

LICENSE PLATE CHARACTER SEGMENTATION AND RECOGNITION

Contents

LIST OF FIGURES	3
LIST OF ACRONYMS	3
1. INTRODUCTION.....	4
2. PROBLEM STATEMENT & OBJECTIVE.....	5
2.1. Problem Statement	5
2.2. Objective	5
3. METHODOLOGY	6
3.1. Defining Custom Made Functions and Importing Python Libraries.	8
3.2. Reading the Image from Project Directory and Resizing the Image.	9
3.3. Defining Kernels Used in Morphological Operations	9
3.4. Performing Image Processing.	9
4. TESTING PHASE	21
5. RESULTS	22
6. CRITICAL ANALYSIS OF RESULTS	30
7. CONCLUSION AND FUTURE RECOMMENDATIONS	33
8. REFERENCES	34
9. APPENDICES.....	35

LIST OF FIGURES

FIGURE 1: PROCESS FLOWCHART OF PROJECT METHODOLOGY	6
FIGURE 2: IMAGE FROM STAGE 1 PROCESSING - BLACKHAT	9
FIGURE 3: IMAGE FROM STAGE 2 PROCESSING - LIGHT	10
FIGURE 4: IMAGE FROM STAGE 3 PROCESSING - L_BH_OR	10
FIGURE 5: IMAGE FROM STAGE 4 PROCESSING - GRADX	11
FIGURE 6: (A) SMOOTHEN GRADX, (B) CLOSING PERFORMED ON GRADX AND (C) IMAGE FROM STAGE 5 PROCESSING GRADX_THRESH. ...	11
FIGURE 7: A COMPARISON BETWEEN L_BH_OR BEFORE (A) AND AFTER BEING OVERLAYED BY GRADX_THRESH (B). THE RESULTANT IMAGE FROM STAGE 5 IS SAVED AS GRADX_THRESH REPLACING ITS PREDECESSOR.	12
FIGURE 8: IMAGE FROM STAGE 6 - GRADX_THRESH WITH 41% DARKENING ROW-WISE AND 50% DARKENING COLUMN-WISE	12
FIGURE 9: IMAGE FROM STAGE 7 - MASK1 OBTAINED FROM CONTOUR DETECTION ON GRADX_THRESH.	13
FIGURE 10: PERFORMING REGION FILLING WITH SEED POINTS ALONG THE 4 EDGES OF THE IMAGE.	14
FIGURE 11: (A) MASK2, (B) MASK3	15
FIGURE 12: MASK3 OVERLAYED ON THE ORIGINAL IMAGE.	15
FIGURE 13: (A) CROPPED IMAGE, (B) DILATED AND ERODED MASK4, (C) PROCESSED MASK4 OVERLAYED ON THE CROPPED IMAGE	16
FIGURE 14: (A) MASK5, (B) MASK5 OVERLAYED ON THE CROPPED IMAGE.	17
FIGURE 15: PERSPECTIVE TRANSFORMED IMAGE	18
FIGURE 16: SAMPLE OF OUTPUT WHERE ALPHABETS OR NUMBERS ARE MISINTERPRETED AS SIGNS	18
FIGURE 17: A LIST WITH ALL KNOWN SIGNS.	19
FIGURE 18: REMOVAL OF EMPTY FIELDS FROM THE RESULT	19
FIGURE 19: FIGURE SHOWING THAT THE NUMBER CHARACTERS IN MALAYSIAN NUMBER PLATES DO NOT EXCEED 4 CHARACTERS	19
FIGURE 20: LIST 1 TRANSFORMED TO FORM OF LIST 2.	20
FIGURE 21: (A-AA) RESULTS PRODUCED BY ALGORITHM FROM TESTING	29
FIGURE 22: COMPARISON OF PLATES WITH (A) PLATE BORDER, (B) WITHOUT PLATE BORDER.	32

LIST OF ACRONYMS

4WD	4-Wheel Drive
ANPR	Automated Number Plate Recognition
CAGR	Compound Annual Growth Rate
IPCV	Image Processing and Computer Vision
OCR	Optical Character Recognition
VAR	Variable Name

1. INTRODUCTION

The field of Artificial Intelligence has been rapidly developing over these past few years and its growth is projected to increase exponentially over the coming years. The expected global spending on Artificial Intelligence and various intelligent machine technologies was projected to reach an approximate of US\$19.1 Billion in 2018, which is a significant 54.2% increase as compared to the preceding year. [1]. According to the International Data Corporation based on one of its articles, the forecasted investment in AI is expected to double in the upcoming 4 years, growing from US\$ 50.1 billion in 2020 to an expected value of more than US\$ 110 billion in 2024. The compound annual growth rate (CAGR) of AI between years 2019 to 2024 is predicted to be approximately 20.1%. [2]. Organizations are starting to realize how AI can transform their organization by executing tasks and projects with higher efficiency and accuracy to achieve their goals.

A branch of AI that is very prominent is the field of Image Processing and Computer Vision. Digital Image Processing is the field of studying and developing theories, algorithms, and models for image manipulation. Its application covers a broad field such as image histogram manipulation, image sharpening, image blurring, restoration, and compression. Computer vision on the other hand is the field that focuses on theories and algorithms for automating the process of visual perception i.e., mimicking human vision for computers. This is done through means of various tasks such as noise removal, segmenting images to isolate certain regions of focus and interpreting the data within the particular filtered scene. [3][4]. Several examples of the vast use of IPCV in our daily lives are like the Amazon Go store, Kinect by Microsoft, and the Autopilot driven cars by Tesla and Nio, camera image processing and enhancement in smart phones, filter applications on apps such as Instagram and Snapchat and finally in semiconductor manufacturing inspection and testing [5].

This report focuses on the development of an algorithm for Automated Number Plate Recognition (ANPR) via license plate character segmentation and recognition. This report details the problem statement / analysis of the problem, the project objectives, the details on the proposed solution / algorithm, experimental results and testing, discussion followed by the conclusion and recommendation.

2. PROBLEM STATEMENT & OBJECTIVE

ANPR is defined as an accurate system with the capability of reading vehicle number plates without the need of human intervention. This is typically done with the use of high-speed image capture via a camera and algorithms that takes in the image and filters through only areas of interest within the image to process and verify the characters sequences being shown on the car number plates. The system converts image into text data. ANPR's application is vast and can be applied for instances such as in car park management, traffic management, retail park security and intelligent transport systems.

An example of the application of ANPR is seen on many motorways that uses cameras as a method of detecting speeding vehicles through average speed calculation. Its application is beneficial in not only security and public safety but also improves the infrastructure in which transportation revolves around. Hence, the problem statement and objective of this project is defined as below.

2.1. Problem Statement

How can a license plate character segmentation and recognition algorithm be developed for an Automated Number Plate Recognition (ANPR) system for detecting number plates from images of cars entering a gate?

2.2. Objective

The objective of this study is to develop an algorithm for license plate character segmentation and recognition for an ANPR system detecting number plates from images of cars passing through a gate.

3. METHODOLOGY

The process flow diagram of the proposed methodology for this project is displayed in Figure 1 below.

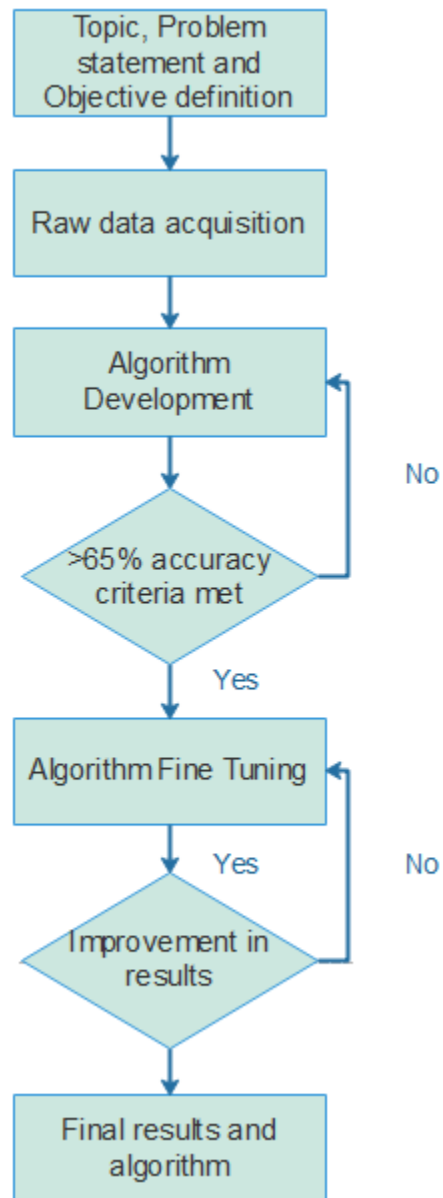


Figure 1: Process Flowchart of Project Methodology

The process begins with the topic, problem statement and objective definition. Next, the raw data to be used for testing and developing the algorithm was acquired. A total of 27 images of vehicles consisting of sedans, 4WD and hatchbacks were acquired. It is important to note that commercial vehicles such as trucks, vans and lorries were not considered. All of the raw image data obtained were white characters printed on a black background plate. All images were also taken indoors with relatively similar angles. These factors were controlled variables in the project in order to set a benchmark for developing the algorithm.

The next step would involve developing the algorithm for processing the images. A detailed outline of the final algorithm implementation is presented from Section 3.1 to Section 3.4. Once the algorithm had been developed, it would then be tested with the raw image data of vehicles to gauge its performance in filtering out noise, segmenting the number plate region from the rest of the image and accurately recognizing the vehicles' number plates. Each image processing stage defined was tested and its results were observed, and the algorithm was fine-tuned throughout the development stage in order to ensure the intended results were obtained.

The 27 images were used for testing the algorithm and the number of car plates that were 100% successfully recognized by the algorithm was counted. The overall percentage of number plate recognition success by the algorithm was calculated to evaluate the algorithms performance / accuracy. Based on its performance, the algorithm was fine-tuned in terms of its parameters and processes to improve its accuracy. A base criterion was set such that the algorithm had to achieve a minimum of 65% accuracy in order for its overall image processing flow operations to be deemed acceptable. This is done in order to establish the baseline of the algorithm. Beyond this point, much finer parameters within the algorithm were tuned such as areas, perimeter, and aspect ratio in order to further increase the algorithms accuracy.

3.1. Defining Custom Made Functions and Importing Python Libraries.

2 custom functions were built for this project which are:

Function Name	Functionality.
Myregionfill(img,seed,pix)	A function that takes in an image and fills regions in the image based on the seed point that was set. Region filling is done within the limits of the boundaries / contours in the image using the cv2.floodFill function. Variables used are the image matrix (img), the seed point (x and y coordinates), and the pixel filling density.
Mytesseract(r)	Converts output results and header from pytesseract into a dictionary format for ease of data management and extraction

Table 1: Built Functions for the algorithm

4 python libraries were imported in this project as detailed in Table 2 below

Python Library	Functionality.
Numpy	Library used for working with matrices / images.
Opencv-python	Image processing library used for a majority of the image processing functions.
Os	Used for writing and removing temporary images from the project directory.
Pytesseract	A python wrapper for Google's Tesseract-OCR. Used for performing OCR on the final processed image to obtain the number plates in string.

Table 2: Imported Python Libraries

3.2. Reading the Image from Project Directory and Resizing the Image.

Next, the user is prompted to enter the image number as named in project directory. Using the `cv2.imread()` function and the input provided, the program then reads the image. Two types of images were read, one which was grayscale (`ori_img`) and in colour (`img_src`). Due to the large size of the raw image, the `cv2.resize` function is used to resize the image to 25% of its original size. A number of copies of the grayscale and coloured image were made for overlaying masks and creating a masked image for subsequent steps of processing.

3.3. Defining Kernels Used in Morphological Operations

A total of 3 kernels were defined which are the rectangle kernel (`rect_kern`) of size 30 x 10 pixels, a cross shaped kernel (`cross_kern`) of size 3 x 3 pixels and finally a square_kern (`square_kern`) of size 3 x 3 as well using the `cv2.getStructuringElement()` function. These kernels were used for various morphological operations.

3.4. Performing Image Processing.

Performing Image Cleaning

STAGE 1:

The raw grayscale image is taken and blackhat operation is performed on the image. Blackhat is a morphological operation whereby the output image produced is the difference between the closing of the input image and the input image. The output blackhat image is then thresholded with a threshold limit of 100 pixels to 255 pixels. The method of thresholding is binary thresholding method. Any grayscale pixel value within the image which is lower than 100 pixels is automatically converted into a 0-pixel value (black) whereas anything above 100 is set to the upper limit which is 255 (white). The resultant image (VAR: `blackhat`) produced from this first operation is shown below in Figure 2. The purpose of this step is to isolate the region around the characters on the number plate as marked red.



Figure 2: Image from Stage 1 Processing - blackhat

STAGE 2:

In stage 2, the original grayscale image is put through a morphological close operation with the square_kern kernel. In the close morphological operation, the image is first dilated and then eroded using the kernel. The image is then binary thresholded using Otsu's method. Otsu's method involves iterating through all of the possible threshold value and calculates the spread for the pixel levels on each side of the threshold. It then decides a threshold value such that it creates a minimum sum between the foreground and background spreads [6]. The image is then inverted to its negative image before performing a final morphological erode operation on the image. The resultant image (VAR: light) is displayed in Figure 3. This operation was performed to isolate the characters on the number plate only and highlight the characters.

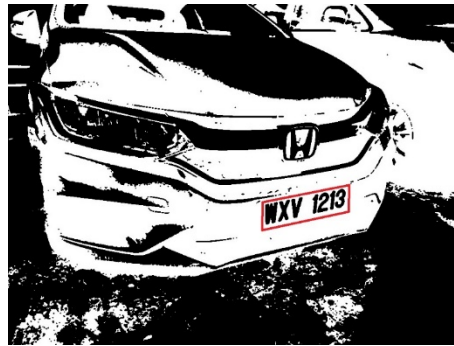


Figure 3: Image from Stage 2 Processing - light

STAGE 3:

A bitwise or operation is then carried out between both of the images from Stage 1 and 2 to produce the resulting image (VAR: L_BH_or) as shown below in Figure 4 where the characters on the number plate are highlighted in white for further processing.



Figure 4: Image from Stage 3 Processing - L_BH_or

STAGE 4:

The blackhat image produced from STAGE 1 is taken and the Sobel gradient is then calculated. The Sobel operator is a combination of gaussian smoothing and differentiation. For this particular case, the Sobel derivative is calculated in the x direction in the first order with a Scharr Filter of size 3 x 3 using the cv2.Sobel() function. The calculated values in the image are then normalized between the range of 0-255 pixels and the data type of the matrix calculated is converted into unsigned integer of 8 bits. The resultant image produced (VAR: gradX) is displayed in Figure 5. The purpose of this stage is to highlight the edges around the number plate characters to distinguish between the characters and the number plate background.



Figure 5: Image from Stage 4 Processing - gradX

STAGE 5:

The gradX image is then smoothen using a Gaussian Blurring function (cv2.GaussianBlur) with a 5 x 5 filter. Morphological close operation is then carried out on the image using the rect_kern. This is done to blend the white areas which are close by, detected in Stage 4 as a consolidated area. The image is subsequently thresholded using Otsu's binary thresholding method to produce the gradX thresholded image (VAR: gradX_thresh) which clearly defines the white regions in the consolidated areas. The image then undergoes morphological dilate operations for 5 iterations to expand this white area using a cross kernel. The progression of the image from gradX to the closed, thresholded and dilated image is shown in Figure 6 below

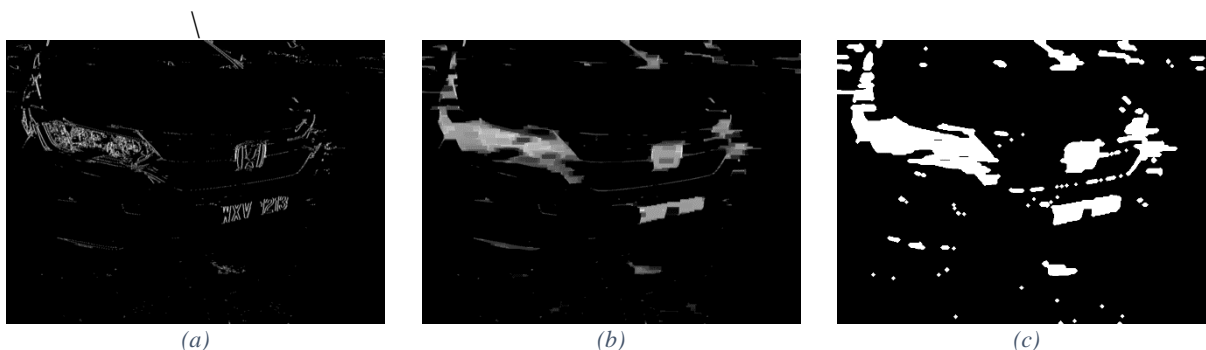


Figure 6: (a) Smoothen gradX, (b) Closing performed on gradX and (c) Image from Stage 5 processing gradX_thresh.

Then, the image serves as a mask and is overlayed on L_BH_or image from Stage 3 by performing a bitwise and operation. The bitwise operation helped in removing a significant amount of the white regions on the L_BH_or image by blackening them. This aids in filtering out more noise in the image, focusing more on the characters on the number plate. The resultant image in Stage 5 is named gradX_thresh. A comparison of before and after overlaying the mask is depicted in Figure 7.



Figure 7: A comparison between L_BH_or before (a) and after being overlayed by gradX_thresh (b). The resultant image from Stage 5 is saved as gradX_thresh replacing its predecessor.

STAGE 6:

Looking at the raw image dataset, it was noticed that the view in which the images were acquired typically placed the number plate on the bottom right corner position of the image. This observation holds true for a majority of the test images acquired. Based on this observation, Stage 6 was implemented to gradX_thresh to blacken 41% of the rows in the image starting from the first row at the top of the image and 50% of the columns in the image from the left side onwards. The following step aimed to further reduce noise present in the image. The resultant image is shown in Figure 8.

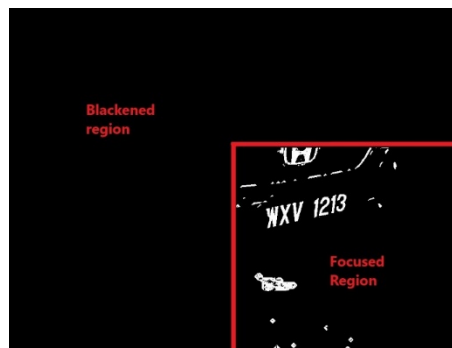


Figure 8: Image from Stage 6 - gradX_thresh with 41% darkening row-wise and 50% darkening column-wise

Contour Detection and Filtering

Once the image was cleaned to an acceptable level, the next stage would involve contour detection and filtering to refine the regions of interest in the image. Contours are detected using the `cv2.findContours` function. Once the regions had been detected, surrounding rectangles are enclosed around the contours using the `cv2.minAreaRect` function. This defines the 4 corners of the rectangles surrounding the contours. The contours are then filtered based on parameters such as area (`cv2.contourArea`), aspect ratio (height and width from `cv2.boundingRect`), perimeter (`cv2.arcLength`), and centroid distance (`cv2.moments`). Regions that fit the criteria of selection are drawn on a mask and overlayed on the original image.

It is important to note that contour detection would result in many contours that have very close parameters to each other making filtering very difficult. Hence in order to mitigate this, steps were taken to perform contour detection and filtering in multiple stages.

STAGE 7:

This is implemented by first filtering out noise contours and drawing the contours of interest on a mask (VAR: `mask1`) based on the contours detected in `gradX_thresh`. The exterior areas of the mask that is not bounded by rectangles are then filled in white using the `myregionfill()` function whereas the regions within the contours remained black. The image is then inverted (`img = 255 - img`) to produce the negative image.

The initial contour detection on `gradX_thresh` would detect multiple contours, but the contours of interest are the ones surrounding each number plate character as shown in red in Figure 9. The first step involved filtering contours based on its area ($\text{area} > 150 \text{ pixel}^2$ & $\text{area} < 5000 \text{ pixels}^2$)

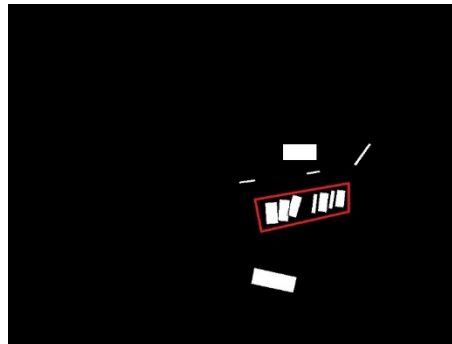


Figure 9: Image from Stage 7 - `mask1` obtained from contour detection on `gradX_thresh`.

As observed, the rectangles of interest surrounding the characters are located close by to one another.

STAGE 8:

The mask1 is then subjected to a second step of contour detection which then produces newer contours that merges the smaller and close by rectangle contours into consolidated larger contours. Isolated contours on the other hand would then be relatively smaller in size and different in parameter. The centroid of each rectangle contour detected in the second stage is determined and its resultant distance from the nearest neighboring contour was calculated. Conditions for filtering were then set with conditions, $\text{min_centroid_distance} < 200$ pixels, $\text{area} > 1000$ pixel² and aspect ratio ($\text{ar} < 3$) to further filter out unnecessary regions. The selected regions are then drawn on a 2nd mask called mask2. It was also observed that some of the unwanted contours in mask2 fell along the edges of the image, as shown in Figure 10 (a). Hence an additional step was taken to perform region filling with `myregionfill()` with seed points all along the 4 edges of the image to eliminate these regions.

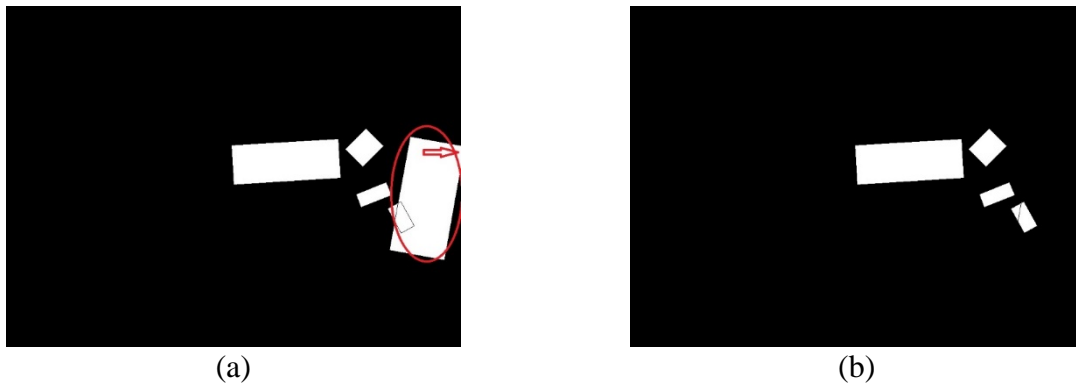


Figure 10: Performing region filling with seed points along the 4 edges of the image.

The resultant image for mask2 produced is shown in Figure 10 (b). Mask2 is then overlayed on a copy of the original image using a bitwise and function. The mask2 image is then saved as a temporary file into the project directory as a .jpg file and re-read into the program. The temporary file is then removed from the project directory using the `os.remove()` function.

STAGE 9:

Contour detection is carried out again on mask2, filtering out more unwanted regions based on their area whereby only regions with $\text{area} > 2000 \text{ pixels}^2$ were selected. The selected regions were then drawn on mask3. The progression of contour filtering from mask2 to mask3 is shown below in Figure 11.



Figure 11: (a) mask2, (b) mask3

Mask3 is then overlayed on a copy of the original-colored image as shown in as shown in Figure 12 via a bitwise and operation.



Figure 12: mask3 overlayed on the original image.

STAGE 10:

In order to reduce unwanted areas in the image, the image was cropped to only include the final selected contour. This is done by creating a new image within the limits of the minimum and maximum row and column-wise corners' coordinates. The image is then binary thresholded with a threshold limit of 100 pixels. The image is assigned to crop_img variable.

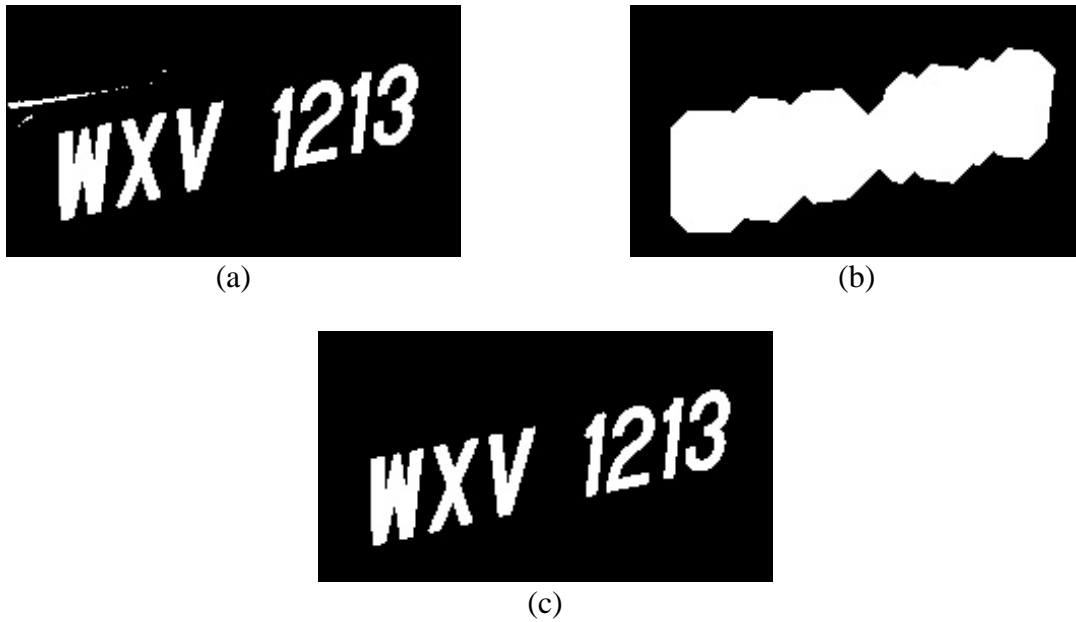


Figure 13: (a) Cropped image, (b) Dilated and eroded mask4, (c) Processed mask4 overlayed on the cropped image

STAGE 11:

Contour detection is then carried out on the cropped image ensuring that the region around the characters of the number plate is only selected based on the conditions $\text{area} > 100 \text{ pixel}^2$ and $\text{area} < 8000 \text{ pixel}^2$ and $\text{height} > 20 \text{ pixel}$ and $\text{height} < 80 \text{ pixel}$ and $\text{aspect ratio} > 0$ and $\text{aspect ratio} < 2.2$ and finally $\text{width} < 150 \text{ pixel}$. The selected contour is drawn on mask4. Mask4 is then dilated for 15 iterations and eroded for 7 iterations to adjust the contour size to only include all of the characters of the number plate and to ensure it did not cover any characters. Mask4 is then overlayed on the crop_img Resultant image is shown in Figure 13 above. Mask 4 is also shown in Figure 13 above.

STAGE 12:

A copy of `crop_img` is then created using the `np.copy` function and named as `final_image`. The final stage of contour detection is then done on `mask4` to detect the final rectangle contour surrounding the number plate characters. The reason why the final stage was carried out instead of ending the process at `mask4` was because for some test cases, it was noticed that `mask4` covered certain areas of the number plate characters which would make it OCR inaccurate. Hence, morphological operations of dilation and erosion were carried out and a final stage of contour detection was carried out. The contour detection from `mask4` were drawn on `mask5` and the contours were selected based on the criteria that $\text{area} > 5000 \text{ pixel}^2$ and $\text{aspect ratio} > 1$. `Mask5` was then overlayed on the `final_image`. `Mask5` and `final_image` are shown in Figure 14.



(a) (b)
Figure 14:(a) mask5, (b) mask5 overlayed on the cropped image.

STAGE 13:

As the final image produced is viewed at an angle / in perspective mode, perspective transform had to be carried out in order to reorientate the final image for better accuracy in OCR readings.

Firstly, the corners from `mask5` were saved into a “corners” variable. If image processing resulted in `mask5` not having any detected contours, random 4 corners were assigned as an exception. Then, the new coordinates for perspective transform were defined at positions, `[80,100]`, `[580,100]`, `[620,250]`, `[120,250]`. These positions were defined by means of testing in order to determine the best size and stretching orientation of the final image for maximizing OCR accuracy. Subsequently, perspective transform was then carried out using the `cv2.PerspectiveTransform()` and the `cv2.warpPerspective()` functions. The final perspective transform image (VAR: `result`) is shown in Figure 15.



Figure 15: Perspective transformed image

OCR and Results Data Filtering.

STAGE 14:

The “final_image” and “result” image are subjected to OCR using `pytesseract.image_to_data()` function using Google's Tesseract OCR Engine and the results are put through the `mytessdata()` function to convert the output into a more formatted and structured dictionary output. Only the text data results were saved in variables `words_transform` (for perspective transformed image result) and `words_cropped` (for final_image without perspective transform) in the form of a list and is further processed.

The reason why both of these images were subjected to OCR was to maximize the possible outcomes to improve accuracy and as failover option. Results from the perspective transform image is given precedence first as the final output and second is given to the image without perspective transform as backup should the result from the primary image be inaccurate.

It is firstly important to note how the un-processed data in `words_transform` and `words_cropped` is produced. The output is grouped into several texts in which the OCR recognizes it, and it may not be as accurate as the original image. Noises are also present in which unrelated signs or characters are detected and finally, some texts or numbers can also be misinterpreted. As shown in Figure 16 below. The actual number plate is actually “WB 1524 S”, but the “S” was detected as “\$” instead.

```
[',', ',', ',', ',', 'WB' '1524' '$']
```

Figure 16: Sample of output where alphabets or numbers are misinterpreted as signs.

This would require the next step which would be to filter and process the data obtained from `words_transform` and `words_cropped` to firstly remove any noise or unwanted texts detected and to also process / replace misinterpreted texts. The steps are detailed as below.

Firstly, empty fields are removed from the lists as shown in Figure 18 as they are deemed to be insignificant. Next a list with all possibly known signs are defined as the variable signs. The variable is shown below in Figure 17.

Figure 17: A list with all known signs.

```
[ 'WXV' '1213']
```

Figure 18: Removal of empty fields from the result

```
[ 'WXV' '1213' ]
```

Figure 19: Figure showing that the number characters in Malaysian number plates do not exceed 4 characters.

Misinterpreted Detected Character	Actual alphabet Character
\$	S
!	I
&	B
(or [C

Then the remaining elements in both lists were split into single length characters as shown in the transformation below from list 1 to list 2 as shown in Figure 20.

['wxv', '1213'] – List 1
['w', 'x', 'v', '1', '2', '1', '3'] – List 2

Figure 20: List 1 transformed to form of List 2.

Conditions were set on the updated lists such that if the first character was misinterpreted based on Table 4, the misinterpreted character was rectified. This was implemented on the basis that the starting character of a Malaysian car number plate is typically an alphabet and not a number or sign.

Misinterpreted Detected Sign	Actual alphabet Character
\$	S
1	I
5	S
4	A
3 or 8	B
6 or &	G
(or [C

Table 4: Conditions for filtering the first characters in the OCR output lists.

After processing, all of the remaining elements are joined as a single string using the list.join() function. 2 final string characters are compared to one another and as mentioned earlier, precedence is given to the output from the perspective transform image which is words_transform. Based on the processing steps defined, should the output from words_transform be inaccurate, the array would be empty and hence a condition was set such that if words_transform was empty, the output from words_cropped would then be chosen as the final output.

4. TESTING PHASE

The performance of the algorithm during its development process was tested using the 27 vehicle images obtained prior. The total number of car plates that were successfully 100% recognized by the algorithm were counted and the percentage accuracy of the algorithm at being capable of accurately predicting the number plate was calculated using Equation 1 below




$$\text{Overall accuracy (\%)} = \frac{\text{Correctly Detected number plates}}{27} * 100 \text{ (Equation 1)}$$




In order for the major steps of the algorithm to be deemed acceptable, a minimum of 65% accuracy was set as the benchmark. Should the algorithm produce results with lesser than 65% accuracy, the code was modified in terms of major operations such as morphological operations, contour detection steps, parameters to consider when filtering, cropping the image and darkening a fraction of the image. This was done to produce significant improvements in the algorithm's performance.




Once a 65% accuracy was achieved, the finer detailed parameters of the algorithm were further scrutinized to see if a higher accuracy can be achieved. This was done by adjusting intricate parameters such as criteria for filtering contours (areas, aspect ratios and heights and widths), iterations of morphological operations, kernels, adjusting the new coordinates of perspective transform image, implementing pytesseract on both perspective transform, and non-perspective transformed image and finally filtering out noises in the OCR results. All of these steps managed to improve the accuracy of the algorithm significantly.




5. RESULTS




The final algorithm managed to achieve a final accuracy of 77.78% by detecting 21 out of the 27 images tested. Figure 21 below presents the original image, result (perspective transform), the final_image (no perspective transform), result obtained (pass or fail).



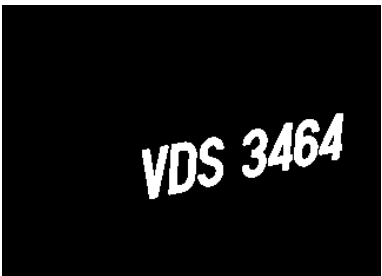
(a) Image 0		
		
Actual Plate: WXV1213	Detected Plate: WXV1213	Result: PASS




(b) Image 1		
		
Actual Plate: WYN9799	Detected Plate: WYN9799	Result: PASS




(c) Image 2		
		
Actual Plate: VFC8292	Detected Plate: VFC8292	Result: PASS




(d) Image 3		
		
Actual Plate: VFS4979	Detected Plate: VFS4979	Result: PASS




(e) Image 4		
		
Actual Plate: AHS65	Detected Plate: AHS&5	Result: FAIL




(f) Image 5		
		
Actual Plate: VDS3464	Detected Plate: VDS3464	Result: PASS


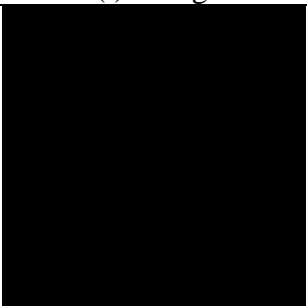
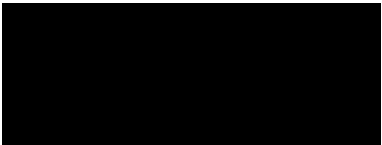
(g) Image 6		
		
Actual Plate: WHM2662	Detected Plate: WHM2662	Result: PASS




(h) Image 7		
		
Actual Plate: VAV6143	Detected Plate: VAV6143	Result: PASS

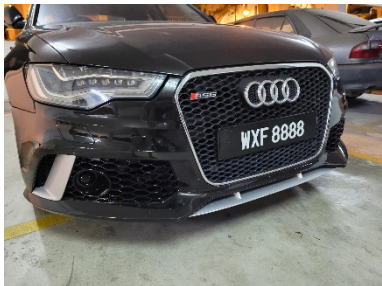


(i) Image 8		
		
Actual Plate: JSB1719	Detected Plate: JSB1719	Result: PASS




(j) Image 9		
		
Actual Plate: IQ370	Detected Plate: IQ370	Result: PASS


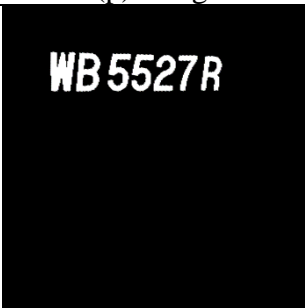

(k) Image 10		
		
Actual Plate: WB1524S	Detected Plate: WB1524S	Result: PASS




(l) Image 11		
		
Actual Plate: WA2698M	Detected Plate: NaN	Result: FAIL


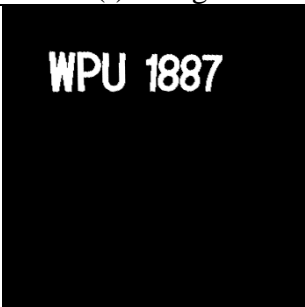

(m)Image 12		
		
Actual Plate: VAM8882	Detected Plate: VAM8882	Result: PASS


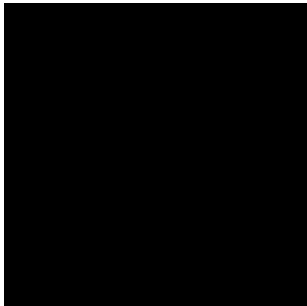
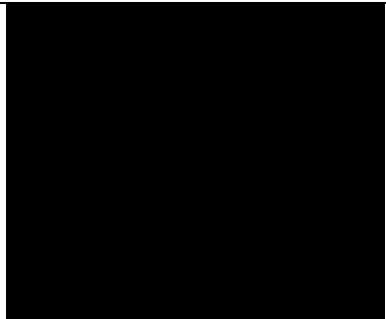
(n) Image 13		
		
Actual Plate: WXF8888	Detected Plate: WXF8888	Result: PASS


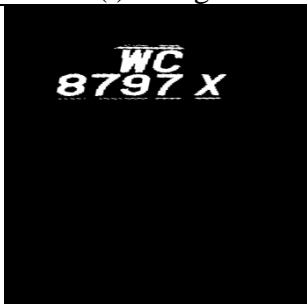

(o) Image 14		
		
Actual Plate: QBD20	Detected Plate: QBD20	Result: PASS




(p) Image 15		
		
Actual Plate: WB5527R	Detected Plate: WB5527R	Result: PASS




(q) Image 16		
		
Actual Plate: WNT8799	Detected Plate: —~ae-	Result: FAIL




(r) Image 17		
		
Actual Plate: WPU1887	Detected Plate: WPU1887	Result: PASS




(s) Image 18		
		
Actual Plate: WNQ4118	Detected Plate: NaN	Result: FAIL


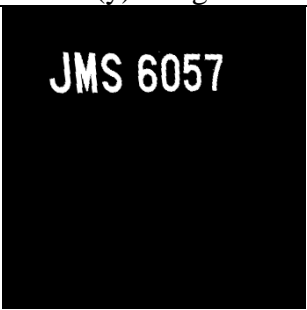

(t) Image 19		
		
Actual Plate: WC8797X	Detected Plate: @asx	Result: FAIL




(u) Image 20		
		
Actual Plate: VBL1001	Detected Plate: VBL1001	Result: PASS

(v) Image 21		
		
Actual Plate: QTT6688	Detected Plate: QTT6688	Result: PASS

(w) Image 22		
		
Actual Plate: VBT8524	Detected Plate: VBT8524	Result: PASS

(x) Image 23		
		
Actual Plate: WC543U	Detected Plate: LY543U	Result: FAIL

(y) Image 24		
		
Actual Plate: JMS6057	Detected Plate: JMS6057	Result: PASS

(z) Image 25		
		
Actual Plate: WTW1777	Detected Plate: WTW1777	Result: PASS




(aa) Image 26		
		
Actual Plate: JKG7751	Detected Plate: JKG7751	Result: PASS

Figure 21: (a-aa) Results produced by algorithm from testing.

6. CRITICAL ANALYSIS OF RESULTS

The algorithm managed to achieve satisfactory results, being able to take in an image of a vehicle entering a gate, filter the image, and segment its car number plate characters and recognize the number plate with an overall accuracy of 77.78%.

An important observation made was that a majority of the PASS (accurate plate detection) results came from the perspective transformed image (result) as compared to the non-perspective transformed image (final_image). This implied that implementing the perspective transform significantly improved pytesseract's capability in carrying out OCR. However, it is also important to note that not all of the perspective transformed images contributed to PASS results. Some PASS results were based on detection from the final_image without transforming the image. This can be seen in instances of Image 6, 20 and 26. This can be attributed to:

- 1) The filtering process where the algorithm could have produced more than 1 contour regions in mask5 which is overlayed on the image which is OCR-ed. The perspective transform step would have re-shifted the view of the wrong contour containing only partial or no information whatsoever of the plate. This would have led to inaccurate readings as only one of the contours would be perspective transformed and OCR-ed. However, combined together these contours would reveal sufficient regions in the non-transformed image for accurate OCR. Instances of this can be seen in cases of Image 20 and 26.
- 2) The OCR results from the perspective transformed image were inaccurate though the transformed image was clear and visible to the naked eye. This could be attributed to several factors such as the size, aspect ratio of the transformed image, the angle in which it was transformed and the clarity of the image. An example of this can be seen in Image 6.

On the other hand, the OCR produced satisfactory results from the non-transformed image which made it a viable output. This justifies having pytesseract performing OCR on both the transformed and non-transformed image and filtering the output as it helped in improving the performance of the algorithm. However, it is important to note that only a handful of the non-perspective transformed image produced accurate results which was why the primary result was taken from the perspective transformed image.

Looking at the 6 FAIL results, there are several reasons why the 6 out of the 27 images failed to be recognized:

- 1) Similar to the explanation above on perspective transformed vs non-perspective transformed image, the filtering process could have removed crucial regions with vital information in the image for OCR. The effect of filtering could have been much more extensive on these images where a majority of regions were removed and only minor insignificant regions with very little information were left in both the transformed and non-transformed image. Having perform OCR on both these images would have led to zero or very inaccurate results. These can be seen in Images 11, 18 and 23.
- 2) Another reason would be due to the orientation or view of the perspective transformed image that made it difficult for accurate OCR as explained above. Though the non-transformed was

also clear, the OCR however for these instances did not manage to produce any or produced inaccurate results. Examples of these are like Images 16 and 19.

- 3) A final observation made on the failed Image 4 showed that the results failed due to misinterpreted reading by pytesseract where for both transformed and non-transformed image produced similar results of AHS&5 instead of AHS65. The character “6” was misinterpreted as “&”. Instances of these appeared in other images as well but the locations of the misinterpreted text were at either ends of the image. This was the reason why filtering of the output was implemented. What distinguishes this image was that the misinterpreted character was located in the middle of the entire text output. The conditions that formed the basis for filtering either ends were less fuzzy and easily implemented. However, there were a lot of ambiguity in setting conditions for misinterpreted texts in the middle. Hence, it was not possible to correct this particular output.

It can be seen that the operation of filtering / contour detection was carried out in such a way that focused more on detecting / drawing smaller boundaries around the individual number plate characters first. The smaller contours were then processed, combined, and filtered further. This was done iteratively in multiple stages forming bigger contours until eventually a final contour around the number plate is detected that is shown in mask5 for a majority of the images. In essence, smaller contours were detected first and consolidated to form larger contours to fit the number plate.

This methodology was implemented as opposed to directly detecting the number plate itself and working on recognition directly due to 2 reasons:

- 1) It was observed during development that the rate at which the algorithm was capable of accurately focusing on the entire number plates was very low. Though it worked for some of the vehicle plates, its efficiency in doing so did not make it justifiable albeit its simplicity. There were several cases where the size of contour detected was too large covering unnecessary areas leading to noise. Another case is where the contours detected did not cover complete areas of the number plate and there were also cases where two different portions of the plate were covered with 2 different contours. It was noticed that this early method worked well vehicles with a silver border around their plates but poorly with vehicles without the border as shown in Figure 22. However, this parameter was a difficult element to control in vehicles. However, what was constant was the contrast between the characters and the black colored plate. Hence, this contrast was taken advantage of in defining the final algorithm.

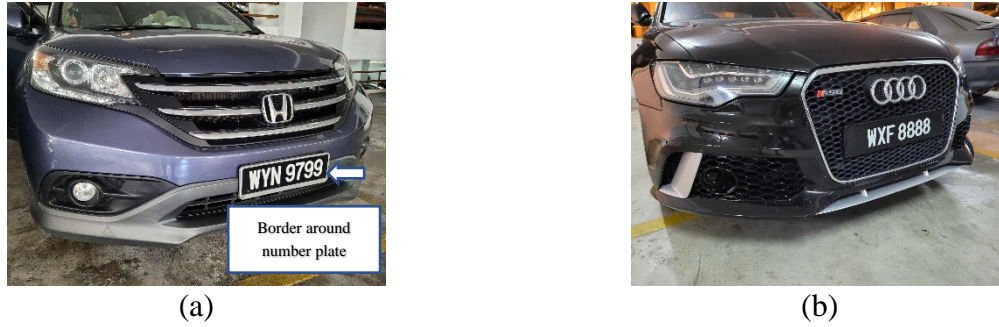


Figure 22: Comparison of plates with (a) Plate border, (b) Without plate border.

- 2) Another problem with detecting the entire number plate first instead of characters was that it would make it very difficult to filter the other unwanted rectangles (noise) in the image as they had very close parameters to the intended rectangle contour such as area, perimeter, and aspect ratios. Most of the noise rectangles would have overlapping parameters with the intended rectangle. By updating the algorithm to instead detect the characters first which have relatively smaller sized rectangles, this gave the room to first eliminate comparatively large and very small noise rectangles. It also provided the algorithm with an extra parameter to utilize which was the centroid distance of each rectangle. This parameter was a very useful parameter to further filter out noise rectangles that were located very far away from other rectangles as characters in the number plate are relatively close to one another. The remaining rectangles were then combined together to form larger rectangles by putting the mask through another level of contour detection. As the characters were very closely grouped to one another, the resultant combined rectangle would be relatively larger as compared to others by area, it would have a more specific height and aspect ratio as it would most likely be horizontal. It in essence would have more distinguishable parameters as compared to noise rectangles making it easier to filter and crop out the number plate only for perspective transform and OCR.

A point to note was that the method of filtering the contours used `cv2.minAreaRect()` instead of `cv2.approxPolyDP()` in which rectangle boundaries are drawn over the contours instead of a polygon. The rectangle boundary approach was selected due to consistency in its number of 4 sides as opposed to polygons which would result in multiple geometries with different number of sides being drawn on the mask. Having the consistent number of sides provided more control for filtering the contours and also made it easier for perspective transform.

Another point to note is that the algorithm works well under the controlled angle / view in which the image was taken. It is also assumed that ambient light intensity is a controlled variable and that the image should be taken indoors. Should the environment or view of the image be manipulated significantly, it would affect the algorithm's performance.

7. CONCLUSION AND FUTURE RECOMMENDATIONS

The rapid development and increasing applications of Artificial Intelligence, specifically the domain of Image Processing and Computer Vision has been growing exponentially over these recent years. The project goal is to develop an algorithm for license plate character segmentation and recognition for an ANPR system detecting number plates from images of cars passing through a gate.

The license plate character segmentation and recognition algorithm for an ANPR system was successfully developed. The algorithm works in such a way that it takes in an image of a vehicle at a particular view angle, process and filter the image to segment out only the characters and number plate of the vehicle, perform OCR on the processed image and post OCR data processing in order to finalize the output of the algorithm which is the detected number plate. The proposed algorithm was able to perform well, achieving an overall accuracy of 77.78%. The problem statement, objectives, methodology and justification of algorithm, results, and critical analysis of the results were presented and discussed. In overall, the algorithm managed to achieve its project objectives.

However, there are still some limitations to the capability of the algorithm where room for improvement is seen in its development to improve its overall performance.

- 1) Firstly, the algorithm's process flow or minor detailed parameters can be tweaked to refine the image processing methods to overcome the problems faced in the failed images as discussed in Section 6.
- 2) The image used in this current system is a single framed image. Further development would involve taking in multiple frames of images of the vehicle from a video camera source and feeding them into the algorithm for processing and validating the results from the multiple frames before producing the final output.
- 3) The computational time of the algorithm is also considerably slow. Improvements can be made by using better hardware or modifying the algorithm to reduce its computational complexity.

Given additional time, hardware and computing resources, the mentioned limitations can certainly be overcome to improve the algorithm.

8. REFERENCES

- [1] Williams, J. et. al. (2019) The oil and gas sector faces challenges on multiple fronts. Embracing AI may provide a way to address them., EY. Available at: https://www.ey.com/en_bh/oil-gas/is-ai-the-fuel-oil-and-gas-needs (Accessed: 14 October 2020).
- [2] <https://www.idc.com/getdoc.jsp?containerId=prUS46794720#:~:text=FRAMINGHAM%2C%20Mass.%2C%20August%2025,than%20%24110%20billion%20in%202024.>
- [3] JOSHI, N. (2019) *Understanding the difference between computer vision and image processing*, Allerin. Available at: <https://www.allerin.com/blog/understanding-the-difference-between-computer-vision-and-image-processing> (Accessed: 4 April 2021).
- [4] Bezdek, J. C. et al. (1999) 'Image Processing and Computer Vision', in *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*. Boston, MA: Springer US, pp. 547–678. doi: 10.1007/0-387-24579-0_5.
- [5] Faggella, D. (2020) *Computer Vision Applications – Shopping, Driving and More*, EMERJ. Available at: <https://emerj.com/ai-sector-overviews/computer-vision-applications-shopping-driving-and-more/> (Accessed: 4 April 2021).
- [6] Greensted, D. A. (2010) *Otsu Thresholding*, *The Lab Book Pages*. Available at: <http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html#:~:text=Otsu's thresholding method involves iterating,fall in foreground or background.&text=This final value is the,for the threshold value 3.> (Accessed: 10 April 2021).

9. APPENDICES

No	File Description	File Attachment
1	Python Code “Final_Part_2”	
2	Raw Images Dataset	