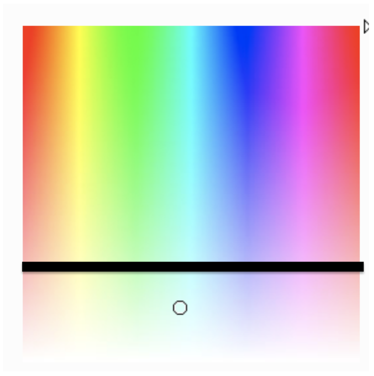
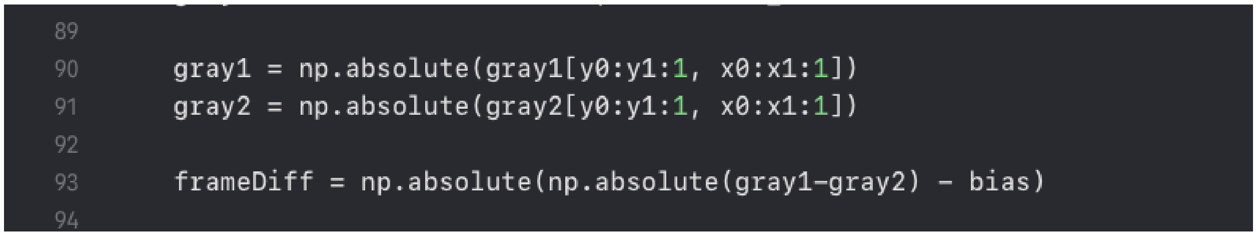


Image Source: https://alloyui.com/examples/color-picker/hsv.html

This would guarantee that the color is close to white and could be achieved easily by requiring that red pixel values to be above 180 for a page to be detected.

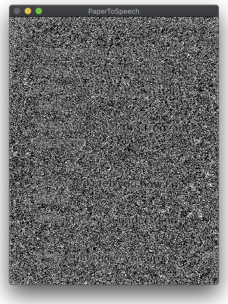
<u>Motion Detection</u>

Motion detection was accomplished by subtracting two successive frames and then summing over all of the pixels of the absolute value of the resulting image. If there was no motion, the images would be similar, and this number would be low. If there was motion in the frame, the two images would be different, and this number would be much higher.
The following code crops both frames, then subtracts them and the bias term.



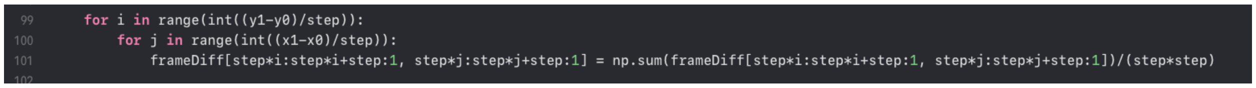
```

    gray1 = np.absolute(gray1[y0:y1:1, x0:x1:1])
    gray2 = np.absolute(gray2[y0:y1:1, x0:x1:1])

    frameDiff = np.absolute(np.absolute(gray1-gray2) - bias)

```

Simply subtracting two successive frames was not very effective at detecting motion since there was a lot of noise present due to the webcam's image being grainy. The following image is the image formed from two successive frames with no motion in between.



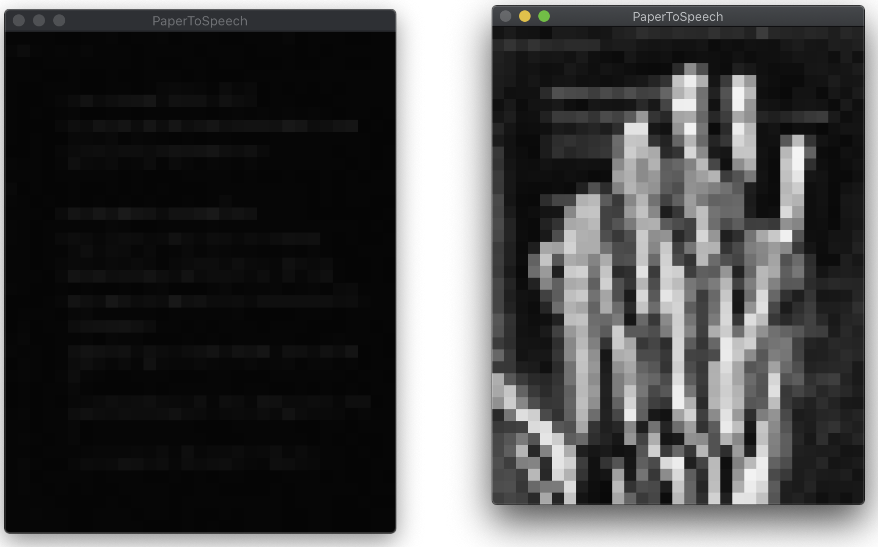
The motion signal calculated by summing over all the pixels of the above image would increase when motion occurred, but the noise dominated the signal. The noise signal was around 4 times larger than the signal generated from differences between two frames. To reduce the noise in the image, I divided the image into regions of 25x25 pixels and replaced each pixel in each region with the average of the region. This method is similar to applying a low pass filter for an image, except a convolution was not used. The result was similar in that the high frequency noise was removed.

The following code performs the averaging and replacement of each region:



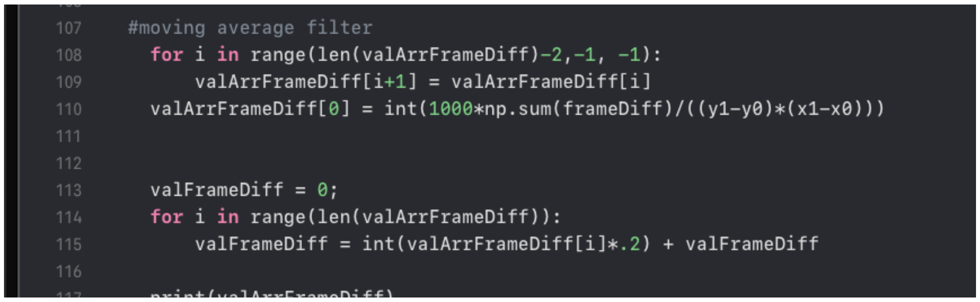
```
    for i in range(int((y1-y0)/step)):
        for j in range(int((x1-x0)/step)):
            frameDiff[step*i:step*i+step:1, step*j:step*j+step:1] = np.sum(frameDiff[step*i:step*i+step:1, step*j:step*j+step:1])/(step*step)
```

After averaging out each region, I subtracted a bias term that was approximately equal to the average noise. The resulting image had much less noise and the signal generated from motion was now up to 10x higher than the noise. This made using a simple threshold value to detect motion much more robust. The following two images show the webcam without motion and with a hand waving in front of the camera. As is seen by the first image, the noise is drastically reduced with the averaging method.



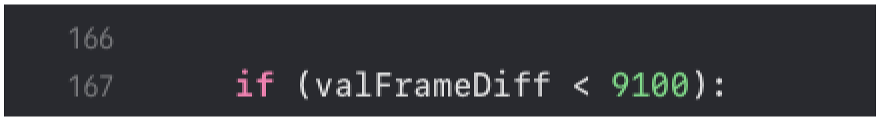
Next, I passed the motion signal through a moving average filter. The rationale behind this was that the time between two frames was only 0.2 seconds. I felt that motion not occurring for 0.2 seconds was inadequate to determine if it was actually present. If someone is placing the paper in the scanning region, their hands may stop for a short period of time, but this would not be an optimal time to take a picture of the paper. Therefore, I used a moving average filter with a period of 5 so that motion would have to stop for a full second before the webcam would take a picture.

I implemented the moving average filter by creating an array in Python of length 5. The first for loop shifts each value in the array to the right. Then, the current signal is added into the first index of the array. The multiplication by 1000 is not necessary but it makes it easier to set a threshold with a precise number that is an integer. In the second for loop, I sum over the entire array and divide by 5 to get the output of the moving average filter.



```
  #moving average filter
    for i in range(len(valArrFrameDiff)-2,-1, -1):
        valArrFrameDiff[i+1] = valArrFrameDiff[i]
    valArrFrameDiff[0] = int(1000*np.sum(frameDiff)/((y1-y0)*(x1-x0)))


    valFrameDiff = 0;
    for i in range(len(valArrFrameDiff)):
        valFrameDiff = int(valArrFrameDiff[i]*.2) + valFrameDiff

```

The variable valFrameDiff is the output of the moving average filter. I found that when it went under 9100, that this threshold was representative of no motion occuring in the scanning region.



```
        if (valFrameDiff < 9100):
```

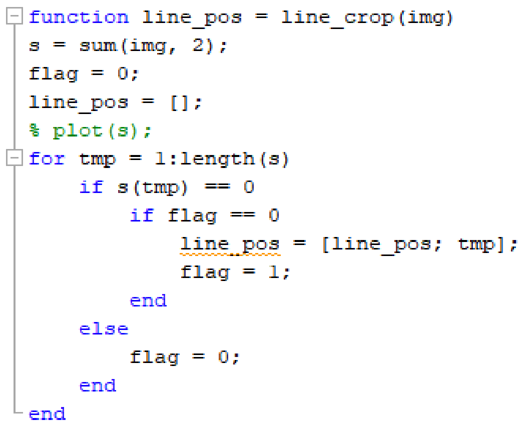
The thresholds for motion and page detections would determine when to capture an image. Additional if statements using these threshold values were added to cause the images to be overlaid on the video feed and play the voice recordings. One issue encountered was that when a video recording was playing, the execution of the Python script stopped. This made the program feel glitchy but was ultimately not super noticeable. In future versions, fixing this issue and having the sound recordings play concurrently with the video feed would make the program feel smoother.

III. Characters cropping

After getting the grayscale image, we converted the grayscale image to a binary image. The noise reduction helps a lot with the accuracy of the conversion between grayscale and binary. We utilized the binning method in image processing to realize character cropping. Binning is the procedure of combining a cluster of pixels into a single pixel.

Line cropping

We first wanted to separate the lines from the page. Since the black pixel is zero and the white pixel is one in the binary image, we then inverted the binary image. So, only the pixels that value are one is the pixel that is part of a letter. We added the pixel values of each row and stored them into arrays (I_1 (i) = P (i,1) +P (i,2) +...+P (i,n)).

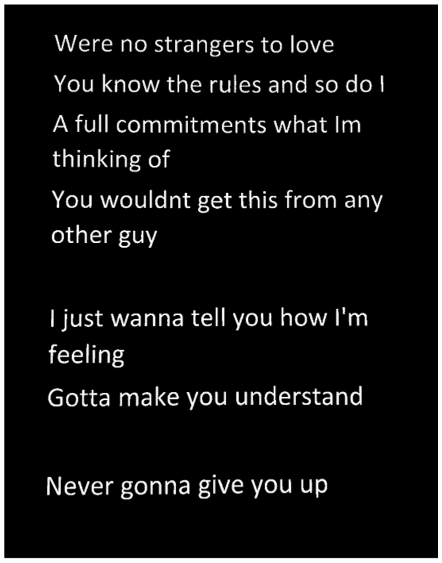


```
function line_pos = line_crop(img)
s = sum(img, 2);
flag = 0;
line_pos = [];
% plot(s);
for tmp = 1:length(s)
    if s(tmp) == 0
        if flag == 0
            line_pos = [line_pos; tmp];
            flag = 1;
        end
    else
        flag = 0;
    end
end
```

Then, we got the horizontal profile of this image. If we plotted the horizontal profile, we would get numbers of spikes, representing the number of lines of characters on a whole page. We also got many zero values in this array, and it showed the white space between each line. Once we have the boundaries between lines and spaces, we can crop the lines using the imcrop function in Matlab.

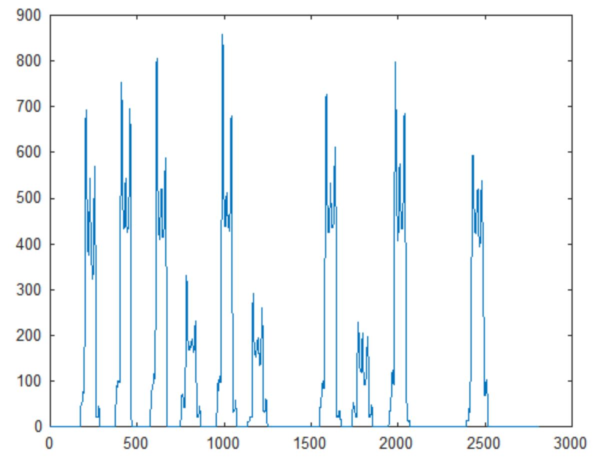
```
function img_out = clip(img_in)


if size(img_in, 3)==3

    img_gray = rgb2gray(img_in);

    threshold = graythresh(img_gray);
    img_bw = ~imbinarize(img_gray, threshold);
else
    img_bw = img_in;
end
[r, c] = find(img_bw);
pos = [min(c), min(r), max(c)-min(c), max(r)-min(r)];
img_out = imcrop(img_in, pos);
```

Thus we would get several horizontal rectangle images, and each of them represented a line of characters.


Were no strangers to love
You know the rules and so do I
A full commitments what Im
thinking of
You wouldnt get this from any
other guy

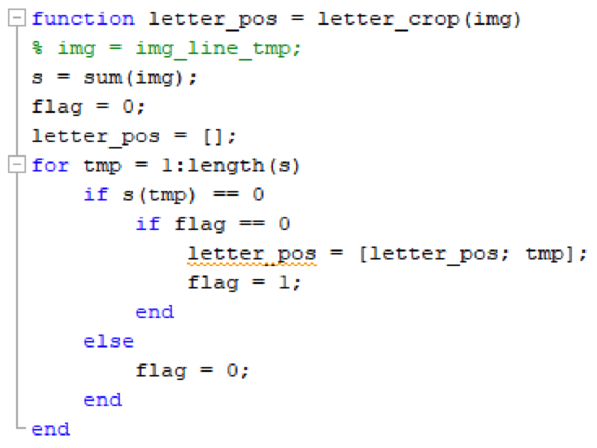
I just wanna tell you how I'm
feeling
Gotta make you understand

Never gonna give you up



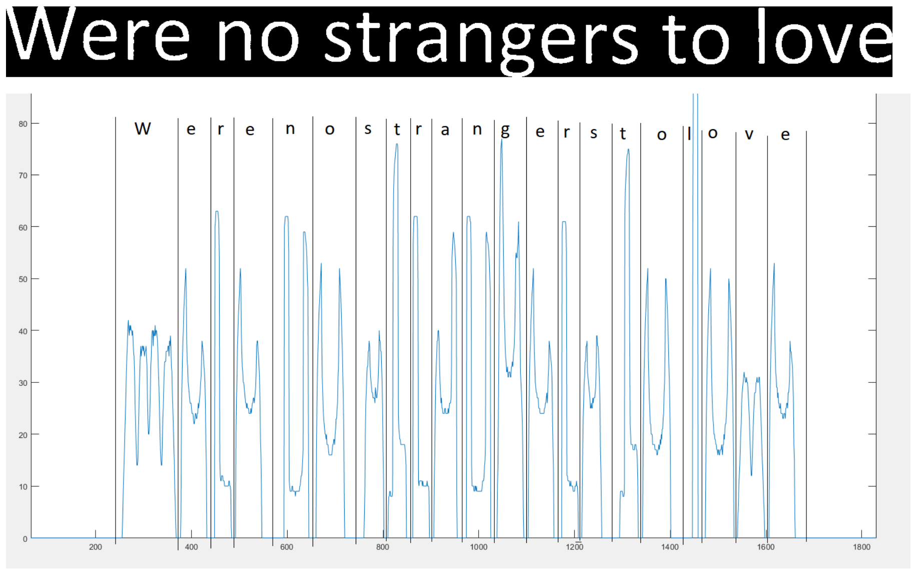
Letter cropping

After we had a line of characters, we were able to crop the letters. We used the same method as line cropping. We added the pixel values of each column and stored them into different arrays (I_2 (i) = P (1,i) +P (2,i) +...+P (n,i)).



```
function letter_pos = letter_crop(img)
% img = img_line_tmp;
s = sum(img);
flag = 0;
letter_pos = [];
for tmp = 1:length(s)
    if s(tmp) == 0
        if flag == 0
            letter_pos = [letter_pos; tmp];
            flag = 1;
        end
    else
        flag = 0;
    end
end
```

Thus, we got the vertical profile of the line image. We also got numbers of spikes on the vertical profile plot, which showed the number of the word in a specific line. The small gap between each letter is the location where we split them into different images. The small white spaces between each letter are several zero values in this array. Therefore we can crop the letters using the imcrop function in Matlab since we have determined the boundaries.



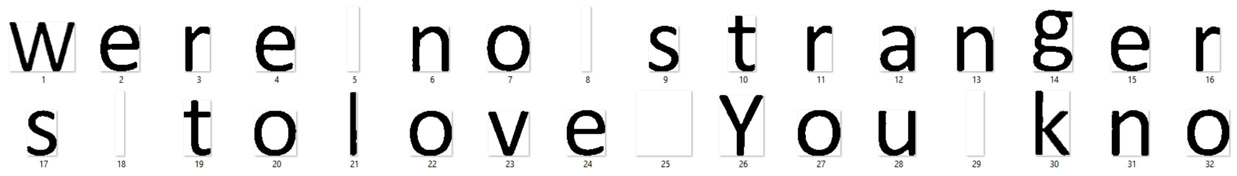
Were no strangers to love

<u>Space detection</u>

In order to have a better audio output, the space between each letter is preferable. Therefore the next step is space detection. The white space between words is larger than the space between each letter. Thus, we can determine the threshold to redefine the boundaries between two letters. Then, we can crop the space between two letters.
The output for this algorithm is multiple image files, and then we can do the training for character recognition.

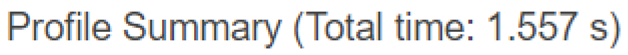
<u>Limitations</u>

There are some limitations to this algorithm. For example, the letters cannot be cropped properly if the font size is too small. This is because the webcam camera cannot perfectly scan the paper, which causes some words to be connected to each other in the binary image. In this case, since there is no space between these words, it cannot crop the words perfectly. We could have solved this problem by setting a threshold, but due to the randomness of the word arrangement and the degree of the connection, it might not function properly with a different setup. Another limitation is the paper must be perfectly placed on the desk. Since we used the binning method to achieve character cropping, the lines must be close to perfectly horizontal. We solved this by setting up three pins to guide the paper to be perfectly placed. However, if the paper is rotated a little, the cropping might not correctly function.



<u>Results and summary</u>

The runtime for around a hundred words page is less than two seconds, which is ideal compared to other methods we have tried before.

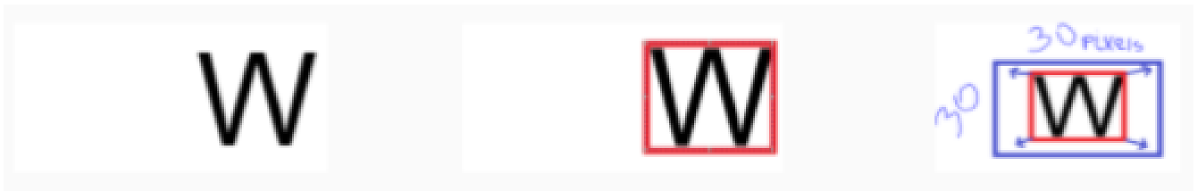

Profile Summary (Total time: 1.557 s)

The quality of the output image files is excellent; there was little to no noise and distortion in the image files, which helps a lot with word recognition accuracy.

IV. Optical Character Recognition

After getting the cropped images, we can apply optical character recognition (OCR) using Bayes classifier. To run the classifier, we first have to generate the training images. For lack of databases we had to generate our own. Using google docs, we typed in the entire alphabet in the following 12 fonts: arial, times new roman, verdana, calibri, bree serif, conforta, courier new, cambria, georgia, Roboto, Merriweather and Permanent Marker. The classification is based on images of individual letters. Using a snip tool, each image is cropped by hand. This poses a problem because the image is not centered within the frame in the snip tool, and each image will have a different dimension. Similarly, the individual letters will have different dimensions depending on the font of the letters on the paper, the focus of the lens, and the proximity of the paper to the camera. The training images and testing images must be standardized to one dimension prior to the classification.

The training images are first converted to 8 bits and a threshold of 200 is set to binarize the image data. Using the binary image, we cut out the letter as precisely as possible using the upper, lower, left most, and rightmost 0 pixel intensity. This process is done for each image in the folder containing all the cropped letters. Once the image is formed, it is zero padded to form a 30 by 30 image depending on its dimensions.



The final image is stored into a 4D matrix. The first 2 dimensions form the pixel intensities of the image, the third dimension goes from 1 to 52 and describes the letter and if its upper or lower case. The 4th dimension goes from 1 to 12 and describes the font of the letter. The 4D matrix is processed to form the training and testing data. For each letter, the 12th image is used as testing, and the remaining 11 is used for training. The fonts used for training are never the one used for testing.

$$g_i(\mathbf{x}) = \mathbf{x}^t \mathbf{W}_i \mathbf{x} + \mathbf{w}_i^t \mathbf{x} + w_{i0},$$

$$\mathbf{W}_i = -\frac{1}{2}\boldsymbol{\Sigma}_i^{-1},$$

$$\mathbf{w}_i = \boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i$$

$$w_{i0} = -\frac{1}{2}\boldsymbol{\mu}_i^t \boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i - \frac{1}{2}\ln|\boldsymbol{\Sigma}_i| + \ln P(\omega_i).$$

Bayes classifier is a linear classifier that uses $g_i(x)$ to classify the data. The equation $g_i(x)$ basically describes a regular linear equation (y=mx+b) with the slope being (W_i+w_i) and the y intercept being w_{io}. Before the classification, we apply Principal Component Analysis (PCA) to reduce the dimensionality of the training data, and speed up the classification process. PCA requires the data to be standardized. All of the training data is compiled together and the mean is calculated. The centralized data becomes the training data minus the mean. Using the centralized data, the eigenvectors are extracted using singular value decomposition. The eigenvectors give us a ranking of the importance of the features of the training data. We don't have to specify the number of features to keep and which to discard. SVD selects a projection that maximizes the variance of the output. The least significant features are discarded to reduce the dimensions. To generate the new training data, we multiply the eigenvector with the centralized data. The same process is applied to the testing data with the eigenvectors of the training data.

The classification starts using the new training and testing data. The 12 fonts for each letter are summed together and divided by 12 to get the mean. Then we calculate the covariance matrix for each letter with the following script

```
for i=1:classes
   for j=1:no_training_class
      sigma(:,:,i) = sigma(:,:,i) +  ( training(:, no_training_class*(i-1)+j) - mu(:,i) ) * ( training(:, no_training_class*(i-1)+j) - mu(:,i) ).'
   end
   sigma(:,:,i) = sigma(:,:,i)/no_training_class;
   sigma(:, :, i) = sigma(:, :, i) + eye(picsize);
   sigma_inv(:, :, i) = inv(sigma(:, : ,i));
end
```

Once the covariance matrix is calculated, we simply take its transpose as described in figure_. w_{io} is calculated by multiplying W_i with the mean from each letter.

```
w(:, i) = sigma_inv(:, :, i) * mu(:, i);
w0(i) = -1/2 * mu(:, i)' * sigma_inv(:, :, i) * mu(:, i) - 1/2 * log(det(sigma(:,:,i)))
```

Now we have all the components we need for the classification. $g_i(x)$ gives probabilistic estimates of the testing data belonging to a class. For each testing data, we compute $g_i(x)$ for each class and take the maximum as the most likely class for the testing data.
Each testing input is classified as a number from 1 to 52. The classification tells us the predicted letter. The table below shows how each predicted output is classified as a letter.

Classification	Upper case	Classification	Lower case
1	A	a	27
2	B	b	28
3	C	c	29
4	D	d	30
5	E	e	31
6	F	f	32
7	G	g	33
8	H	h	34
9	I	i	35
10	J	j	36
11	K	k	37
12	L	l	38
13	M	m	39
14	N	n	40
15	O	o	41
16	P	p	42
17	Q	q	43
18	R	r	44
19	S	s	45
20	T	t	46
21	U	u	47
22	V	v	48
23	W	w	49
24	X	x	50
25	Y	y	51
26	Z	z	52

A simple conversion algorithm takes the output out the classifier and writes the predicted letter to the command line and to a txt file.

OCR detection speed: 4.14s

OCR Improvements

Accuracy Optimization

For the PCA, we started with 5 fonts per letter with an accuracy percentage of 62%. To improve the accuracy, we did 2 things: first increase the number of training samples, and second apply some type of dimension reduction algorithm.

The first idea we had was to reuse the training samples. The testing data also came from the training data. This gave us an accuracy of 100%, because we were testing with data that the computer already knew (overfitting). Using new testing data dropped the accuracy to 48%.

To increase the accuracy, we needed more training data that was as differentiable as possible. So we increased the number of fonts to 12. This resulted in a 90% accuracy rate, meaning there were still 5 letters that were still being misclassified.

Using the PCA algorithm described above, the accuracy increased to 96% meaning that there were 2 letters that were being misclassified.

I-L detection problem

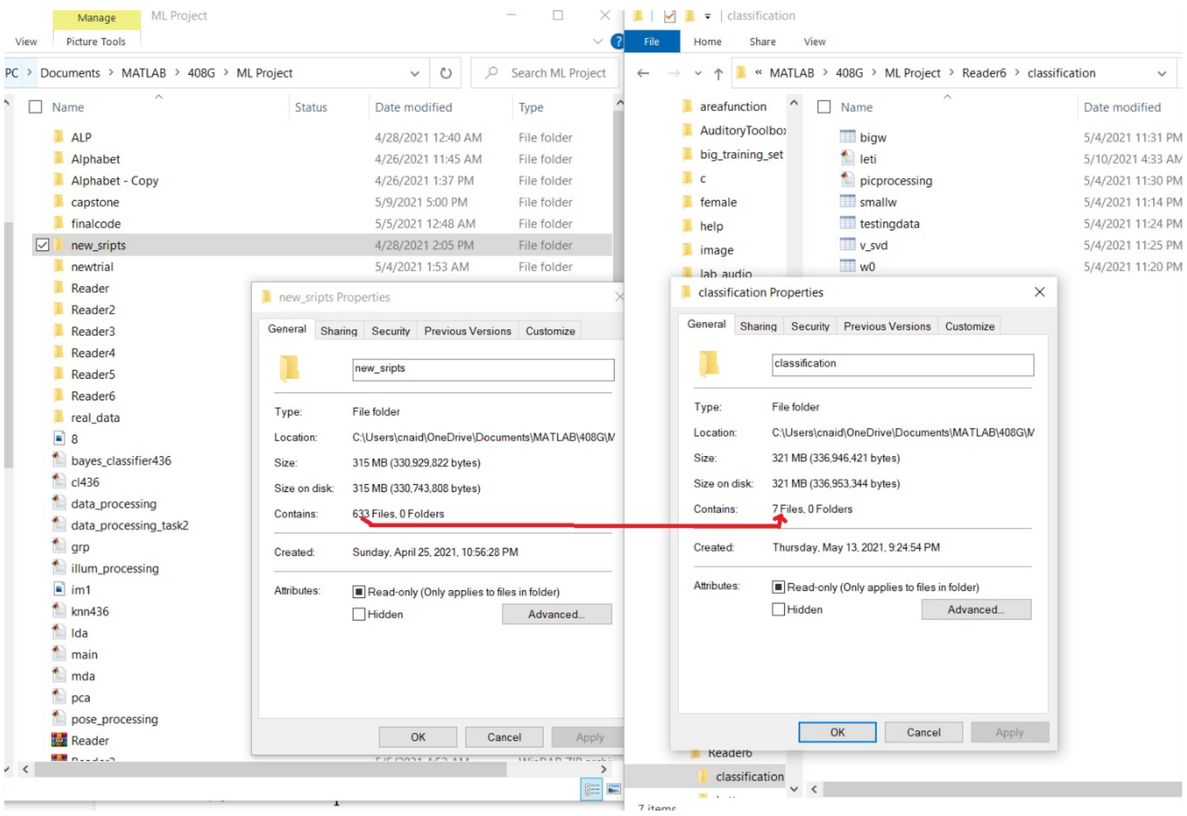
After looking at the misclassified letters and the images used for their training, we realized that some training images for upper case I and lower case L looked almost the same. There is no way to classify the 2 letters if they look exactly the same. We made the decision to classify letters that look like as lower case L, and the upper case I is classified with the following letters.

leti1
leti2
leti3
leti4
leti5
leti6
leti7
leti8
leti9
leti10
leti11
leti12

After running the classifier with the improvements discussed, the accuracy percentage went to 100%, and all the testing data were different from the training data.

Speed and Space improvements

Classifying each letter at a rate of 4 seconds takes a while for large images. To speed up the process, we decided to reduce the amount of computing from the classifier. The eigenvectors produced by the SVD are computed from the training data. That means for each run we are reusing the same training data to recompute the same eigenvectors for the PCA, and the same classification equations ($g_i(x)$). We decided to simply download the eigenvectors and the classification equations and just call them in Matlab when we have to do a classification. However, since the classification equations are calculated from the output of the PCA, we always have to recompute the dimensions for the testing data. But that is a simple multiplication.



```
PC > Documents > MATLAB > 408G > ML Project

Name                    Status   Date modified        Type
ALP                              4/28/2021 12:40 AM   File folder
Alphabet                         4/26/2021 11:45 AM   File folder
Alphabet - Copy                  4/26/2021 1:37 PM    File folder
capstone                         5/9/2021 5:00 PM     File folder
finalcode                        5/5/2021 12:48 AM    File folder
new_sripts                       4/28/2021 2:05 PM    File folder
newtrial                         5/4/2021 1:53 AM     File folder
Reader
Reader2
Reader3
Reader4
Reader5
Reader6
real_data
8
bayes_classifier436
cl436
data_processing
data_processing_task2
grp
illum_processing
im1
knn436
lda
main
mda
pca
pose_processing
Reader
```

```
new_sripts Properties
General | Sharing | Security | Previous Versions | Customize
new_sripts
Type:          File folder
Location:      C:\Users\cnaid\OneDrive\Documents\MATLAB\408G\M
Size:          315 MB (330,929,822 bytes)
Size on disk:  315 MB (330,743,808 bytes)
Contains:      633 Files, 0 Folders
Created:       Sunday, April 25, 2021, 10:56:28 PM
Attributes:    [■] Read-only (Only applies to files in folder)
               [ ] Hidden                Advanced...
OK   Cancel   Apply
```

```
« MATLAB > 408G > ML Project > Reader6 > classification

areafunction
AuditoryToolbox
big_training_set
c
female
help
image
lab audio

Name             Date modified
bigw             5/4/2021 11:31 PM
leti             5/10/2021 4:33 AM
picprocessing    5/4/2021 11:30 PM
smallw           5/4/2021 11:14 PM
testingdata      5/4/2021 11:24 PM
v_svd            5/4/2021 11:25 PM
w0               5/4/2021 11:20 PM
```

```
classification Properties
General | Sharing | Security | Previous Versions | Customize
classification
Type:          File folder
Location:      C:\Users\cnaid\OneDrive\Documents\MATLAB\408G\M
Size:          321 MB (336,946,421 bytes)
Size on disk:  321 MB (336,953,344 bytes)
Contains:      7 Files, 0 Folders
Created:       Thursday, May 13, 2021, 9:24:54 PM
Attributes:    [■] Read-only (Only applies to files in folder)
               [ ] Hidden                Advanced...
OK   Cancel   Apply

Reader6
classification
7 items
```

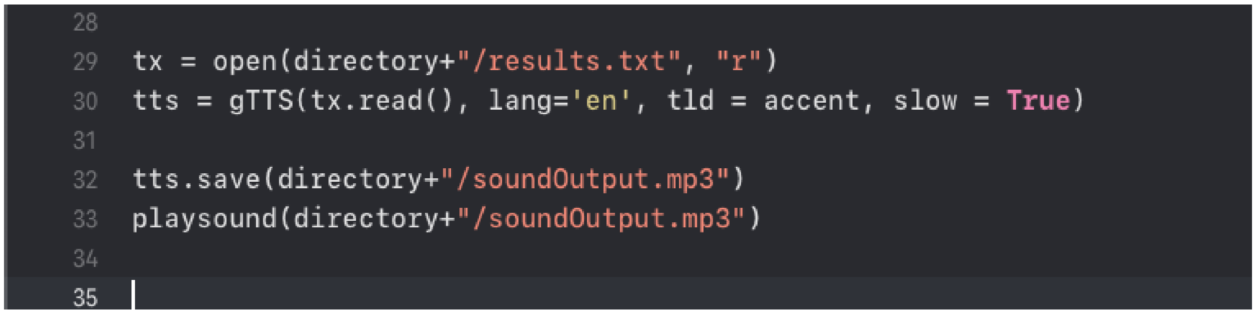
The total number of files went from 633 to just 7. Additionally, the processing time for each letter went from 4.14 seconds to 2.6 seconds.

V. Text to speech:

I initially had planned to create the entire text to speech program myself using concatenative synthesis. My plan was to use an online dictionary to obtain the IPA spelling of common words and store these in a file. For each word, I would look up the IPA spelling and then concatenate the consonant and vowel sounds to create each phoneme that was in the IPA spelling.

I created and carefully cropped over 40 recordings of consonants and vowels and spent many hours experimenting with concatenating them to create phonemes. I also experimented with modulating the volume of the vowels and consonant sounds based on the stress markings in the IPA spelling of the word. The words generated by concatenating my recordings were somewhat intelligible when listened to individually. However, when I started combining these words to generate sentences, they were not intelligible. I gained appreciation for how difficult the task of speech synthesis is, and found that creating my own speech synthesizer would require too much work for one class, especially given how much other work our project required.

I decided that I would have to use an existing speech synthesizer for this project. I experimented with using Google's WaveNet voice which is extremely realistic and runs on Google's cloud servers. However, this solution was not free and required using the Google cloud computing console with authenticated non-human accounts. After this, I found the (Google Text to Speech) gTTS library for Python which is free and runs on your computer without an authentication process. Doing text to speech with this library was very simple. You pass the gTTS function text and it returns a speech object that you can save on the computer as an mp3 file. The first two lines of the following code snippet read the text from a text file and then convert it to speech with the gTTS library. The last two lines of code save it to the computer and then play it over the speakers with Python's Playsound library.



```
tx = open(directory+"/results.txt", "r")
tts = gTTS(tx.read(), lang='en', tld = accent, slow = True)

tts.save(directory+"/soundOutput.mp3")
playsound(directory+"/soundOutput.mp3")
```

Having a simpler solution to converting the text to speech gave me more time to work on the GUI, and the Python script to capture an image from the webcam.

Discussions of future work and improvements

For future improvements, our goal should be to make the software more distributable to other computers. Currently, the code is a combination of Matlab and Python scripts and is heavily dependent on the paths of the files. Also, it is unlikely that any of our customers will have Matlab installed. To make our code more portable, we could rewrite all the Matlab code in Python using the NumPy library which is very similar to Matlab. We can then use the PyInstaller program to create an executable file which will be easy to distribute to our customers. Then it will not depend on which file paths the files are in and which Python libraries are installed on the user's computer.

We could also replace the webcam stand with a new model that standardizes the distance from the table to the webcam. This could prevent movement which would make the system immune to being bumped out of position.

Another focus should be on making our system work with smaller fonts sizes. This could be accomplished by using a higher resolution webcam and using better classification techniques. If we could train a neural network on a very large dataset, it would improve our classification rates and allow smaller text sizes to be detected.

A significant improvement will be having a feature to read text in one language and output in another. With this feature, we could expand our target base to tourists in a foreign country in addition to people with visual disabilities. There are online tools that can easily read data from one language and output a text or speech into a different language of the user's choosing. The only limitation to this is the number of languages that can be translated.

The most important improvement from our current algorithms would be the processing time. To run the OCR for a full picture can take upwards of 5 minutes depending on the number of characters to be read. There are multiple ways we can improve on this. One of them could be to run the OCR on a supercomputer using a cloud server.

Finally, an important part of many documents is the formatting. We could improve our program to detect and understand the pages formatting. For example, if the paper is a newspaper that is formatted in columns, we would want it to detect the line break and move on to the next line instead of skipping over the next column. Or if the page has images or logos, we would want it to skip over that part. Creating better programs to detect these features, possibly using machine learning, we could create a more useful system.

Division of Work

Cherif: Optical Character Recognition, txt file write up

John: Webcam Python script, GUI, and Text to speech Python script

Zeting: Characters segmentation

Conclusion

Overall, our project worked well and accomplished our goal of creating a system which could read out the text on a piece of paper. While it did not work perfectly, it was able to run smoothly with high accuracy for a prototype. We also made it accessible enough so that it could be used by someone with vision problems.

Citations:

[1] Scanlon, J. (n.d.). READERS: WHAT ARE THEY AND HOW DO BLIND PEOPLE USE THEM? Retrieved from https://nfb.org//sites/default/files/images/nfb/publications/fr/fr6/issue1/f060109.html

[2]Visual impairment, blindness cases in U.S. expected to double by 2050. (2016, May 19). Retrieved from https://www.nih.gov/news-events/news-releases/visual-impairment-blindness-cases-us-expected-double-2050

[3]Parul Sahare, Sanjay B. Dhok "Multilingual Character Segmentation and Recognition Schemes for Indian Document Images".

[4] Patriot voice Plus. (n.d.). Retrieved from https://patriotvisionindustries.com/product/patriot-voice-plus/

Code citations:

We used a few lines of code from the documentation for OpenCV and gTTS libraries when making the Python scripts. We took these to learn how to use the libraries.

1. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html
2. https://gtts.readthedocs.io/en/latest/

Additionally, some of the code for the Bayes classifier was written by Cherif for a ENEE436 project to classify faces but was very extensively modified for use in this project.