MONASH University
Information Technology

GROUP ASSIGNMENT COVER SHEET

Student ID Number	Surname	Given Names
32286988	Kuah	Jia Chen
32156944	Ng	Kai Yi
32025068	Tai	Samuel Meng Yao
31110347	Chong	Wilbert

* Please include the names of all other group members.

Unit name and code	Image Processing FIT 3081		
Title of assignment	Assignment 2		
Lecturer/tutor	Dr. Raveedran Paramesran		
Tutorial day and time	Tuesday 12pm - 2pm	**Campus**	Malaysia Campus
Is this an authorised group assignment?	☑ **Yes** ☐ **No**		
Has any part of this assignment been previously submitted as part of another unit/course?	☐ **Yes** ☑ **No**		
Due Date	2/5/2023	**Date submitted**	2/5/2023

All work must be submitted by the due date. If an extension of work is granted this must be specified with the signature of the lecturer/tutor.

Extension granted until (date) ..... Signature of lecturer/tutor .....

Please note that it is your responsibility to retain copies of your assessments.

Intentional plagiarism or collusion amounts to cheating under Part 7 of the Monash University (Council) Regulations

Plagiarism: Plagiarism means taking and using another person's ideas or manner of expressing them and passing them off as one's own. For example, by failing to give appropriate acknowledgement. The material used can be from any source (staff, students or the internet, published and unpublished works).

Collusion: Collusion means unauthorised collaboration with another person on assessable written, oral or practical work and includes paying another person to complete all or part of the work.

Where there are reasonable grounds for believing that intentional plagiarism or collusion has occurred, this will be reported to the Associate Dean (Education) or delegate, who may disallow the work concerned by prohibiting assessment or refer the matter to the Faculty Discipline Panel for a hearing.

Student Statement:

- I have read the university's Student Academic Integrity Policy and Procedures.
- I understand the consequences of engaging in plagiarism and collusion as described in Part 7 of the Monash University (Council) Regulations http://adm.monash.edu/legal/legislation/statutes
- I have taken proper care to safeguard this work and made all reasonable efforts to ensure it could not be copied.
- No part of this assignment has been previously submitted as part of another unit/course.
- I acknowledge and agree that the assessor of this assignment may for the purposes of assessment, reproduce the assignment and:
  - i. provide to another member of faculty and any external marker; and/or
  - ii. submit it to a text matching software; and/or
  - iii. submit it to a text matching software which may then retain a copy of the assignment on its database for the purpose of future plagiarism checking.
- I certify that I have not plagiarised the work of others or participated in unauthorised collaboration when preparing this assignment.

Signature ..... *Date*.....

* delete (iii) if not applicable

Signature _____ Date: 2/5/2023 Signature _____ Date: 2/5/2023

Signature _____ Date: 2/5/2023 Signature _____ Date: _____

Signature _____ Date: 2/5/2023 Signature _____ Date: _____

Privacy Statement
The information on this form is collected for the primary purpose of assessing your assignment and ensuring the academic integrity requirements of the University are met. Other purposes of collection include recording your plagiarism and collusion declaration, attending to course and administrative matters and statistical analyses. If you choose not to complete all the questions on this form it may not be possible for Monash University to assess your assignment. You have a right to access personal information that Monash University holds about you, subject to any exceptions in relevant legislation. If you wish to seek access to your personal information or inquire about the handling of your personal information, please contact the University Privacy Officer: privacyofficer@adm.monash.edu.au

Part 1

Introduction

The literature review of the methods used in locating the car license plate will be presented in this section.

Methods used in locating the car license plate

The process of car license plate detection involves several image processing techniques, including edge detection and projection profiles. An essential step that results in an edge image used for further processing is edge detection. After edge detection, regions with high edge values can be searched for using horizontal and vertical operations, potentially revealing the location of the vehicle's license plate. An important tool in this process is the projection profile, which is a condensed depiction of the spatial pixel content distribution. (Khalifa et al., 2007) The resulting vector of the sums of the pixel intensities over each row, also known as the horizontal projection profile, is particularly helpful for locating areas with sharp edges.

The extraction of potential horizontal places for the license plate is made possible by graph representations of horizontal projections, like those in Figure 3. The method then signifies horizontal segments that correspond to significant peaks in the edge image's projection profile, indicating where the license plate is located. The precision of the edge detection method, such as the Canny algorithm, and the quality of the source image both play a significant role in how well these techniques work, despite their utility.

Figure 1: Original Image

Figure 2: Edge Image after Canny

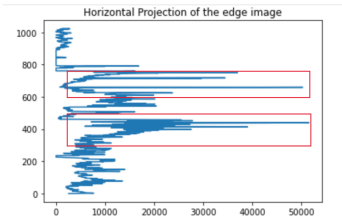

Figure 3: Horizontal Projection

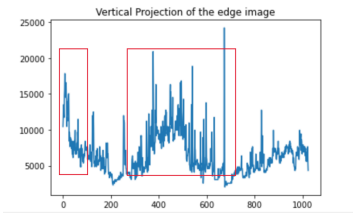

Figure 4: Vertical Projection

In their work, Khalifa et al. (2007) outline a straightforward technique for determining the vertical coordinates of the vehicle license plate using the edge image. The approach slides a window through each horizontal section of the image to determine the average gradient magnitude per pixel inside the window at a particular location. To do this, add up the inner edge values of the window, then divide the result by the window's area. The approach builds a new vector called the horizontal density vector (HDV) to retain the values acquired for each window step for each horizontal segment. In Figure 5, the HDV formula is shown, where G(x, y) is the edge image produced by the Canny operator, W and Aw are the width and area of the sliding window, respectively, M is the width of the window, and ImW is the width of the image. The i is from 0 to (ImW - M + 1).

The algorithm then searches for peaks in each horizontal axis of the HDV graph; if a peak is found and it exceeds a certain threshold, a license plate is thought to be present. During the vertical projection operation, the same process is used to determine the license plate's vertical position, and HDV graphs are generated for each horizontal segment. The algorithm eventually determined the contour of the vehicle license plate from the image.

$$HDV(i) = \frac{\sum_{x \in W} \sum_{y \in W} G(x, y)}{A_w}$$

Figure 5: HDV Formula

Other than the technique mentioned above, contours can also be used to locate a car license plate. According to Assaialangaram (2020), the first step is to use the `findContours()` function from `cv2` to locate all the contours in the image. These contours can then be sorted by their area in descending order using a sort function, and the top 10 possible contours can be extracted. The license plate is usually the largest object in the image and is enclosed by a contour. By filtering out small contours that are unlikely to correspond to the license plate, we increase the likelihood of finding the license plate.

After extracting the possible contours, further validation is necessary to select the best possible license plate contour. Sharma et al. (2022) and Assaialangaram (2020) propose using OpenCV2 functions, including the `approxPolyDp()` function, to approximate the shape of the contours to another shape. The output is a list with the length equal to the number of vertices of the approximate shape. This output can be used to identify whether the contour represents a possible license plate by comparing its length with 4, assuming that a license plate is a rectangle. If the contour has 4 corners, we consider it as the license plate and exit the loop.

In addition, license plate region verification can also be done through determining the aspect ratios and vertical edges (Taewan & Eungtae, 2017). They bring up the idea that the aspect ratio of the plate is fixed, which is between 2 to 5 assuming we have a fixed size of the number plate. After the candidate region is determined using the aspect ratio feature, those regions are selected from the original image, calculate the vertical edges using vertical Sobel mask and project them onto the vertical plane to calculate the edge distribution. This is because the characters and letters in a real plate usually have strong vertical edges and they will concentrate in the middle of the plate, whereas in other regions, the vertical edges might exist randomly. Subsequently, license plate region can be identified by calculating the sum of the normalized edge histogram values multiplied by a weighting coefficient value which is calculated using equation in Figure 6. This process is used to locate the higher edge value in the center of the vertical plane, which corresponds to the concentration of vertical edges in the license plate region.

$$coefficient : f(x) \begin{cases} x \div \frac{\mathrm{row_{total}}}{2} & \left(\text{if} \quad x \leq \frac{row_{total}}{2}\right) \\ (\mathrm{row_{total}} - x) \div \frac{\mathrm{row_{total}}}{2} & \left(\text{if} \quad x > \frac{row_{total}}{2}\right) \end{cases}$$

Figure 6: coefficient equation

Overall, we implemented two car license plate detection algorithms - one using horizontal and vertical projection operations and the other using the `findContours()` function. However, the horizontal and vertical projection operation approach took a considerable amount of time to run and produced very low accuracy, whereas the `findContours()` approach resulted in much better results in terms of finding the contours of the car license plate. Therefore, we decided to use the `findContours()` function as the primary approach and included the manual code that uses the horizontal and vertical projection operation approach in the Python notebook for Algorithm 1.

Task allocation

Task	Contributor
Preparation of Image Dataset	All
Python code - algorithm 1	Kuah Jia Chen, Samuel Tai Meng Yao
Python code - algorithm 2	Ng Kai Yi, Wilbert Chong
Report Part 1a & 1b	All
Report Part 2 - algorithm 1	Kuah Jia Chen, Samuel Tai Meng Yao
Report Part 2 - algorithm 2	Ng Kai Yi, Wilbert Chong

Part 2

Algorithm 1 - Lucid chart link

i) The overall methodology. Include a flow chart and a short description.

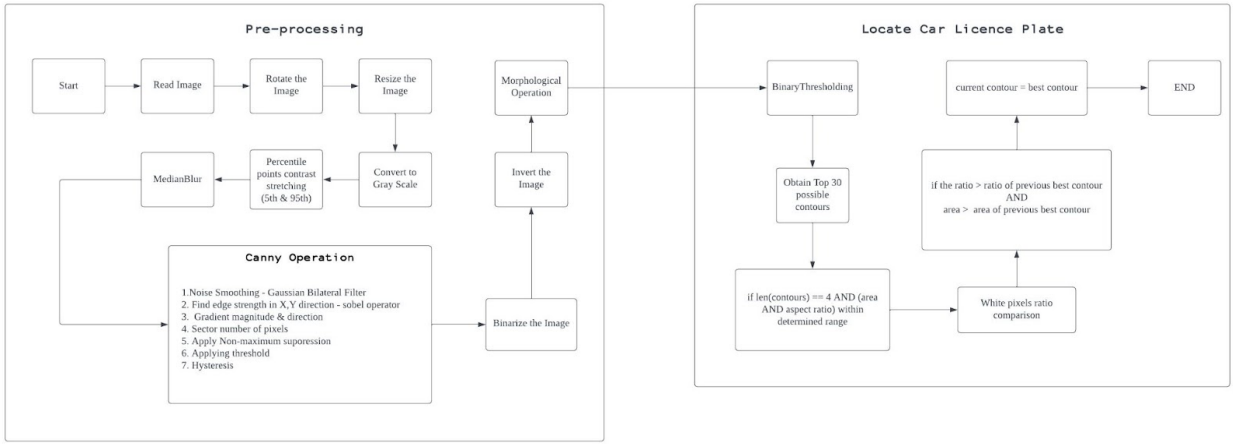

Figure 5: overall methodology for Algorithm 1

This section provides an overview of the methodology used to preprocess the image and locate the car license plate. The process begins by reading the image, followed by image rotation and resizing. Next, the image is converted to grayscale and subjected to percentile points contrast stretching, median blur, and bilateral filtering. The Canny operator is then applied to obtain the edge image, which is thresholded and processed using the `findContours()` function to obtain the top 30 potential contours of the car license plate. Each potential contour is assessed for having 4 corners, and its area and aspect ratio are checked against predetermined ranges. The white pixel ratio within the contours is also evaluated since the number plate characters are represented by white pixels in the edge image. If the white pixel ratio and area exceed those of the previously selected best contour,

the current contour becomes the new best contour. This process is repeated for all potential contours, and the most likely contour containing the car license plate is identified.

ii) Detailed description of your developed algorithms and the Python functions used in each step.

The following paragraph describes the pre-processing steps involved in a license plate detection algorithm mainly implemented using `OpenCV` and `Pillow` libraries. First, the image is read using the `Image.open()` function from the `PIL` library. If necessary, the image is rotated to an upright position by checking its metadata, and then resized to 1024x1024 size. This step helps to standardize the image orientation and improve processing speed. The image is then converted to grayscale using the custom function `convert_to_grey_scale()`, which simplifies the image processing task and makes it easier to identify important features. Next, contrast stretching is performed using the 5th and 95th percentile points with the function `contrast_stretching()`, which improves the contrast of the image. To remove any noise in the image, noise smoothing is applied using `medianBlur()` function from `cv2` library with a kernel size of 5. The edge detection process is then performed on the image using a manually coded Canny detector. Within the Canny function, noise reduction is performed using `cv2.bilateralFilter()` with d=13, sigma_color=15, and sigma_space=15 parameters, which is better than a Gaussian filter as it keeps edges sharp while removing noise.

The Sobel operator is used to find edge strength in both the X and Y directions, followed by gradient calculations, non-maximum suppression, double thresholding, and hysteresis. The threshold values used here are 60 and 200, which have been found by us to provide a good balance between sensitivity and specificity after experimenting. Although a manual Canny operation has been coded, the built-in version `cv2.Canny()` is used in the final implementation due to better performance in testing. Next, we will perform thresholding to binarize the image and enhance its features using `cv2.threshold()` with the parameters of source=edged_img, thresholdValue=0, maxVal=255 and thresholdingTechnique= cv2.THRESH_BINARY+cv2.THRESH_OTSU. The reason we used the combination of binary and Otsu's thresholding method is because it can be a more robust approach when the foreground and background of the image are not well separated which is common in our testing dataset where the license plate's color is same as the car color or the shadow of the car that affects the identification of the boundaries of license plate.

Then, we created a morphological kernel of size (3,3) using function `np.ones()` which will be used in the upcoming morphological operation. The binarized image is then inverted by using `cv2.bitwise_not()` as the morphological operation on the next step works better on white objects on a black background. The last step of the pre-processing stage is applying the morphological gradient operation to the inverted binarized image using `cv2.morphologyEx()` function with parameters of src=inverted binarized image, op= cv2.MORPH_GRADIENT, kernel= kernel created earlier. This step is to emphasize the boundaries of the regions in the image which can help in locating the license plate.

To prepare the image for contour detection, we'll apply binary thresholding to the pre-processed image using `cv2.threshold()`, with the parameters source=morpho_img, thresholdValue=0, maxVal=255, and thresholdingTechnique=cv2.THRESH_BINARY. Due to the fact that the `findContours()` function needs a binary image as input in order to recognise and extract the contours of objects in the image, this is required. Next, we'll use the `findContours()` function with the input parameters src=binarize_img, contour_retrieval=cv2.RETR_LIST, and contours_approximation= cv2.CHAIN_APPROX_SIMPLE. The cv2.RETR_LIST parameter ensures that all the retrieved contours are without any hierarchical relationships, while the

cv2.CHAIN_APPROX_SIMPLE parameter compresses horizontal, vertical, and diagonal segments and leaves only their end points. For instance, an up-right rectangular contour is encoded with only four points. This function allows us to get all the possible contours from our pre-processed image. Finally, we'll sort the possible contours based on their area and extract only the top 30 contours.

Next, we will use the `cv2.arcLength()` function to compute the perimeter of each possible contour. Then, we will approximate the contour shape by utilizing the `cv2.approxPolyDP()` function, with the input parameters cnt=contour, epsilon=0.02 * perimeter, and closed=True. Here, epsilon signifies the maximum distance between the contour and the approximated contour, and closed=True denotes that the approximated contour is closed. Subsequently, we will obtain the coordinates, width, and height of a bounding rectangle enclosing the contour by utilizing the `cv2.boundingRect()` function, which takes the contour as an input parameter. Using the width and height, we can calculate the aspect ratio and area of the contour by utilizing the formula float(width)/height and width * height, respectively. If the contour has four corners and its aspect ratio and area lie within the determined range, we will calculate its bounding box using the `cv2.boundRect()` function. This function returns the coordinates of a rectangle that covers the entire contour. Next, we will compute the white pixel ratio of the contour (i.e., the number of white pixels divided by the total number of pixels in the contour). We will then compare the white pixel ratio and area of this contour to those of the previously selected contour. If the white pixel ratio and area of the current contour are better than those of the previous contour, we will select this contour as the best contour so far. This process will be repeated for each possible contour, and ultimately, the best contour, which most likely contains the number plate, will be chosen.

iii) Detailed description of the experimental results.

Dataset	Number of Image	Accuracy
1	45	19/45
2	15	4/15
3	10	1/10

Figure 6: Accuracy table of built in Canny Edge detection

Dataset	Number of Image	Accuracy
1	45	0/45
2	15	0/15
3	10	1/10

Figure 7: Accuracy table of built in Customized Edge detection

The results from Figure 6 indicate that the accuracy of the built-in Canny Edge detection algorithm for detecting car license plates is only 34.286%, which clearly shows that there is room for improvement in the developed algorithm to achieve higher accuracy. One of the reasons for the low accuracy is that the preprocessing methods used are not effective enough to extract all the edges from the original images. Therefore, there is a need to improve the preprocessing pipeline. Additionally, during the validation process, possible contours are determined based on whether they are rectangular and their area and aspect ratio fall within a certain range. However, this approach may not be effective in cases where the license plate is not always in a rectangular shape or when images are taken at different angles. Furthermore, the acceptable range of area and aspect ratio may not accurately eliminate incorrect contours, leading to inaccurate license plate detection.

The results obtained from the algorithm that uses Customized Edge detection are shown in Figure 7. The accuracy achieved was merely 1.429%, which is considerably lower than the algorithm that employs the built-in Canny Edge detection method. This suggests that the customized approach is not appropriate for accurately identifying all the car plates in the dataset, which is due to its lower effectiveness compared to the built-in method. The following section shows an example.

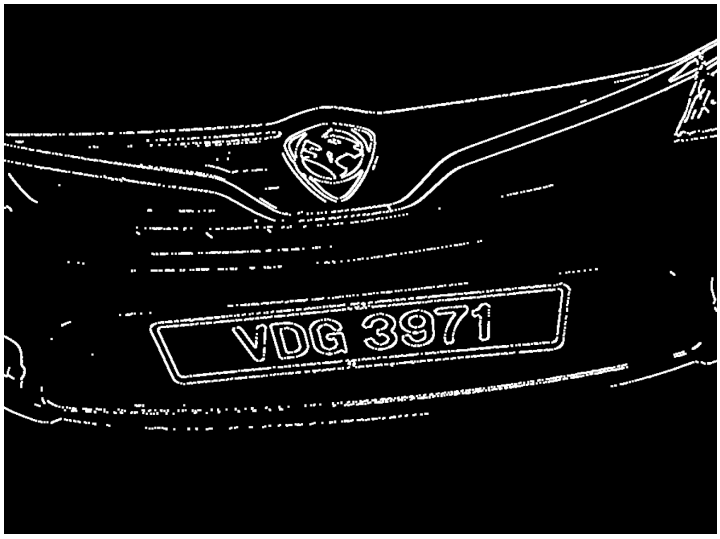

Figure 8: Customized Canny Edge Detection

Figure 9: Built-in Canny Edge Detection

Figure 10: Customized Canny Edge Detection

Figure 11: Built-in Canny Edge Detection

Figures 8 and 9 illustrate the edged images generated by the customized and built-in Canny edge detection algorithms, respectively, after passing through the entire preprocessing pipeline for a specific image. Comparing the two figures, we can observe that the customized Canny algorithm missed many edge points, resulting in the inability to locate the correct contour for the car license plate (Figure 10). Conversely, the built-in Canny algorithm performed much better, with edges being detected more accurately (Figure 9). As a result, Figure 11 shows that the correct contour for the car license plate was accurately identified.

iv) Discussion / critical evaluation on the outcomes and any observations

Upon initial analysis of the overall results, we have decided to conduct a more thorough examination of the data by dividing it into various groups. This deeper analysis will provide us with a better understanding of the shortcomings and vulnerabilities of our algorithm in detecting car license plates.

The following table presents the accuracy of car license plate detection for different image categories.

Category	Sub-category	Number of plate detected / Total plate in the sub-category	Accuracy
Distance	Long	1/3	33%
	Medium	9/39	29%
	Short	14/28	50%
Angle	Front	13/29	45%
	Side	11/41	27%
Brightness	Low	10/26	38%
	High	14/44	32%
Clarity	Clear	20/63	32%
	Blurry	4/7	57%

Table 1: Image categories & respective accuracy

Observation

Based on the data presented in the table, it can be observed that the accuracy of the car license detection algorithm varies significantly across different categories and subcategories of car images. The algorithm performed the best on the "Short" subcategory, with a high accuracy rate of 50%, while the "Medium" and "Angle-Front" subcategories had moderate accuracy rates of 29% and 45% respectively. On the other hand, the algorithm had a relatively low accuracy rate for the "Angle-Side" subcategory at 27%.

The accuracy of the algorithm also varied depending on the distance, brightness, and clarity of the car images. The algorithm performed better on images taken from a shorter distance, with an accuracy rate of 33% for the "Long" distance subcategory and 50% for the "Short" distance subcategory. In terms of brightness, the algorithm had a higher accuracy rate of 38% for images with low brightness compared to images with high brightness, which had an accuracy rate of 32%. The algorithm also had a higher accuracy rate of 57% for blurry images compared to clear images, which had an accuracy rate of 32%.

Overall, the algorithm's performance varied depending on the subcategory and image quality factors such as distance, brightness, and clarity. Therefore, further improvements may be required to enhance the accuracy of the algorithm across different subcategories and image quality factors.

Critical Evaluation on the outcomes

The outcomes demonstrate the difficulties of developing a reliable algorithm for car license detection, particularly when dealing with different aspects of image quality. The accuracy scores for the various subcategories indicate that the algorithm might have trouble detecting license plates that aren't facing the camera directly or that have differing brightness or clarity levels. As noted in the observation, noise removal and contour detection were insufficient to reliably recognise the license plates, suggesting that this may be because of the algorithm's preprocessing approaches.

The preprocessing procedures may need to be improved in order to increase the algorithm's accuracy, such as by utilizing more complex edge detection algorithms or cutting-edge noise reduction techniques or using more suitable morphological operation to further enhance and highlight the car license plate region. Additionally, applying machine learning techniques may enhance the algorithm's capability to precisely identify license plates. To increase the accuracy of the car license detection technique, for instance, deep learning models have shown promise in enhancing object detection and image recognition.

In conclusion, there is still potential for development even though the car license detection algorithm has shown diverse degrees of accuracy across all categories and image quality criteria. Although we have a more detail analysis on the category where our algorithm does not perform well on, but further research still need to be done because some of the categories having too little samples which leads to higher accuracy but that does not mean our algorithm will perform well on that. Lastly, the preprocessing methods used by the algorithm need to be improved, and it is necessary to study the usage of modern machine learning methods in order to increase their accuracy and suitability for use in real-world scenarios.

Algorithm 2

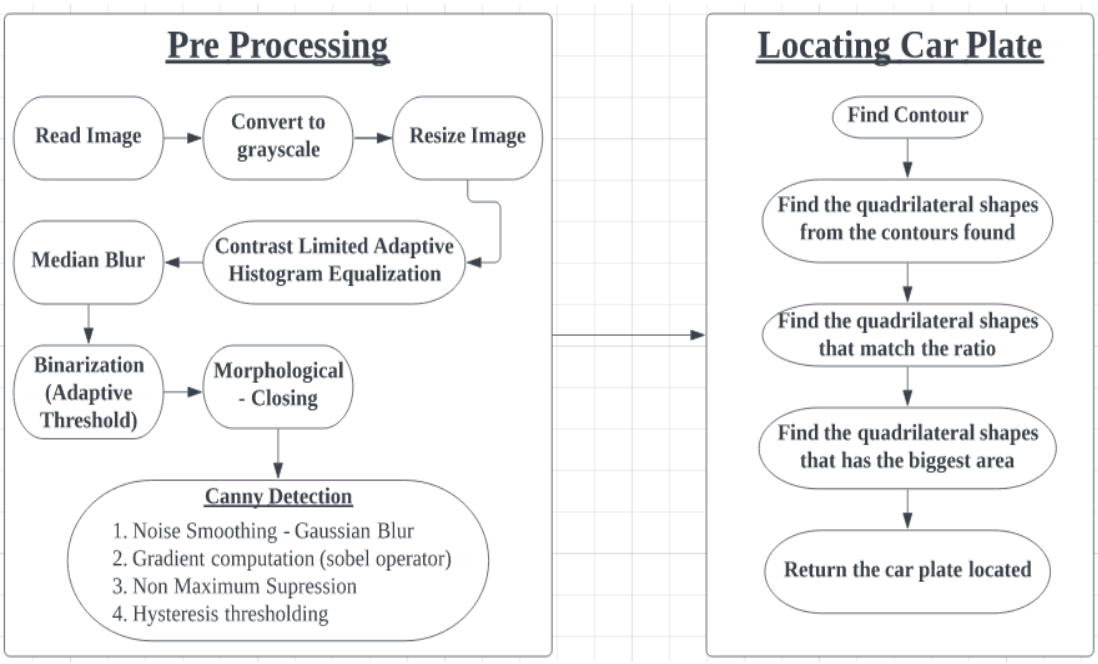

Figure 12. Flow Chart

i) The overall methodology. Include a flow chart and a short description.

The overall methodology of the assignment is shown above. Pre-processing and car plate locating are the two key pipelines in our flow. The initial pipeline consists of eight primary operations: reading the image, grayscale conversion, scaling, contrast limited adaptive histogram equalization, median blurring, binarization, morphological operation(closing) and Canny Edge detection. After the first pipeline, the pre-processed image will move to the second pipeline to locate the car plate. There are five crucial processes on the second pipeline that help locate the car plate. The pipeline begins by finding the contour and then determining the vertices that can result in quadrilateral shapes based on the contour. We will filter out quadrilateral shapes that meet a specific aspect ratio using the result discovered, and we will then look through the list of shapes to locate those with the largest area. Finally, the car plate from the original image will be returned using the coordinates of the quadrilateral shapes with the biggest area found in earlier phases.

ii) Detailed description of your developed algorithms and the Python functions used in each step.

The description will be focused on the developed algorithms that used built in canny edge detection as we implemented two algorithms which one using the built-in function while the other one using manual code function, according to the first pipeline which is focused on enhancing the image and extracting useful information (edges).

The pipeline begins by reading the image using the function from `OpenCV` library `cv2.imread()` and resizing the image to 1024x1024 which reduces the dimension and helps standardize the image size.

The image will then be converted to grayscale image using the function `cv2.cvtColor` with the parameter of the image and `cv2.COLOR_BGR2GRAY`.

After that, the image will then be further processed by using contrast limited adaptive histogram equalization which performs local histogram equalization. The purpose of applying contrast limited adaptive histogram equalization is to improve the visibility of details in the image, especially in regions that have low contrast. This method works by dividing the image into small, overlapping tiles and applying local histogram equalization to each tile separately. This helps to prevent over-amplification of noise in areas with low contrast, which can occur with traditional global histogram equalization methods. This process is being carried out with the function we have created which is `apply_clahe()`, this function includes three parameters which are the image, clip_limit with a default of 2.0 and tile_size with a default of (8,8). In this function, we will be using `cv2.createCLAHE()` using the parameter clip_limit and tile_size to create a CLAHE object then apply the CLAHE object to the grayscale image.

The image will then proceed to removing noise by applying median blur to it. We used the built-in function from `OpenCV cv2.medianBlur()` with a kernel size of 5. We used kernel size of 5 because it can balance the trade-off between noise-reduction and preserving image details.

Then the image is being binarized by using the function cv2.adaptiveThreshold() to separate the foreground objects from the background in an image which is a local thresholding technique. This is useful in car plate detection as it helps to isolate the plate from the rest of the image and make it easier to locate. We applied these parameters to the function src = grayscale image, maxValue = 255, adaptiveMethod = `cv2.ADAPTIVE_THRESH_GAUSSIAN_C`, thresholdType = `cv2.THRESH_BINARY`, blockSize = 41, C = 2. `cv2.ADAPTIVE_THRESH_GAUSSIAN_C` calculates the threshold value based on a weighted sum of neighborhood pixel values where the weights are calculated using a Gaussian window while `cv2.THRESH_BINARY` assigns the maxValue to all pixels with a value above the threshold and 0 to all pixels below the threshold. The use of cv2.ADAPTIVE_THRESH_GAUSSIAN_C ensures that the threshold value is calculated using a weighted sum of the neighborhood pixel values, which helps to reduce the impact of noisy pixels on the threshold calculation. BlockSize of 41 is the size of the neighborhood window used for calculating the threshold value while C is the constant subtracted from the weighted sum of the neighborhood pixel values which is used to adjust the threshold value.

The binarized image is then used to perform a morphological operation, closing, to remove small holes, gaps, and dark spots in an image. We have created a function called morphology_close to perform the operation in which the default parameter of the function is the image and a kernel_size of (3,3). The function consists of two parts which are `cv2.getStructuringElement()` with the parameter `cv2.MORPH_RECT` and kernel_size that will create a rectangular kernel of kernel_size, and `cv2.morphologyEx()` with the parameter of gray, `cv2.MORPH_CLOSE` and the kernel created to carry out the morphological closing operation. The morphological closing operation involves two main steps which are first dilation then erosion.

The pre-processed image will then be passed to the function `cv2.Canny()` to detect the edges in the image. The parameter applied is 100 and 200 which are the low and high threshold values to define the strong and weak edges. The function traces out the edges by linking the strong edges with weak edges that are connected to them and the strong edges will be shown in white pixels while the rest in black pixels.

The car plate detection process begins by detecting the contours of objects in the pre-processed image using the `cv2.findContours()` function. For each contour, the area is calculated using the `cv2.contourArea()` function, and the perimeter is calculated using the `cv2.arcLength()` function. The contour shape is then approximated using the `cv2.approxPolyDP()` function, and the resulting shape is checked to see if it has four vertices. If it does, it is considered a quadrilateral. If the contour is a quadrilateral, its bounding box is calculated using the `cv2.boundingRect()` function, which returns the coordinates of a rectangle that completely covers the contour. The aspect ratio of the bounding box is then calculated by dividing its width by its height. Finally, if the bounding box has an aspect ratio within a specified range, and it has the largest area among all the bounding boxes, it is considered as a car plate.

iii) Detailed description of the experimental results.

Dataset	Number of Image	Accuracy
1	45	17/45
2	15	4/15
3	10	2/10

Figure 13. Accuracy table of Built-In Canny Edge detection

Based on Figure 13, the overall accuracy is 32.857% which clearly shows that the developed algorithm required further improvement to achieve higher accuracy. One of the reasons for the low accuracy of the algorithm is that the algorithm assumes that the license plates are rectangular and have a specific aspect ratio while in real-world scenarios, the license plate may not always have a rectangular shape or the expected aspect ratio which could lead to low accuracy of the algorithms. Besides that, the reason for low accuracy can also be the selection of certain parameters in the algorithm such as threshold values, kernel sizes, etc. If the parameters are not selected appropriately, the algorithm might fail to detect the license plate accurately.

Dataset	Number of Image	Accuracy
1	45	0/45
2	15	0/15
3	10	0/10

Figure 14. Accuracy table of Customized Canny Edge detection

Based on Figure 14, the overall accuracy is 0% which indicates that the algorithm is clearly not applicable as it failed to locate all the car plates of the dataset. This is crucial as we observed several factors that lead to this result. The first factor is the outcome of the customized and built in Canny Edge detection.

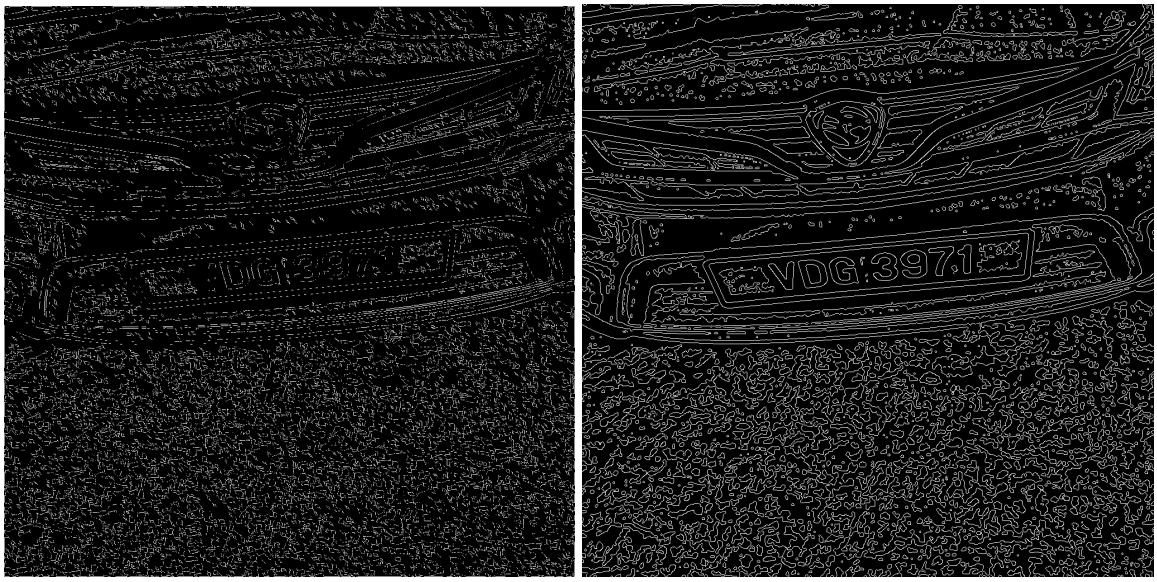

Figure 15. Customized (Left) and Built-In (Right)

The customized canny detection algorithm neglected much detailed information while the built in one managed to capture more detailed information from the image based on Figure 4. This factor will lead to another factor which is the cv2.approxPolyDP() function unable to locate informative quadrilateral shapes.

Figure 16. Customized (Left) and Built-In (Right)

Due to the reason that customized canny detection fails to capture much detailed information, we can observe from Figure 16 that the quadrilateral shapes that are being obtained are all irrelevant and non-informative.

iv) Discussion / critical evaluation on the outcomes and any observations

The failure of the developed algorithms might be because of the real-world scenarios that are out of control which can be clearly seen from the 3 sets provided. In these 3 different sets, there are many pictures that have different distances between the camera and the car, different angles of the camera which can affect the perspective and distort the shape of the car plate.

Category	Sub-category	Number of plate detected / Total plate in the sub-category	Accuracy
Distance	Long	1/3	33%
	Medium	12/39	31%
	Short	10/28	36%
Angle	Front	12/29	41%
	Side	11/41	27%
Brightness	Low	10/26	38%
	High	13/44	30%
Clarity	Clear	20/63	32%
	Blurry	3/7	43%

Table 2: Image categories & respective accuracy

Based on Table 2, we can observe that our algorithm performs differently on varying conditions such as distance, angle, brightness and clarity. In the distance category, our algorithm has the best performance on a short distance between the camera and the car where it has an accuracy of 36% with 10 car plates located out of 28 cars. Besides that, we can see that long and medium distance have a relatively lower accuracy than the short distance which are 33% and 31%. However, we are not able to justify the lowest accuracy as the samples of long distance are low compared to others. Following is the angle category while we observed that our algorithm performs relatively well in the front orientation having a relatively high accuracy of 41% with 12 car plates located in 29 cars ; the side orientation having a lower accuracy of 27% which is 11 out of 41. The brightness category has an accuracy of 38% in the low brightness sub category compared to the high brightness sub category with an accuracy of only 30%. The last category which is the clarity has an accuracy of 32% of the sub category, clear clarity, whereas the sub category, blurry, has a relatively high accuracy of 43%. However, the result of the sub category, blurry, is not definite as the sample size is too small, which is 3 car plates located out of 7 cars. We can conclude that our algorithm performs better on short distance, front orientation and low brightness.

Besides that, the developed algorithm is dedicated to the built-in library as we observed that some of the manual code has relatively high complexity and some of the manual code has different outcome compared to the built-in function which can be crucial to the accuracy of the car plate locating. Overall, the developed algorithms of built-in canny edge detection shows potential for detecting car plates in some images, further research can explore more algorithms to improve the accuracy and robustness of locating the car plate.

Reference

Assaialangaram (2020) License Plate Detection System. Retrieved from https://www.hackster.io/Aasai/license-plate-detection-system-b9297d

Khalifa, O. O., Khan, S., Islam, M. R., & Suleiman, A. (2007). Malaysian Vehicle License Plate Recognition. *ResearchGate*. Retrieved from https://www.researchgate.net/publication/220413760_Malaysian_Vehicle_License_Plate_Recognition

Sharma, V., Jain, M., Jain, T., & Mishra, R. (2022). License Plate Detection and Recognition Using OpenCV–Python. In *Lecture notes in electrical engineering* (pp. 251–261). Springer Science+Business Media. https://doi.org/10.1007/978-981-16-8248-3_20

Kim, T., & Kim, E. T. (2017). A Vehicle License Plate Recognition System Using Morphological ROI (Region of Interest) Map Generated from Morphology Operation. *Journal of Physics*. https://doi.org/10.1088/1742-6596/806/1/012004