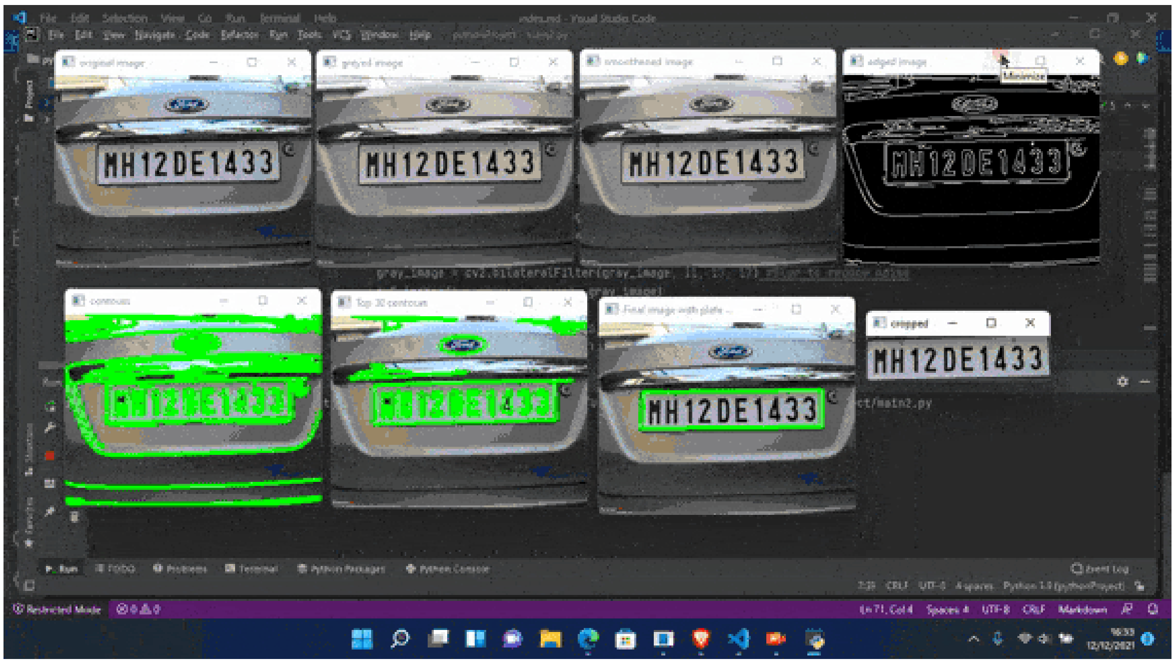
Part I

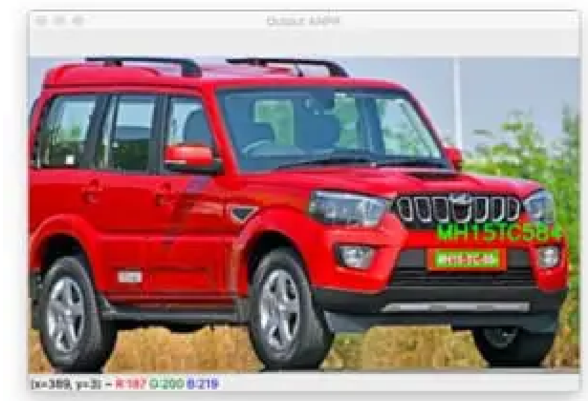
Literature Review

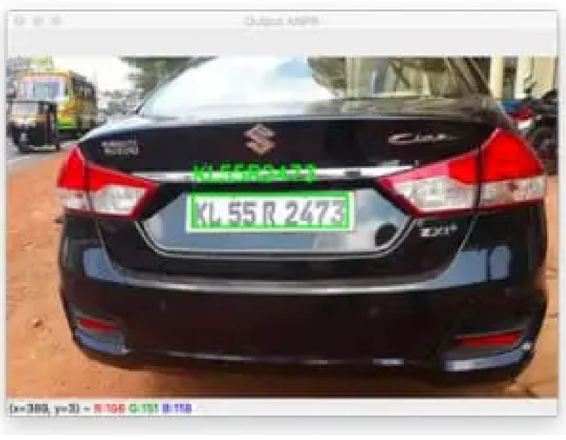
A tutorial (Simon Kiruri, 2021) uses computer vision to identify the part of a car that is the number plate. The tutorial uses pytesseract, imutils, and OpenCv libraries to process, resize, and detect the license plate from an input image. The provided Python code takes an image of a car as input and converts it to a grey image, reduces the noise, detects the edges, draws the contours, finds the best contour, crops the license plate part, and extracts the text from the image.



Resizing the image helps to increase the efficiency of the program. Using a Bilateral filter reduces the noise in the grey image. After applying Canny edge detection, the author uses findContours to find the contours in the image. Since the license plate is assumed to be large, only the 30 largest contours are considered. For each contour, approxPolyDP is used to approximate the polygon that bounds the contour. The program looks for only 4 sided polygons. They then used pytesseract to do optical image recognition on the cropped car plate.

Another tutorial (Rosebrock, 2023) details an Automatic License/Number Plate Recognition (ANPR) system using OpenCV and Python. It focuses on 3 main steps: detecting and localizing a license plate in an input image/frame, extracting the characters from the license plate and applying some form of Optical Character Recognition (OCR) to recognize the extracted characters.





The approach is similar to the first tutorial. They used a blackhat morphological operation to reveal dark characters (letters, digits, and symbols) against light backgrounds (the license plate itself). This is the opposite of Malaysian car plates which have white text on a black background. They applied a closing operation using a small square kernel to fill small holes and help identify larger structures in the image. The Otsu method was then used to perform binary thresholds to reveal the light regions in the image. Using cv2.Sobel, the Scharr gradient magnitude representation was computed in the x-direction of the blackhat image. A series of erosions and dilations were performed to denoise the thresholded image. After finding contours, the program considers contours if the aspect ratio of the bounding rectangle is between 4 and 5. They then used pytesseract to do optical character recognition. They cleared any foreground pixels touching the border of the license plate.

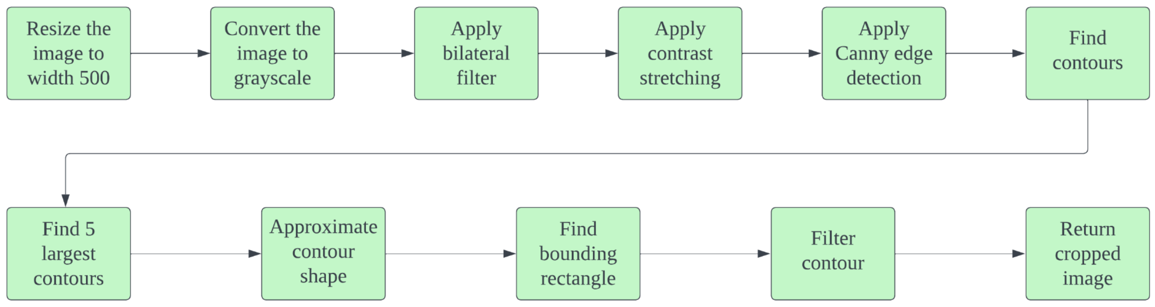
Task Allocation

All 3 members worked on the literature review. Ivan worked on the program which was checked and improved by Chen Xi and Hong Bo by suggesting to check the aspect ratio of the bounding rectangle..

Part II

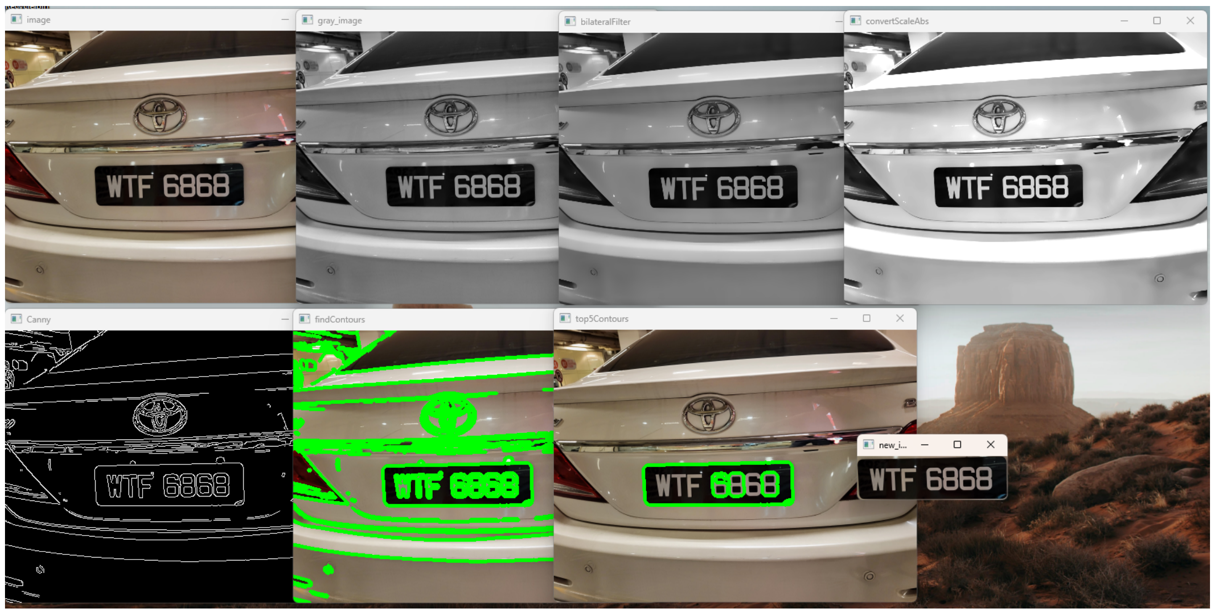
Overall Methodology

Code flowchart

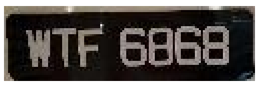


Resize the image to width 500
Convert the image to grayscale
Apply bilateral filter
Apply contrast stretching
Apply Canny edge detection
Find contours
Find 5 largest contours
Approximate contour shape
Find bounding rectangle
Filter contour
Return cropped image

Image result at each stage



Final result



The main code can be broken down into a few key processes

1. Read the input image and resize it to a width of 500 pixels. This allows us to standardise the image sizes and increase the program's efficiency.
2. After converting the image to grayscale, apply a Bilateral filter to reduce noise
3. Scale the intensity of the pixels using contrast stretching to enhance the edges
4. Use Canny edge detection to find edges in the image. This allows us to only focus on the edges of the objects.
5. Finding the contours in the image. A contour is a curve joining all the continuous points (along the boundary), having the same colour or intensity. These contours represent the objects in the image.
6. Sorting the contours based on their area and keeping the 5 largest contours. This reduces the number of contours that need to be processed. Since we assume that the plate number is large, the largest contours are most likely to correspond to the plate number.
7. Find the approximate polygon for the contours and the bounding rectangle. The approximation is done using the Douglas-Peucker algorithm, which recursively removes vertices from the original curve or contour to create a simplified version
8. We employ a rule base filter to find the most appropriate contour. This is done by considering the number of sides of an approximate polygon, the aspect ratio of the bounding rectangle and its area and whether the approximate polygon is convex. We can tweak these rules to account for different cases.

Description of developed algorithms

```
image = cv2.imread("set 1/" + filename)
image = imutils.resize(image, width=500)
```

First we read the image and resize it to a standard width of 500 pixels while maintaining the aspect ratio.

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray_image = cv2.bilateralFilter(gray_image, 11, 17, 17)
```

We convert it to grayscale and then apply a bilateral filter to the image to reduce noise. The bilateral filter is non-linear, edge preserving and smoothing filter that applies a weighted average to the pixels in a local neighborhood. The weight of each pixel in the neighborhood is based on its distance from the center pixel and the similarity in intensity values between the center pixel and its neighboring pixels.

```
gray_image = cv2.convertScaleAbs(gray_image, alpha=1.5, beta=0)
```

We use convertScaleAbs to perform contrast stretching. This highlights the edges and contours of the objects. The alpha parameter determines the contrast scaling factor, which multiplies the pixel values by a constant to increase the image contrast.

```
edged = cv2.Canny(gray_image, 100, 200)
```

We then used Canny to find all the edges in the binary image. Canny edge detection includes smoothing using a Gaussian filter, gradient calculation using the Sobel operator, non-maximum suppression, double threshold filtering and edge tracking using hysteresis. The 2 other parameters represent the lower and upper thresholds for the hysteresis procedure.

```
cnts,new = cv2.findContours(edged, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)
cnts = sorted(cnts, key = cv2.contourArea, reverse = True)[:5]
```

We used findContours for contour detection. RETR_LIST retrieves all the contours and doesn't create any hierarchical relationship. cv2.CHAIN_APPROX_SIMPLE compresses horizontal, vertical, and diagonal segments and leaves only their endpoints. This is used since we only need to crop the license plate base on its endpoints. We sort the contours based on contour area and keep only the 5 largest contours. This is done to reduce runtime as the plate number should be large in comparison to other objects.

```
for c in cnts:
    perimeter = cv2.arcLength(c, True)
    approx = cv2.approxPolyDP(c, 0.018 * perimeter, True)
    x,y,w,h = cv2.boundingRect(c)
    if len(approx) == 4 and cv2.isContourConvex(approx) and 
cv2.contourArea(approx) > 1500 and w/h > 0.5 and w/h < 4.5:
        new_img=image[y:y+h,x:x+w]
        cv2.imwrite("locs/"+filename,new_img)
        break
```

For each contour, we calculate the perimeter of the closed contour. This length is used for the approximation accuracy of approxPolyDP. approxPolyDP works by finding the vertex with the farthest distance from the line segment between the first and last vertex, and recursively splitting the curve or contour until the maximum distance between the original curve and the simplified curve is less than the specified approximation accuracy. We also find a bounding rectangle for the contour.

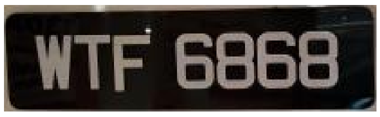
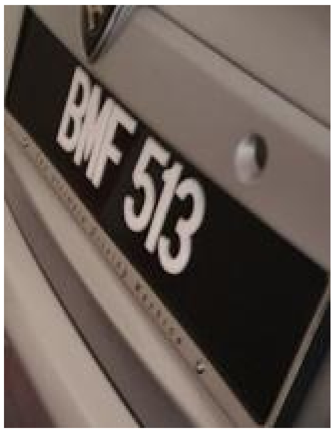
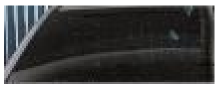
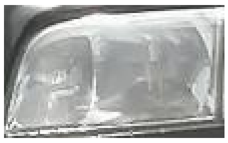
Next, we filter the contour based on some rules:

1. The approximate polygon should have 4 sides since we are looking for a rectangle.
2. The polygon should be convex. This is to filter out 4 sided polygons that are hourglass shape.
3. The area should be more than 1500 pixels squared. This is tuned to remove small objects but should not be too small to omit small plate numbers
4. The aspect ratio of the bounding rectangle should be between 0.5 and 4.5. Since the plate number may be at an angle, there is a slight variance in the aspect ratio of the bounding rectangle.

If the contour fulfils all of the above, it is used to crop out the plate number.

In this implementation, we do not output any image if there is no contour that satisfies the filter. This reduces the false positive rate.

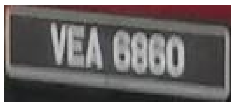
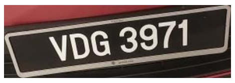
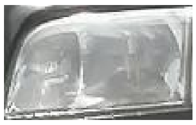
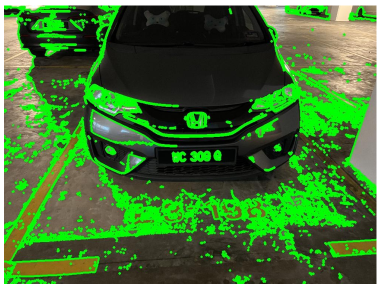
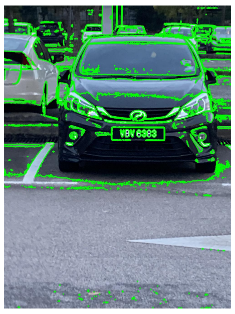
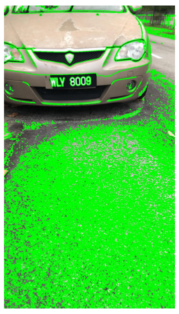
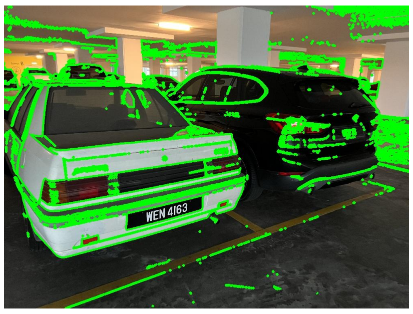
Experimental Results

Category	**Sample**	**Set 1**	**Set 2**
Correctly identified	 	12	3
Incorrectly identified	 	2	0
Did not identify	NA	31	12
Total		**45**	**15**

This implementation correctly identified 15 of the 60 test images. This is a 25% accuracy with a 3% false positive rate.

The accuracy is higher on the first set (27%) vs the second set (20%).

Critical Analysis and Discussion

Category	Examples	Explanation
Correctly identified	 	This implementation works well for inputs where the car plate is directly facing the camera and it has a clear bounding box with a different color from the black background of the car plate.
Wrongly identified		The 2 false positives had similar features to the car plates. More rules will be needed to filter out these cases.
Did not identify (dark body panel)		This implementation suffers greatly from a lack of clear boundary between the car plate and the car body, when the car plate is surrounded by dark body panels or if the car has dark color paint. In this case, it may be easier to identify individual letters as opposed to the entire number plate.
Did not identify (incomplete car plate)	 	There are cases where the contour of the plate number is not closed. This can be due to broken number plates, number plate design or obstructions.
Did not identify (dark body panel)		There are cases where there is more than one plate present in the input. This program will only identify at most one plate. In this case, it identifies the number plate of the white car.

References

Rosebrock, A. (2023, March 22). *OpenCV: Automatic License/Number Plate Recognition (ANPR) with Python - PyImageSearch*. PyImageSearch. OpenCV: Automatic License/Number Plate Recognition (ANPR) with Python - PyImageSearch

Kiruri, S. (2021, December 31). *License Plate Detection And Recognition Using OpenCv And Pytesseract*. Engineering Education (EngEd) Program | Section. License Plate Detection And Recognition Using OpenCv And Pytesseract | Engineering Education (EngEd) Program | Section