22011102044	LOKA NEHAN REDDY
22011102060	MUTHU SANJAI A S
22011102029	JAIDEEP D
22011102011	PRANEETHA
22011102055	M S SHRUTHI
22011102015	DARSHINI P
22011102032	SATHWIK
22011102004	A SAESHWARAN

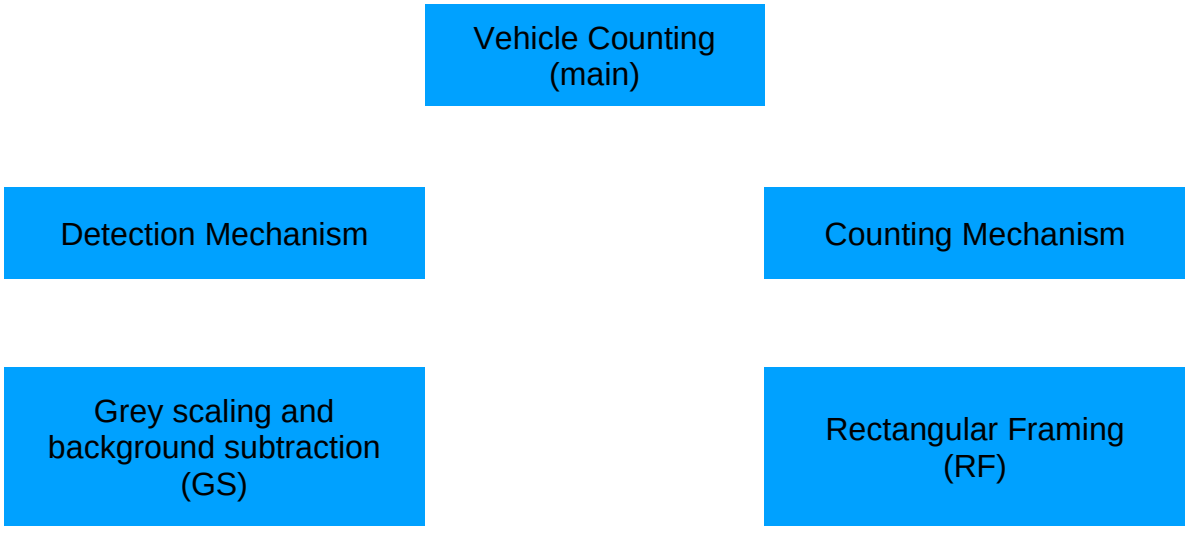
Libraries used:

We have used two libraries, Numpy and open CV for our model. The module imported from Open CV lets us run and analyse components of a video. Along with the computational efficiency provided by Numpy, the mentioned libraries help us analyse traffic videos in detail.

Features used:

We have used features such as grey scaling, background subtraction and dilation to detect a moving body and have created an imaginary boundary which a vehicle must cross to get identified.

Map of the project:



Procedure:

We convert our source video into a greyscale version and define the moving parts(varying pixels) as bright spots on a dark background. We then frame these patches of bright spots into rectangles and define them as a moving vehicle. Then we set up a particular line(reference line) and count if a rectangle crosses the line.

This procedure is repeated and a counter is set up to record the total number of cars crossing the reference line throughout the video.

Functions used:

Main:

'import cv2: Import the OpenCV library.

'import numpy as np': Import the NumPy library with the alias "np".

'cap = cv2. VideoCapture ('video.mp4')': Create a VideoCapture object to read the video file "video.mp4".
'while True:': Start an infinite loop.

'ret, framel= cap. read ()*: Read a frame from the video file using the VideoCapture object and assign it to the variable frame.

'cv2.imshow ('Video Original', frame1)': Display the frame using the cv2. imshow () ' function with a window title "Video Original.

'if cv2.waitKey (1)== 13:' Wait for a key press for 1 millisecond using the cv2.waitKey () ' function. If the pressed key has the ASCII value 13 (which corresponds to the Enter key), break out of the loop.

'cv2. destroyAllWindows () ': Destroy all windows created by 'cv2.imshow ()'.

'cap. release ()': Release the VideoCapture object. This is important to do when you're done working with the video file to free up resources.

GS:

The cv2.findContours () ' function returns a list of contours in the image. Each contour is represented as a list of points, and each point is represented as a tuple of (x, y) coordinates.

The enumerate () ' function is used to loop through the contours and keep track of their index numbers. For each contour, the cv2. boundingRect () ' function is used to calculate the coordinates of the bounding rectangle.

The validate_counter' variable is used to check if the width and height of the bounding rectangle are greater than or equal to the minimum width and height values set by

'min_width _rect' and 'min_height _rect. If the values do not meet the criteria, the loop continues to the next contour.

If the '**validate_counter'** condition is met, the **cv2. rectangle () '** function is used to draw the bounding box around the contour in the original image ('**frame**"). The rectanale is drawn using the top-left and bottom-right coordinates of the bounding rectangle calculated by **cv2. boundingRect ()**. The rectangle is colored red **(* (0,0,255) *)** and has a thickness of 2 pixels **(‘2*)**.

The **'for'** loop iterates through each center point in '**detect** list and checks if the '**y** coordinate of that point falls within the counting line position with an offset of '**+ / -** ' some pixels. If the condition is true, it increments the counter variable by 1, removes that center point from the **detect** list, draws a line on the counting line, and prints the current count on the console.

After the loop, **cv2.putText ()** ' function is used to write the current count on the output frame at position " **(450, 70)** ' with the specified font size, color, and thickness.

This code snippet is used to draw bounding boxes around the contours detected in the image.
The cv2. findContours ()' function returns a list of contours in the image. Each contour is represented as a list of points, and each point is represented as a tuple of (x, y) coordinates.
The 'enumerate () ' function is used to loop through the contours and keep track of their index numbers. For each contour, the cv2. boundingRect () ' function is used to calculate the coordinates of the bounding rectangle.
The 'validate_counter variable is used to check if the width and height of the bounding rectangle are greater than or equal to the minimum width and height values set by

RF:

'min_width_rect' and 'min_height_rect*. If the values do not meet the criteria, the loop continues to the next contour.
If the 'validate counter' condition is met, the cv2. rectangle () ' function is used to draw the bounding box around the contour in the original image ('frame*). The rectangle is drawn using the top-left and bottom-right coordinates of the bounding rectangle calculated by cv2. boundingRect (). The rectangle is colored red (' (0,0,255) *) and has a thickness of 2 pixels ('2*).

The 'for' loop iterates through each center point in detect list and checks if the "y' coordinate of that point falls within the counting line position with an offset of '+/ - ' some pixels. If the condition is true, it increments the counter variable by 1, removes that center point from the detect list, draws a line on the counting line, and prints the current count on the console.
After the loop, 'cv2.putText () ' function is used to write the current count on the output frame at position '' (450, 70) ' with the specified font size, color, and thickness.