Part I

Literature Review

In a car plate recognition study carried out by Zhai, Benssali and Ramalingam (2010), they used mathematical morphology to pre-process the images to extract certain features out of the car plate. They performed a series of morphological operations on a grayscale image, which successfully highlights the region of the car plate into a rectangle and can be easily extracted using certain geometrical conditions. However, we noticed that in the image they used, the car plate had a white background and black characters, as opposed to our case, in which Malaysia car plates have black background and white characters.

After some trial and error, we clarified our doubts that this proposed image processing method does not work effectively on Malaysia car plates as the car plate is also black in colour, which may be accidentally fused with the background, and thus losing the highlighted region. However, we took note of a useful formula in calculating the threshold used for image binarization.

Based on the research by Soon, Lin, Jeng and Suandi (2012), they used the Connected Component Analysis (CCA) method as a part of the car plate detection process since it is useful for removing noise and unwanted areas. CCA labels binary images into several components based on their connectivity. The criterion to determine whether pixels are connected is based on 4-adjacency or 8-adjacency of pixel connectivity. They also stated that CCA works with spatial measurement and filtering gives a good result in number plate detection. We decided to use this method in the assignment to remove partial noise and area which we dont want by tweaking the threshold of the height, weight, area of the connected component to retain the areas which are possible to be the car plate.

In a tutorial released by Rosebrock (2021), he demonstrated how to use connected component labelling and analysis from an inbuilt function in the Python cv2 library called cv2.connectedComponentsWithStats. This function will let us get all connected components with their state recorded which contains x coordinates, y coordinates, width, height, area and others. With this technique we can easily tweak the threshold of the state to keep only the wanted parts.

Shi, Zhao and Shen (2005) used Colour Image Processing techniques to perform an analysis over the image, then proceeded to extract the car plate region based on the histogram projection of a colour exceeding a certain threshold. They made use of the information that the colour combination of the car plate and the characters is unique and occurs mostly only in the plate region. They started off with a vertical frequency analysis of the car plate, assuming that the background colour is blue which is most commonly used in mainland China.

Figure 1: Vertical Frequency Analysis (Shi et al. 2005)

Later on, they extracted the plate region that exceeds the frequency threshold, and performed a horizontal frequency analysis on that region to determine the location of the car plate. Finally, they confirmed the extraction by examining the Width to Height Ratio (WHR) of the region. If it is sufficiently close to some threshold value, the region is considered to be a potential car plate.

Fig. 2. Extraction of a plate region in horizontal

Fig. 3. The extracted plate region

Figure 2: Horizontal Frequency Analysis and Region Selection (Shi et al. 2005)

A similar problem arises in which this method is not really effective with Malaysia car plates as the background colour of the car plate blends in with the front grille of the car, making it difficult to perform a colour frequency analysis. Nevertheless, we still took inspiration from this method and devised our own algorithm, which uses a similar concept.

Task Allocation:

All three of us did separate literature reviews and assimilated our findings. After that, Hong Bo and Chen Xi worked together to devise the algorithm and write the report for the CCA and Histogram Projection method. Hong Bo worked on the CCA method to preprocess the image, whereas Chen Xi devised the algorithm for Histogram Projection to extract the car plate region.

Part II

Overall methodology:

First, the image is resized to decrease the computation load and increase the efficiency of our algorithms. Then the shrunken image is converted to grayscale. This is to reduce the number of channels from 3 (R, G and B) to 1 (Intensity) and effectively tone down the complexity of the image. After that, the image is converted into a binary image using Otsu's Thresholding Method. We calculated the threshold with the formula:

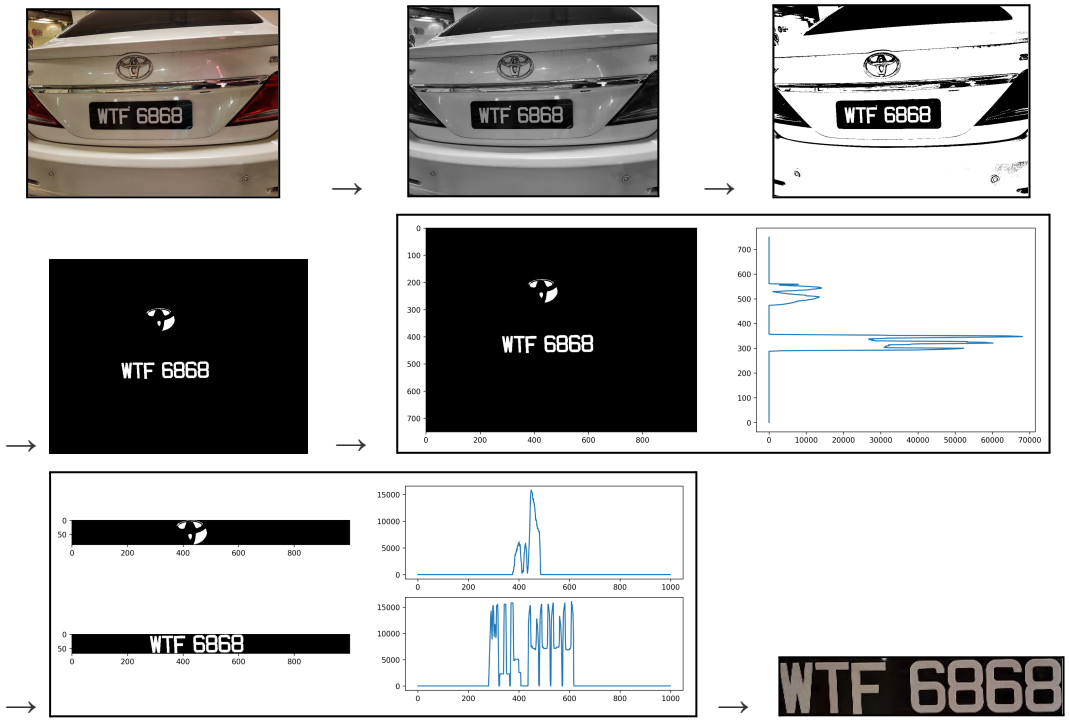
$$T = (max\_px - (max\_px - min\_px)/1.35)/255$$

Where max_px and min_px correspond to the maximum intensity pixel and the minimum intensity pixel in the grayscale image respectively.

After binarising the image, we applied the CCA method to filter out non-plate regions and keep the connected components in the image.

We then performed a horizontal projection analysis on the resulting image to determine possible regions of the car plate. Since we have removed the majority of the unwanted pixels, we can easily obtain horizontal segments as potential candidates. After that, we performed a vertical projection analysis on each of the horizontal segments. We made use of the information that, mostly only in the car plate region, we will observe more troughs that hit 0 (i.e. There is an entirely black region). Therefore we will filter out regions that have few troughs and select the candidate that has the most troughs. We also performed a WHR check to validate the corresponding region. Finally, the best candidate is selected and saved.

The series of photos show the results of each step in order.



Description of developed algorithms:

```
#Converting image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#Calculating threshold value for binarization
max_val, min_val = np.max(gray), np.min(gray)
T = (max_val - (max_val - min_val)/1.35)//255

#Applying Otsu's thresholding method to binarize the image
thresh = cv2.threshold(gray, T, 255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
thresh = cv2.bitwise_not(thresh, thresh)
```

In this code snippet, we converted the image to grayscale, then performed Otsu's thresholding method to binarize the image with the threshold we obtained from the formula.

```
#Connected Components Analysis
(numLabels, labels, stats, centroids) = cv2.connectedComponentsWithStats(thresh,4,cv2.CV_32S)
mask = np.zeros(gray.shape, dtype="uint8")
```

In this code snippet, we use the function connectedComponentsWithStates which captures the connected component and bound it in a rectangle from the binarized image, and checks the 4 adjacencies of the pixel to check if they are connected. The function will return 4 values which are the total number of labels, the list of labels, the corresponding state of the label, and the centroids of the labeled boundary.
The mask is creating a wholly black image that has the same dimensions as the resized image for further use.

```
# extract the connected component statistics and centroid for the current label
x = stats[i, cv2.CC_STAT_LEFT]
y = stats[i, cv2.CC_STAT_TOP]
w = stats[i, cv2.CC_STAT_WIDTH]
h = stats[i, cv2.CC_STAT_HEIGHT]
area = stats[i, cv2.CC_STAT_AREA]
```

Then we get the state of the components to filter the unwanted parts of the image.

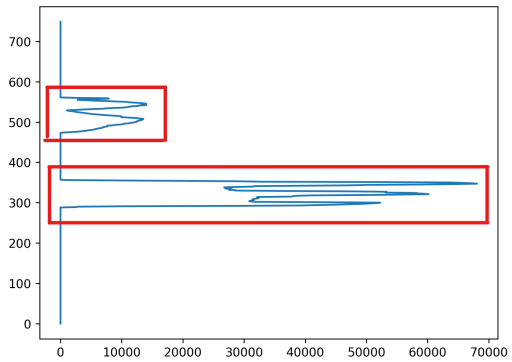
```
# ensure the width, height, and area are all neither too small nor too big
keepArea = area>40 and area < 6100 \
        and h>12 and w>3 and h<150 and w<100 and x>130 \
        and x<mask.shape[1]-100 and y>20 and y<mask.shape[0]-150
# construct a mask for the current connected component by finding a pixels in the labels array that have the current connected component ID
if keepArea:
    lst_state.append((x,y,h))
    # construct a mask for the current connected component and then take the bitwise OR with the mask
    componentMask = (labels == i).astype("uint8") * 255
    mask = cv2.bitwise_xor(mask, componentMask)
```

By keep tweaking and testing the threshold to filter out the part we don't want (eg. height, weight, and area.) Then we will keep the value that passes the threshold and make the components white and bitwise_xor and remain the component in the mask and that will lead us to a filtered image.

```
"""
Segmenting possible car plate areas using Horizontal projection
"""
#Summing up the pixels in each row of the mask
horizontal=np.sum(mask,axis=1)
is_start = False
horizontal_block = []
start_x = 0
end_x = 0
i = 0
while i < len(horizontal):
    if not is_start and horizontal[i]>0:
        is_start = True
        start_x = i
    elif is_start and horizontal[i]==0:
        is_start = False
        end_x = i
        horizontal_block.append((start_x,end_x))
    i+=1
```

Then we calculated the horizontal projection of the image obtained from the CCA method. Here we segmented the horizontal regions by making use of the projection. We iterate over the summations of each row, and we include rows in the same segment if they have values greater than 0.

For example in this horizontal projection, we can obtain 2 segments as marked in red boxes below:




```
#Segments of the image obtained from horizontal projection
horizontal_segments=[]
#Coordinates of the segments
horizontal_seg_coors=[]

for block in horizontal_block:
    segment = mask[block[0]:block[1],:]
    horizontal_segments.append(segment)
    horizontal_seg_coors.append((block[0],block[1]))
```

In the next block of code, we saved the segments of the image into a list, and the coordinates of that segment into another list for further analysis.

We then performed vertical projection on all the segments obtained and proceeded to segment the horizontal segments vertically based on the vertical projection, with a similar logic as the horizontal segment.

```
def join_blocks(block):
    """
    Joins vertical blocks that are close to each other
    """
    t = 90
    res = []
    i = 0
    start = block[0][0]
    end = block[0][1]
    #Keeps track of the number of troughs in the current detected interval
    count = 0
    while i < len(block)-1:
        if (block[i+1][0] - end) < t:
            end = block[i+1][1]
            count+=1
        else:
            res.append((count,start,end))
            start = block[i+1][0]
            end = block[i+1][1]
            count = 0
        i+=1
    res.append((count,start,end))
    return res
```

Due to the fact that car plate characters are usually close to one another, we devised a function called join_blocks, which allows us to join vertical segments that are close to each other into one block. We also kept track of the number of troughs in that block when joining two blocks together to be used later during the filtering process.

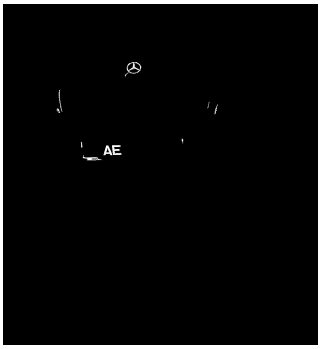
```
if len(sorted_blck) > 0:
    for block in sorted_blck:
        #If we have not found a possible candidate yet, or the current segment has more troughs than the
        if possible_car_plate == None \
            or (possible_car_plate[0][0] < block[0] and block[0] < 10 and 1 < block[2]/block[1] < 3.5):
            possible_car_plate = (block,horizontal_seg_coors[ver])
```

The filtering process consists of us iterating through every vertical block and checking for two main features of the block. The first one being the number of troughs detected in the block. If the number of troughs is greater, the more likely it is going to be a car plate, however if the number of troughs exceed a certain value (e.g. 10), its probability of being a car plate drops drastically. The second is the WHR of the block. If the WHR exceeds a certain range, it is not considered to be a car plate.

```
localised = img[possible_car_plate[1][0]:possible_car_plate[1][1],possible_car_plate[0][1]:possible_car_plate[0][2]]
cv2.imwrite("A2/locs/" + filename, localised)
```

From our series of steps above, we can crop the image based on the final coordinates we have that correspond to the best candidate region. The result is then saved as an image.

Experimental Results:

Category	Sample	Set 1	Set 2
Car plate info retained		44	12
Car plate info fully or partially lost		1	3
Total		45	15
Correctly identified		32	9
Incorrectly identified		13	6
Total		45	15

For the image preprocessing section, we managed to obtain 44/45 ≈ 97.78% of images successfully preprocessed without affecting the characters of the car plate in set 1, and 12/15 ≈ 80% of successfully preprocessed images. By using the preprocessed images for our car plate localization algorithm, we successfully localised 32 images in set 1 and 9 images in set 2. The accuracy of our algorithm in set 1 is ≈ 71.11%, and ≈ 60% in set 2.

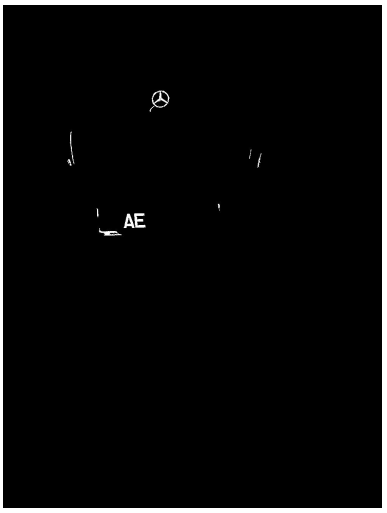
Critical Analysis and Discussion:

We will perform an analysis and discussion of image preprocessing and localisation of the car plate separately.

Image Preprocessing

For the preprocessing we can observe that the characters of the car plate are affected by some external factors such as a broken car plate, the reflection of sunlight on the car plate, and blurry images) which will be filtered out in the preprocessing.

Img 5 Original Image



Img 6 Preprocessed Image

Car Plate Localisation

We think that the reason for the occurrence of the failed cases of car plate localisation may be due regions with similar features as the car plate, such as car grilles and windshield reflections. The horizontal projection and vertical projection in our algorithm is prone to such errors to mistakenly identify such regions as potential car plates. A possible improvement to the algorithm is to perform feature extraction and comparison using the SIFT algorithm to make the detection system more robust and more accurate.

Img 7. CCA Mask of the image

Img 8. Localised region

References

X. Zhai, F. Benssali and S. Ramalingam, "License plate localisation based on morphological operations," *2010 11th International Conference on Control Automation Robotics & Vision*, Singapore, 2010, pp. 1128-1132
License plate localisation based on morphological operations | IEEE Conference Publication | IEEE Xplore

Soon, Choo & Lin, Kueh & Jeng, Chung & Suandi, Shahrel Azmin. (2012). Malaysian Car Number Plate Detection and Recognition System. 6.
https://www.researchgate.net/publication/264888086_Malaysian_Car_Number_Plate_Detection_and_Recognition_System

Rosebrock A., OpenCV Connected Component Labeling and Analysis, February 22,2021:
https://pyimagesearch.com/2021/02/22/opencv-connected-component-labeling-and-analysis/

Shi, X., Zhao, W., Shen, Y. (2005). Automatic License Plate Recognition System Based on Color Image Processing. In: , *et al.* Computational Science and Its Applications – ICCSA 2005. ICCSA 2005. Lecture Notes in Computer Science, vol 3483. Springer, Berlin, Heidelberg.
Automatic License Plate Recognition System Based on Color Image Processing | SpringerLink