Presented to the **Gokongwei College of Engineering Department**

De La Salle University - Manila

A.Y. **2022 - 2023**

In partial fulfillment

of the course

Signals, Spectra, and Signal Processing Laboratory (LBYEC4A - EK2)

OCR and CRAFT Analysis of Road Signs

Submitted by:

Ira Third L. Burgos

Alfred Felix A. Daanoy

Michaello J. Hermogenes

Submitted to:

Engr. Ramon Stephen Ruiz

April 4, 2023

ABSTRACT

The main target of this project is to implement text detection with an audio output for individuals who are driving and walking with eye conditions. MATLAB will be integrated into the code to simulate and visualize the desired results of this endeavor. The main concepts highlighted concern optical character recognition (OCR) and **detectTextCRAFT**, which is a deep learning model for character region awareness for text detection (CRAFT). Moreover, the **speak** operation will be assimilated to accumulate audio outputs that correspond to the detected texts using **ocr** and **detectTextCRAFT**. Several other syntaxes from MATLAB are utilized to enhance the efficiency and proficiency of the program. A selection of images is also encompassed to determine the success rate and errors present within the code. In the latter part of the paper, recommendations regarding the limitations of this project are stated to instill more knowledge for future researchers who desire to tackle an identical scope.

I. INTRODUCTION

In this project, the students desire to integrate the concepts of image and audio processing. Thus, text detection with audio output is the paramount representation of the endeavor. The programming structure of the code will entail the principles of OCR, which is utilized to recognize the text within an image using optical character recognition, and TTS, which illustrates text-to-speech recognition. This project will mainly focus on utilizing OCR. Due to this, the students desire to harness the OCR Trainer to input images the default program cannot comprehend. This will provide the OCR Trainer with more data regarding the letters and numbers imperative to be detected. If the OCR Trainer fails to acknowledge the inscribed photos, it will be paramount to corroborate them with other functions to increase the probability of reading the texts. This may be concerning the background and segmentation of the images. Thus, the entire code entails several functions that result in the desired outcome of the researchers to produce a program that can potentially benefit individuals who drive or walk with poor eyesight [1][2].

II. THEORETICAL CONSIDERATION

Based on the principles integrated into this project, the entire process will start from MATLAB reading the text inscribed in the image using **imread**, specifically road signs, to outputting the encompassed text as audio [3].

- **imread**

A = imread(filename)
A = imread(filename,fmt)
A = imread(___,idx)
A = imread(___,Name,Value)
[A,map] = imread(___)
[A,map,transparency] = imread(___)

However, Artificial Intelligence (A. I.) installed within MATLAB is not capable of detecting fonts with myriads of variations. Thus, it cannot detect some letters due to its unfamiliarity with the showcased fonts. The students plan to combat this hindrance by training the A. I. of MATLAB by accessing the OCR Trainer application available in the software. With this, the individuals can input images of disparate versions or styles of texts to allow the A. I. recognize the photos. A block diagram is represented below to illustrate the process of training the A. I. of MATLAB [4][5].

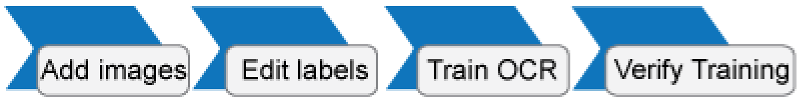


Figure 1.1. OCR Training Block Diagram [1]

Another possible limitation of the A. I. is the inefficiency when the images have a background. Sequentially, the students can mitigate the effects of this predicament by removing the background by utilizing the function **im2gray** or **rgb2gray**. This compels MATLAB to lessen the saturation of the image to underscore the texts visible. The codes below are possible implementations of the operation **im2gray** [6][7]. The operations **graythresh** is required for this program. The functions **strel**, **edge**, and **flatten (flattenlayer)** are optional and can also be applied to amplify the legibility of the generated images [8][9][10[11]. The following syntaxes of the codes are showcased below [6][7][8][9][10][11].

- **im2gray**

I = im2gray(RGB)

- **im2gray**

I = rgb2gray(RGB)
newmap = rgb2gray(map)

- **graythresh**

T = graythresh(I)
[T,EM] = graythresh(I)

- **strel**

SE = strel(nhood)

SE = strel("diamond",r)
SE = strel("disk",r)
SE = strel("disk",r,n)
SE = strel("octagon",r)
SE = strel("line",len,deg)
SE = strel("rectangle",[m n])
SE = strel("square",w)

SE = strel("cube",w)
SE = strel("cuboid",[m n p])
SE = strel("sphere",r)

- **edge**

BW = edge(I)
BW = edge(I,method)
BW = edge(I,method,threshold)
BW = edge(I,method,threshold,direction)
BW = edge(___,"nothinning")
BW = edge(I,method,threshold,sigma)
BW = edge(I,method,threshold,h)

[BW,threshOut] = edge(___)
[BW,threshOut,Gx,Gy] = edge(___)

- **flatten (flattenlayer)**

layer = flattenLayer
layer = flattenLayer('Name',Name)

Sequentially, the modified image will be the input for the function **imbinarize**, which allows MATLAB to convert the 2-D or 3-D grayscale image into binary values [12]. The previous function will be supported by **imcomplement** to accumulate the complement of the processed image [13]. The resulting image will then be tabulated as an array by incorporating the operation **repmat** [14]. The code **detectTextCRAFT** (which is the main function in this program and detects texts in images by utilizing the character region awareness for text detection (CRAFT) deep learning model) will be integrated to initiate the process of segmentation by incorporating **insertShape** [15][16]. This allows MATLAB to process the letter or numbers in the image, which also leads to applying **sort** and **sortrows** to organize the arrangement of the matrix, specifically the columns and rows, respectively [17][18]. The syntax **cat** is also a practical code since it links two matrix values into one [19]. Thus, a single matrix filled with information is generated. This matrix will then be assimilated into **showShape** to exhibit the segmented image, which will then be represented by using **imshow** or **montage** (for side-to-side comparison) [20][21][22]. The result will then be processed in an audio processing code, which will be outputted with **speak** [23]. The codes for the above-mentioned programs are displayed below [11][12][13][14][15][16][17][18][19[20][21][22][23].

- **imbinarize**

BW = imbinarize(I)
BW = imbinarize(I,method)
BW = imbinarize(I,T)
BW = imbinarize(I,"adaptive",Name,Value)

- **imcomplement**

J = imcomplement(I)

- **repmat**

B = repmat(A,n)
B = repmat(A,r1,...,rN)
B = repmat(A,r)

- detectTextCRAFT

```
bboxes = detectTextCRAFT(I)
bboxes = detectTextCRAFT(I,roi)
bboxes = detectTextCRAFT(___,Name=Value)
```

- insertShape

```
RGB = insertShape(I,shape,position)
RGB = insertShape(___,Name=Value)
```

- sort

```
B = sort(A)
B = sort(A,dim)
B = sort(___,direction)
B = sort(___,Name,Value)
[B,I] = sort(___)
```

- sortrows

```
B = sortrows(A)
B = sortrows(A,column)
B = sortrows(___,direction)
B = sortrows(___,Name,Value)
[B,index] = sortrows(___)

tblB = sortrows(tblA)

tblB = sortrows(tblA,'RowNames')
tblB = sortrows(tblA,rowDimName)
tblB = sortrows(tblA,vars)
tblB = sortrows(___,direction)
tblB = sortrows(___,Name,Value)
[tblB,index] = sortrows(___)
```

- cat

```
C = cat(dim,A,B)
C = cat(dim,A1,A2,…,An)
```

- showshape

showShape(shape,position)
showShape(___,Name=Value)

- imshow

imshow(I)
imshow(I,[low high])
imshow(I,[])
imshow(RGB)
imshow(BW)
imshow(X,map)
imshow(filename)
imshow(___,Name=Value)

himage = imshow(___)

imshow(Im,R)
imshow(X,R,map)

- montage

montage(I)
montage(imagelist)
montage(filenames)
montage(imds)
montage(___,map)
montage(___,Name,Value)
img = montage(___)

- speak

speak(y)
speak(y,voice)

In light of the aforementioned information, this project should produce a product that proficiently detects the fonts and generates audio concerning the data exhibited in the photos without any errors in the process. This will mainly be applied to road signs to benefit individuals who have poor eyesight. Henceforth, with the aid of outputting the words stated, the populace will have an easier experience walking or driving in public.

III. METHODOLOGY

Using pre-trained algorithms, the programming platform MATLAB is utilized in order to generate the code for text detection and audio output. The process for these techniques for the code includes a step-by-step process which is significant in order to avoid complications.

a. Image Loading

In order to import the working image for the project, the function **imread** is utilized. This function accepts the specific file path of the image and returns a three-dimensional matrix containing the RGB values of the image.

b. Image Processing

For this process, the functions **im2gray** and **imbinarize** are used. **im2gray** converts the image from a three-dimensional matrix into a two-dimensional matrix making the output into a grayscale image, which makes it easier to process. The function **imbinarize**, on the other hand, converts the image into a black-and-white image.

c. Text Detection

The Algorithm CRAFT (Character Region Awareness for Text Detection) was used. CRAFT is used to identify the specific text in the input image. With this, the function **detectTextCRAFT** was utilized in order to take the image as an input and return a set of bounding boxes for each region of text detected. In addition, the function **sortrows** is used to sort the position of the bounding boxes from top to bottom. This is to ensure accuracy when the text is read as output.

d. Text Recognition

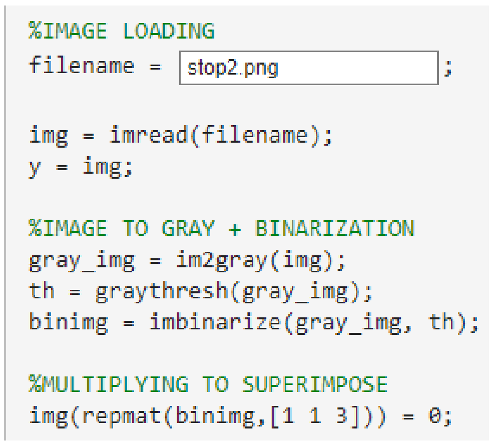
In order to recognize each text in the bounded boxes, another algorithm is used, which is the OCR (Optical Character Recognition) Algorithm. With this, its corresponding function, **ocr()**, was utilized, which recognizes the text in the specific bounded box, which can be either from a binary, inverted binary, or original image depending on the preference of the user. The function returns a structure with the detected text and its location together with the image and sorted bounding boxes as inputs.

e. Audio Dictation

In order for the output from the text recognition to be dictated as audio, the *System Speech* assembly is used. With this, the output structure is used to extract the recognized text, and then a voice synthesizer is used to turn each word into audible speech.

IV. RESULTS

This section presents the MATLAB-generated codes as well as the output images constructed during the testing and debugging phase. Additionally, this aims to establish a strong basis for assessing and comparing data, which will be further examined in line with the theoretical applications studied in this course. Shown below is the generated code for text detection and audio output.



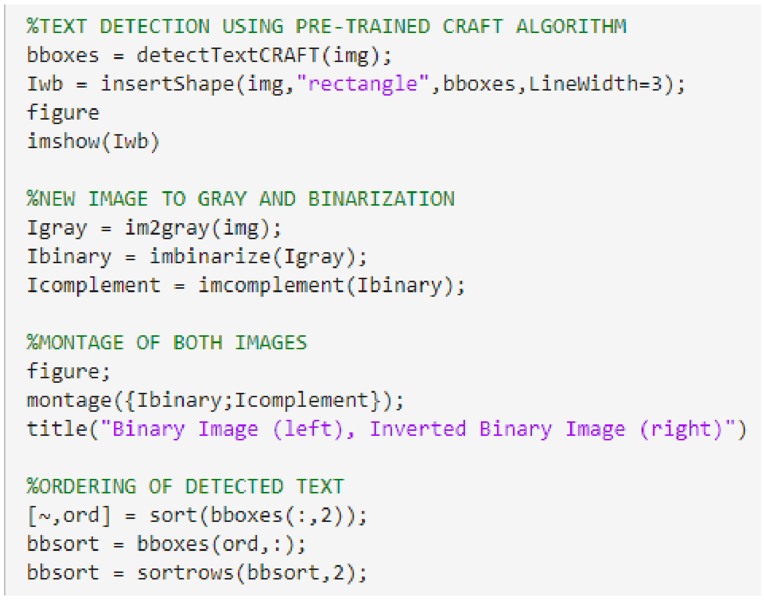
```
%IMAGE LOADING
filename = stop2.png ;

img = imread(filename);
y = img;

%IMAGE TO GRAY + BINARIZATION
gray_img = im2gray(img);
th = graythresh(gray_img);
binimg = imbinarize(gray_img, th);

%MULTIPLYING TO SUPERIMPOSE
img(repmat(binimg,[1 1 3])) = 0;
```

Figure 3.1. Code for Image Loading and Image Processing



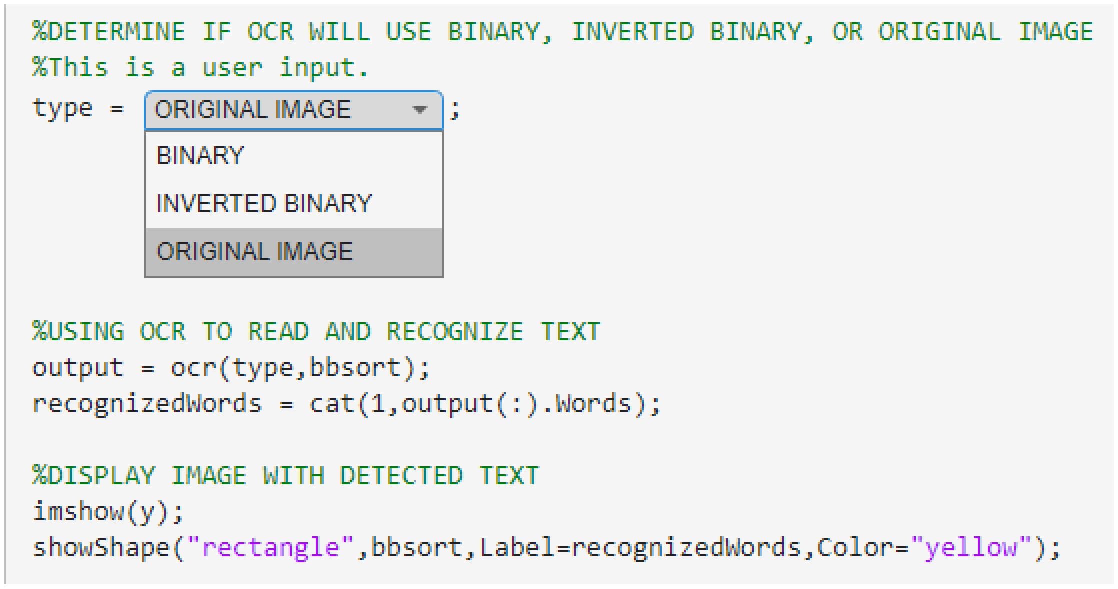
```
%TEXT DETECTION USING PRE-TRAINED CRAFT ALGORITHM
bboxes = detectTextCRAFT(img);
Iwb = insertShape(img,"rectangle",bboxes,LineWidth=3);
figure
imshow(Iwb)

%NEW IMAGE TO GRAY AND BINARIZATION
Igray = im2gray(img);
Ibinary = imbinarize(Igray);
Icomplement = imcomplement(Ibinary);

%MONTAGE OF BOTH IMAGES
figure;
montage({Ibinary;Icomplement});
title("Binary Image (left), Inverted Binary Image (right)")

%ORDERING OF DETECTED TEXT
[~,ord] = sort(bboxes(:,2));
bbsort = bboxes(ord,:);
bbsort = sortrows(bbsort,2);
```

Figure 3.2. Code for Text Detection

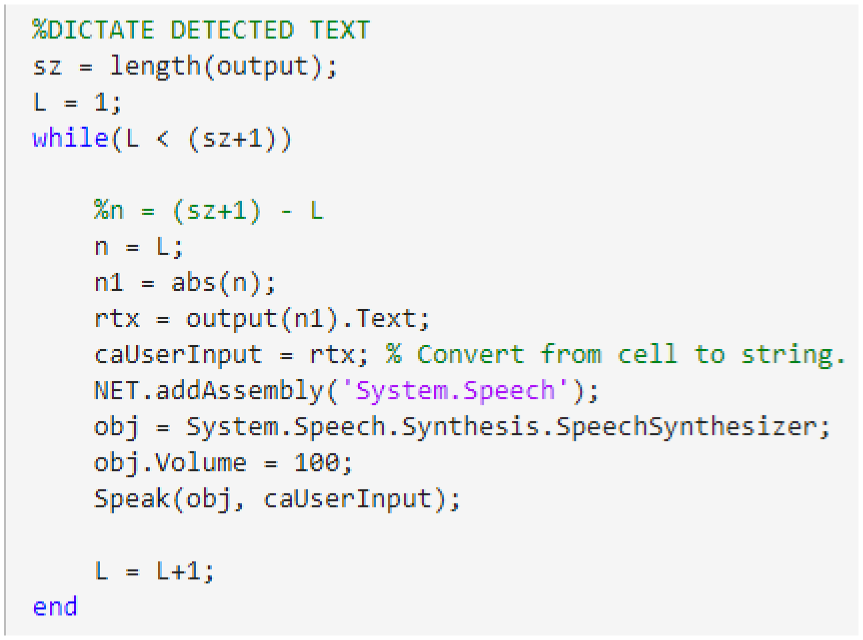


```
%DETERMINE IF OCR WILL USE BINARY, INVERTED BINARY, OR ORIGINAL IMAGE
%This is a user input.
type = ORIGINAL IMAGE;

%USING OCR TO READ AND RECOGNIZE TEXT
output = ocr(type,bbsort);
recognizedWords = cat(1,output(:).Words);

%DISPLAY IMAGE WITH DETECTED TEXT
imshow(y);
showShape("rectangle",bbsort,Label=recognizedWords,Color="yellow");
```

Figure 3.3. Code for Text Recognition



```
%DICTATE DETECTED TEXT
sz = length(output);
L = 1;
while(L < (sz+1))

    %n = (sz+1) - L
    n = L;
    n1 = abs(n);
    rtx = output(n1).Text;
    caUserInput = rtx; % Convert from cell to string.
    NET.addAssembly('System.Speech');
    obj = System.Speech.Synthesis.SpeechSynthesizer;
    obj.Volume = 100;
    Speak(obj, caUserInput);

    L = L+1;
end
```

Figure 3.4. Code for Audio Dictation

Subsequently, during testing, there were instances where the text detection was successful in detecting the text in the images; however, there were also instances that the text detection was unsuccessful due to several factors taken into account with regard to the image (low resolution image, diminutive text size on the image, unique orientation/placement of the text, etc.). The results of both successful and unsuccessful tests are shown in the following figures below.

a. Successful Tests

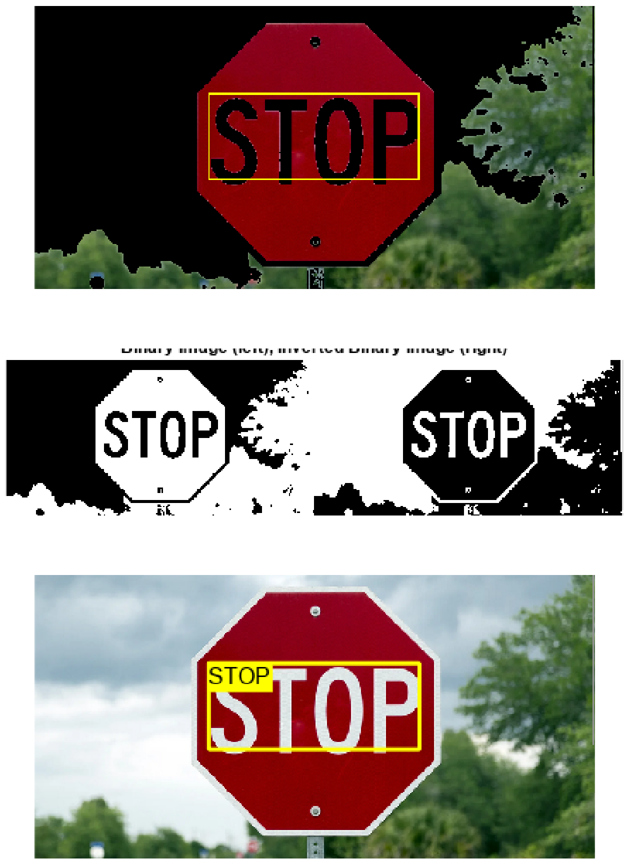


Figure 3.5. Successful Text Recognition Test 1 (OCR Set to Inverted Binary Image)

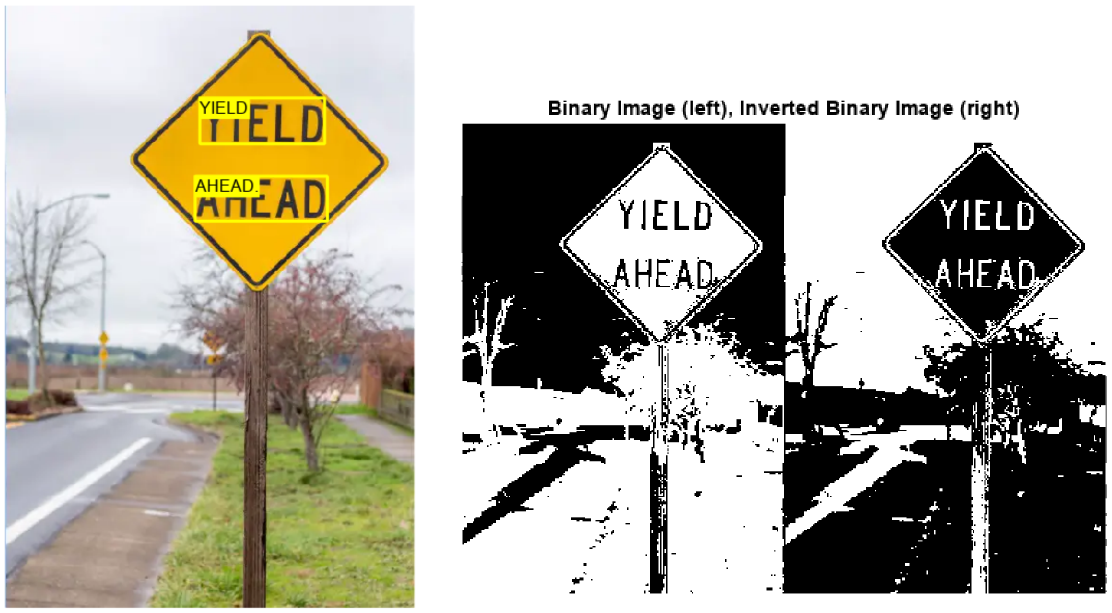


Figure 3.6. Successful Text Recognition Test 2 (OCR Set to Binary Image)

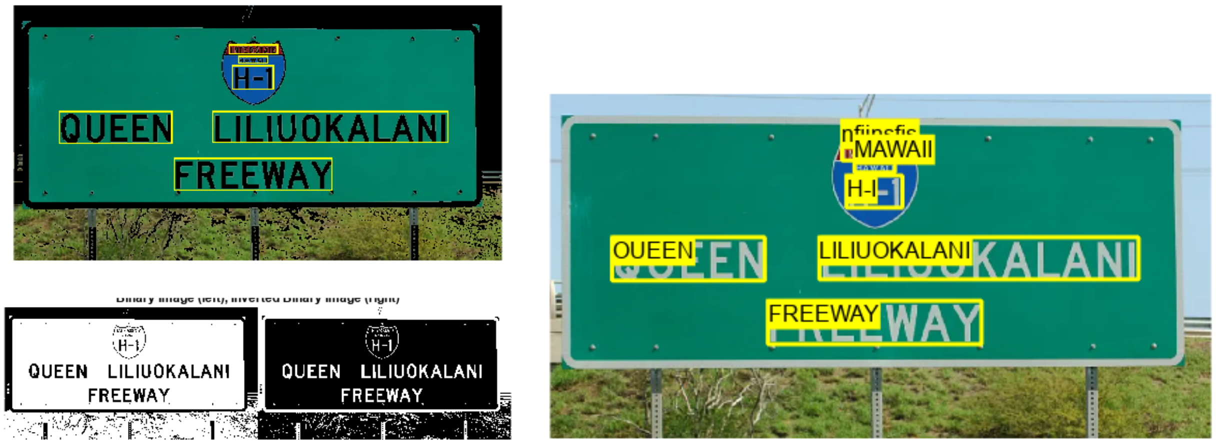


Figure 3.7. Successful Text Recognition Test 3 (OCR Set to Original Image)

b. Unsuccessful Tests

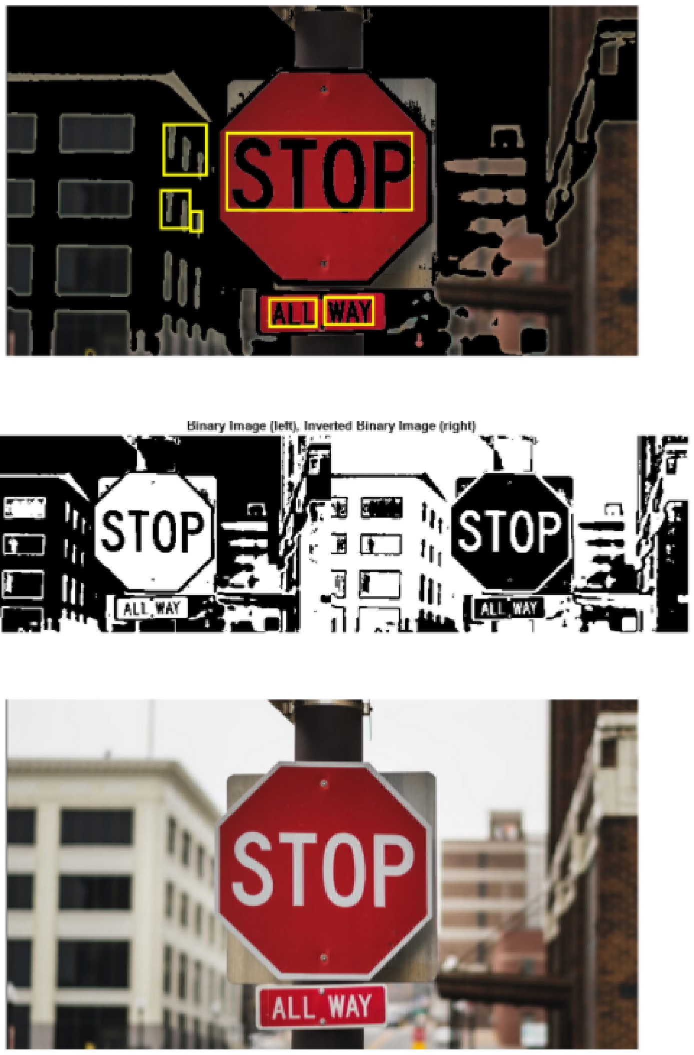


Figure 3.8. Unsuccessful Text Recognition Test 1 (OCR Set to Inverted Binary)

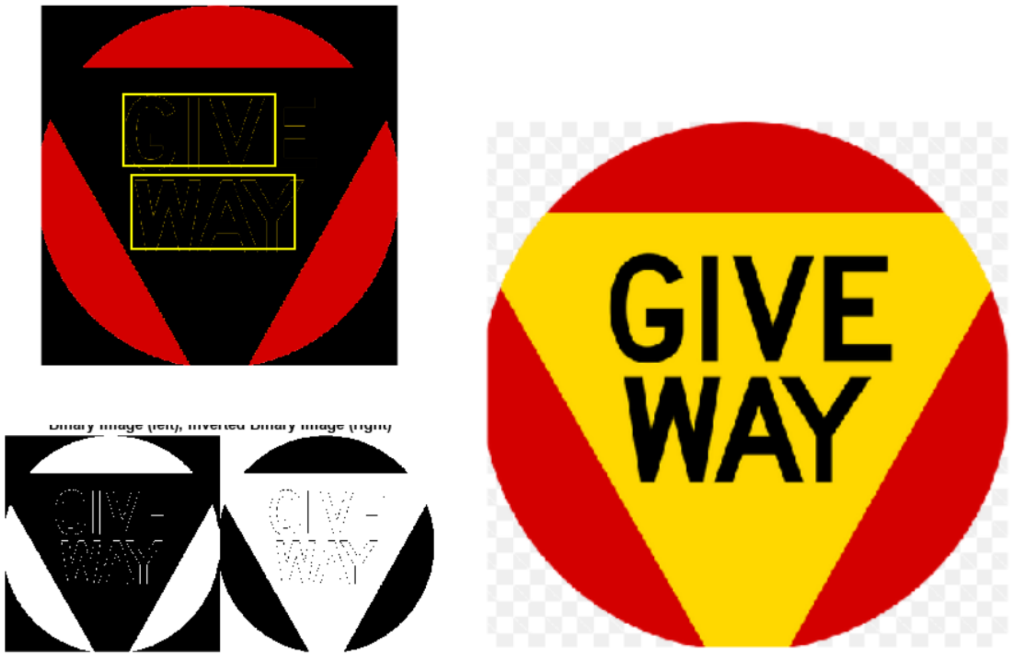


Figure 3.9. Unsuccessful Text Recognition Test 2 (OCR Set to Inverted Binary)

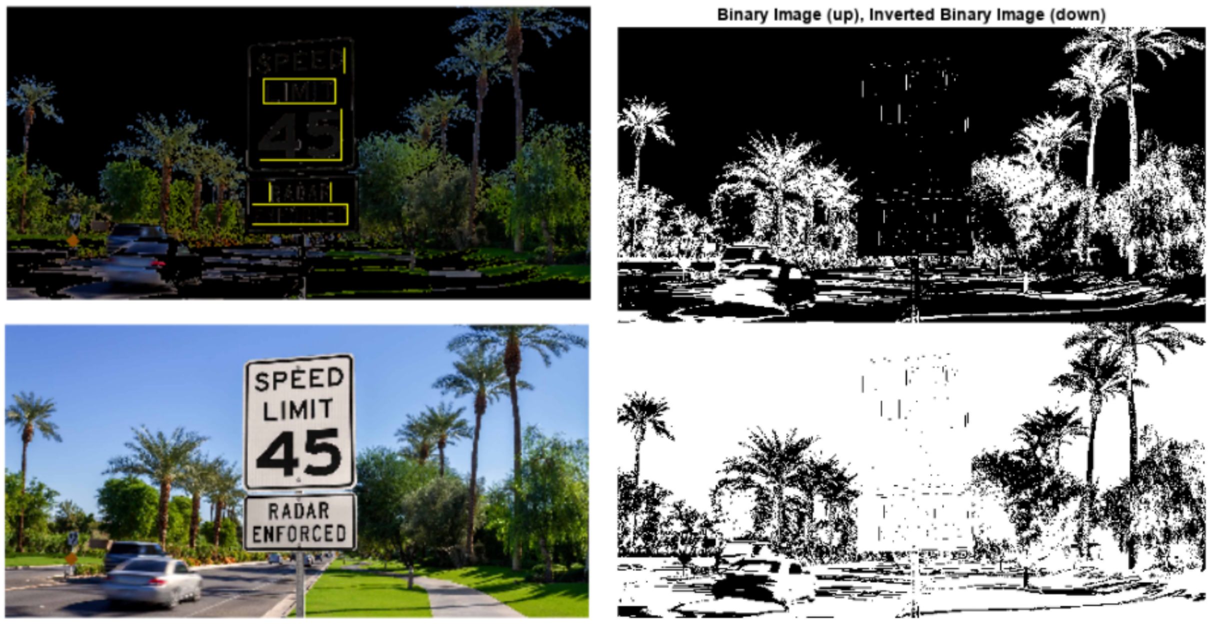


Figure 3.10. Unsuccessful Text Recognition Test 3 (OCR Set to Inverted Binary)

V. DISCUSSION

For this project, the goal is to utilize the software MATLAB to analyze, extract, read, and dictate text from road signs. This can be done through the use of OCR and a pre-trained text detection algorithm called CRAFT.

The code itself has multiple sections.

a. Image Loading

This section is made for the user to input the image that will be analyzed down the line. This section utilizes the **imread** command. This command takes the filename of an image (alongside its file extension) and places it in a variable within MATLAB. In the case of this project, the filename is inputted on a textbox, and its contents are then sent to the actual reading command.

A second variable, **y**, is created as being equal to the image itself. This is there for output purposes, as the **img** variable will be used for filtering. It will also be used as an alternative image for OCR analysis.Using the **img** variable as the alternative for OCR analysis is not ideal because in the output image, it will display a filtered image instead of the original, input image.

b. Initial Filtering

The next section makes use of numerous commands to try and isolate the background of a given image, which was already read and stored in a variable from the previous section.

The first part is to convert the image to gray. This can be done by using the command **im2gray**. Turning the image to gray removes color and thus reduces the complexity of the image. Since the image is now grayscale, computing the global threshold of the image is now possible.

Using the command **graythresh**, the formerly gray image is now converted to a computation of its threshold. This, alongside the **imbinarize** command, can now produce a 2-dimensional image with as much of its background removed. The code for this section and the previous one can be seen in figure 3.1.

The last step is to multiply the resulting image from **imbinarize** to the original image to superimpose it. Superimposing these two images results in maximum background removal—as such, the command **repmat** was used, with parameters **[1 1 3]** to fully maximize the background removal.

c. Text Detection

Arguably the most important section of this project, the algorithm used for text detection is the CRAFT algorithm. CRAFT, which stands for “Character Region Awareness for Text Detection”, is a pre-trained algorithm that detects text that has known borders around them. This means that it works best with signages and text that stand out from the background. In this project, as most of the background has now been removed, the text within road signs now are much more prominent.

The command **detectTextCRAFT** takes the input image as its input, and is equal to a variable that stores the region of detected text—this variable is **bboxes** in the project's case. Since this variable now contains the regions of the detected text, it would be best if it would be visualized—which in this case utilizes the **insertShape** command, with **rectangles** as the primary shape.

The expected output of the insertShape command would be boxes around the supposedly detected text. This, however, is not perfect—there are cases where the algorithm detects text where there are none. This scenario can be seen in the results section, particularly in figure 3.8, where windows of the building behind the stop sign are detected as text—this then causes an error in the code. However, when text has been properly detected, the OCR code can now properly decode this and insert the detected text, as seen in figures 3.5 and 3.6.

d. Secondary Filtering

Before inserting the image for OCR analysis, a second set of filtering was implemented. This is similar to the first set, which uses **im2gray** and **imbinarize**. The second set of filters, however, does not utilize **graythresh**; instead, it now uses **imcomplement**, which is essentially the inverse of whatever image was inputted.

After the image was filtered, the resulting images from **imbinarize** and **imcomplement** are put together in a montage to compare their differences. This is crucial in OCR analysis because sometimes OCR cannot read one type of filtered image but can read another.

Sorting the boxes from the text detection algorithm earlier is also crucial for a later step. If the boxes of text were not sorted, OCR will still be able to properly decode the text. However, the dictation section of the project will not be able to dictate the text in the proper order of left to right, top to bottom. This can be done using the **sort** and **sortrow** commands, which sort the columns and rows, respectively.

e. OCR

Another important aspect of this project is OCR itself. OCR, or "Optical Character Recognition", utilizes a deep-learning algorithm that "reads" text from an image. If **detectTextCRAFT** locates and isolates areas with text, OCR reads the text within those areas.

OCR can be initialized using the **ocr** command. This, by default, uses the English language in detection and decoding. If, however, the text within the image is not a normal font (cursive, road signs), OCR may not be able to properly read these. An example of this can be found in figure 3.7, wherein the sign, which supposedly should read "INTERSTATE H-1" instead reads "nfinsfins H-I". This problem can be alleviated by training the OCR algorithm to custom text, which can be done by using a built-in MATLAB addon. In this project's case, the default algorithm was used, as there are time constraints in training the algorithm.

Another way to prevent misreads from OCR is to change what image it reads in the first place. In this case, there are three different types of images OCR can read, which can be changed by the user. OCR can read either a binary image, the inverted binary image, or the original image. The binary image is taken from the previous section, via the use of the **Ibinary** variable and the **imbinarize** command. The inverted binary image, meanwhile, utilizes the **imcomplement** command, with **Ibinary** as its input. Lastly, the original image uses the earlier duplicate variable, **y**. There are cases, however, where all three fail to be recognized by OCR—these can be found in figures 3.8, 3.9, and 3.10. The reason figure 3.8 fails is because of the text detection algorithm (CRAFT) detecting non-text, as seen when the windows in the background are classified as text. Due to this, OCR tries to read this non-text and eventually fails to read and returns an error. Figure 3.9 fails because the image is two-dimensional to begin with, and OCR seemingly has trouble detecting images that way. Figure 3.10 fails because of the complex background. Both the binary and inverted binary completely mask the road sign in question (which is a speed limit sign), while the original image has too much of a complex background to allow CRAFT and OCR to properly detect and analyze text.

On the bright side, however, most images with road signs were decoded by OCR properly. These can be seen in figures 3.5, 3.6, and 3.7. Figure 3.5 utilizes the inverted binary option, as it has the clearest isolation of the text from the background. Figure 3.6, on the other hand, utilizes the binary option, for the same reason. Lastly, figure 3.7 utilizes the original image, as the original image is clear enough and the background is not too complex.

An optional yet recommended part of the code is to display the detected text themselves. This is done using the **cat** command, and subsequently the **imshow** and **showShape** commands. These three cause text to appear beside the boxes originally shown in **section C** of this discussion. The entire code for this can be found in figure 3.3.

f. Text Dictation

The last step is to dictate what OCR has decoded. This can be done by first determining the length of the detected text. Afterwards, the use of a while loop with the parameters **L < size+1** is created. Within this while loop, a series of commands are used to dictate the text. The use of **NET.addAssembly('System.Speech')** in particular is of importance, as is **System.Speech.Synthesis.SpeechSynthesizer**. Both of these commands are what dictates the text, as well as **speak**.

This is looped until the last value of **text** has been dictated. Afterwards, the code ends and is then complete. There are, however, errors—sometimes, the dictated text is out of order from the image. This is caused by the mismatching and misordering of the rows and columns of the output OCR. Fortunately, this only happens on slanted or tilted text, which does indeed cause an order difference. Otherwise, the code dictates the text from left to right and top to bottom. The code for this can be found in figure 3.4. The resulting project code has been referenced from [24][25] and [26].

VI. CONCLUSION

The utilization of OCR and CRAFT in detecting road signs can be a lifesaver in a lot of conditions, especially when there are drivers who are in need of assistance. However, by default, these can cause a lot of errors, as shown in the project. These errors, such as false readings, no readings, and wrong dictation can cause massive consequences. As such, patching and fixing these errors are tantamount to ensure maximum safety of the drivers. Numerous parts of the project can be improved by others, such as perfecting OCR itself and using other algorithms that can enhance the foreground and further remove the background. Another point of improvement is to automate the selection of the proper image for OCR analysis, as drivers will not be able to do this. As such, finding an algorithm or code that automatically selects the best image for OCR analysis will solve this issue.

Overall, while the code can detect and decode static road signs, the best successor to this is a code that can detect and decode while the image is moving, as in a car. However, that requires numerous algorithms and commands that are reserved for higher level courses or professions.

VII. AUTHORS' CONTRIBUTIONS

1. **Ira Third L. Burgos** - Methodology and Results
2. **Alfred Felix A. Daanoy** - Discussion and Conclusion
3. **Michaello J. Hermogenes** - Abstract, Introduction, and Theoretical Considerations

VIII. REFERENCES

[1] Mathworks. “ocr.” *Mathworks*. Recognize text using optical character recognition - MATLAB ocr (mathworks.com) [accessed Mar. 14, 2023].

[2] Mathworks. “Text-to-Speech Conversion.” *Mathworks*. Text-to-Speech Conversion - MATLAB & Simulink (mathworks.com) [accessed Mar. 14, 2023].

[3] Mathworks. “imread.” *Mathworks*. Read image from graphics file - MATLAB imread (mathworks.com) [accessed Mar. 14, 2023].

[4] Mathworks. “OCR Trainer.” *Mathworks*. Train an optical character recognition model to recognize a specific set of characters - MATLAB (mathworks.com) [accessed Mar. 14, 2023].

[5] Mathworks. “Train Optical Character Recognition for Custom Fonts.” *Mathworks*. Train Optical Character Recognition for Custom Fonts - MATLAB & Simulink (mathworks.com) [accessed Mar. 14, 2023].

[6] Mathworks. “im2gray.” *Mathworks*. Convert RGB image to grayscale - MATLAB im2gray (mathworks.com) [accessed Mar. 14, 2023].

[7] Mathworks. “rgb2gray.” *Mathworks*.Convert RGB image or colormap to grayscale - MATLAB rgb2gray (mathworks.com) [accessed Mar. 23, 2023].

[8] Mathworks. “strel.” *Mathworks*. Morphological structuring element - MATLAB (mathworks.com) [accessed Mar. 14, 2023].

[9] Mathworks. “edge.” *Mathworks*. Find edges in 2-D grayscale image - MATLAB edge (mathworks.com) [accessed Mar. 14, 2023].

[10] Mathworks. “flattenlayer.” *Mathworks*. Flatten layer - MATLAB (mathworks.com) [accessed Mar. 23, 2023].

[11] Mathworks. “graythresh.” *Mathworks*. Global image threshold using Otsu's method - MATLAB graythresh (mathworks.com) [accessed Mar. 23, 2023].

[12] Mathworks. “imbinarize.” *Mathworks*. Binarize 2-D grayscale image or 3-D volume by thresholding - MATLAB imbinarize (mathworks.com) [accessed Mar. 23, 2023].

[13] Mathworks. “imcomplement.” *Mathworks*. Complement image - MATLAB imcomplement (mathworks.com) [accessed Mar. 23, 2023].

[14] Mathworks. “repmat.” *Mathworks*. Repeat copies of array - MATLAB repmat (mathworks.com) [accessed Mar. 23, 2023].

[15] Mathworks. “detectTextCRAFT.” *Mathworks*. Detect texts in images by using CRAFT deep learning model - MATLAB detectTextCRAFT (mathworks.com) [accessed Mar. 23, 2023].

[16] Mathworks. “insertShape.” *Mathworks*. Insert shapes in image or video - MATLAB insertShape (mathworks.com) [accessed Mar. 23, 2023].

[17] Mathworks. “sort.” *Mathworks*. Sort array elements - MATLAB sort (mathworks.com) [accessed Mar. 23, 2023].

[18] Mathworks. “sortrows.” *Mathworks*. Sort rows of matrix or table - MATLAB sortrows (mathworks.com) [accessed Mar. 23, 2023].

[19] Mathworks. “cat.” *Mathworks*. Concatenate arrays - MATLAB cat (mathworks.com) [accessed Mar. 23, 2023].

[20] Mathworks. “showShape.” *Mathworks*. Display shapes on image, video, or point cloud - MATLAB showShape (mathworks.com) [accessed Mar. 23, 2023].

[21] Mathworks. “imshow.” *Mathworks*. Display image - MATLAB imshow (mathworks.com) [accessed Mar. 23, 2023].

[22] Mathworks. “montage.” *Mathworks*. Display multiple image frames as rectangular montage - MATLAB montage (mathworks.com) [accessed Mar. 23, 2023].

[23] “speak.” *MATLAB Function Reference*. speak (MATLAB Function Reference) (jhu.edu) [accessed Mar. 23, 2023].

[24] M. Waris. “Background remove from Image.” *Mathworks.* Background remove from Image - File Exchange - MATLAB Central. [accessed Mar. 30, 2023].

[25] Mathworks. “Automatically Detect and Recognize Text Using MSER and OCR.” *Mathworks.* Automatically Detect and Recognize Text Using MSER and OCR - MATLAB & Simulink. [accessed Mar. 30, 2023].

[26] Mathworks. “Automatically Detect and Recognize Text Using Pretrained CRAFT Network and OCR.” *Mathworks.* Automatically Detect and Recognize Text Using Pretrained CRAFT Network and OCR - MATLAB & Simulink (mathworks.com) [accessed Mar. 30, 2023].