PROJECT TITLE: ADVANCED DRIVER ASSIST SYSTEM (ADAS)

DINESH KUMAR S	(18R211)
MOHAMED RIFAYEE HUSSAIN Z	(18R229)
SANJAY S	(18R242)
SARAVANA KUMAR K	(18R246)
VASANTHAKUMAR S	(18R258)

Branch: ROBOTICS AND AUTOMATION ENGINEERING

15R612 – Internship II and Innovation Practices

APRIL 2021

DEPARTMENT OF ROBOTICS AND AUTOMATION ENGINEERING

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

ADVANCED DRIVER ASSISTANT SYSTEM (ADAS)

Bonafide record of work done by

DINESH KUMAR S	**(18R211)**
MOHAMED RIFAYEE HUSSAIN Z	**(18R229)**
SANJAY S	**(18R242)**
SARAVANA KUMAR K	**(18R246)**
VASANTHAKUMAR S	**(18R258)**

BACHELOR OF ENGINEERING

Branch: ROBOTICS AND AUTOMATION ENGINEERING

15R612 – Internship II and Innovation Practices

…………..………………

Faculty In-charge

Certified that the candidate was examined in the viva-voce examination held on

____

……………………..   …………………………..

(Internal Examiner)   (External Examiner)

CONTENTS

ACKNOWLEDGEMENT

We wish to express our gratitude to **Dr. K. Prakasan**, Principal of PSG College of Technology, for providing the necessary facilities to carry out this project.

We are very much indebted to **Dr. B. Vinod**, Head of Department, Department of Robotics and Automation, for his magnanimous support and guidance.

We are very thankful to **Dr. M. Sundaram**, Program Coordinator, Department of Robotics and Automation, for his encouragement.

We extend our sincere gratitude to **Dr. G. Subashini,** Professor, Department of Robotics and Automation, **Dr. B. Bindu,** Assistant Professor (Sr. Gr.), Department of Robotics and Automation, Dr. **N. Jonah,** Professor, Department of Robotics and Automation for their constant support and encouragement throughout the internship process.

We also extend our special thanks to our family and friends for their support given to us for the successful completion of our project.

SYNOPSIS

The project's main goal is to automate and improve vehicle safety by alerting the driver to possible issues and preventing accidents.

Despite the fact that they happen often, road accidents are the most terrifying thing that can happen to a driver. Worst of all, we refuse to learn from our mistakes along the way. The majority of road users are aware of the general rules and safety measures to take while on the road, but injuries and crashes are caused by the road user’s own negligence. Human error is the most common cause of injuries and crashes.

The aim of this project is to automate and improve the safety of vehicles. We use LDWS and EDAS for this.

The LDWS uses cameras to track lane markers to see if the driver is drifting accidentally. Instead of taking control of the vehicle to help sway it back into the safe zone, the device gives the driver an audio or visual warning.

If the driver falls asleep or does not conduct any driving activity for a specified period of time, the EDAS facilitates emergency countermeasures. These systems do not take control of the vehicle; instead, they give the driver an alarm signal in the form of a warning alert.

Our goal is to follow the lane and adjust to any situation (low light, smoggy road, rainy road). Also, warn the driver if he falls asleep.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

S. No.	Abbreviation	Abbreviated Term
1.	ADAS	Advanced Driver Assist System
2.	LDWS	Lane Departure Warning System
3.	EDAS	Emergency Driver Assist System
4.	FoV	Field of View
5.	CHT	Circular Hough Transform

CHAPTER 1

INTRODUCTION

1.1 AUTONOMOUS VEHICLE

An autonomous vehicle, or a driverless vehicle, will operate itself and perform necessary functions with no human intervention, through the flexibility to sense its surroundings. In the autonomous vehicle, there are many functions and devices that are present to get the data from its surroundings. An autonomous car can go anywhere a traditional car and can do everything that an experienced human driver does. Sensors to get the distance between neighbouring cars and cameras to detect paths and traffic signs. The most crucial part of an Autonomous car is Lane detection, it should correctly trace the path to guide the vehicle accordingly. Advanced Driver Assistance Systems are an automated system which works for the safety of people and improves the vehicle safety by giving warning to the driver.

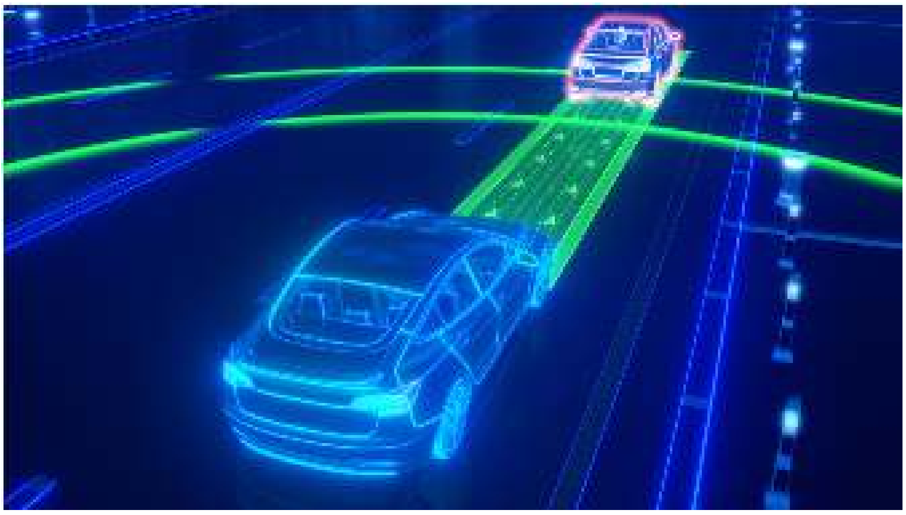


Fig 1.1 Autonomous vehicle

1.2 PROBLEM STATEMENT

Improving safety and reducing road accidents, thereby saving lives is one of the solutions within the context of Advanced Driver Assistance Systems. Among the challenging tasks of Self-driving vehicle are Lane detection, Road boundary recognition and alerting the driver during inactivity. This is an initiative to eliminate the need for a driver.

1.3 TYPES OF ADAS

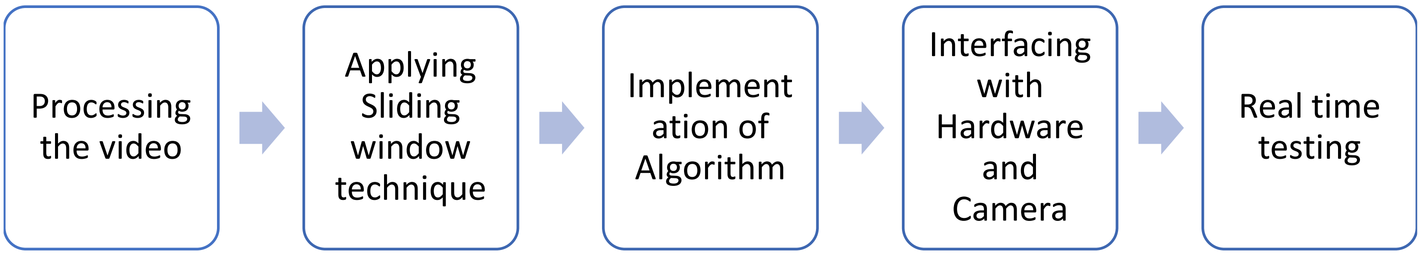
Advanced driver assist system has various types where each type has its specific functionality and automation level increases on each type. The first two types come under passive type and the other two comes under active type.

Table 1.1 Types of ADAS

MONITORING	In this type of ADAS, many sensors and cameras are employed to observe the surrounding and just gives information to the driver
WARNING	Here to be the information from the sensor and cameras is analysed and Insist the driver in case of a warning or any emergency situation. An alert message is given to the driver based on the information collected from th surrounding
ADAPTIVE	The ADAS system gathers information about the surroundings and takes control over the vehicle by adapting based on the information
AUTOMATED	Entire control of the vehicle will be taken over by the ADAS system where th input of the sensors and cameras are analysed and perform the required action in vehicle. This type of system is completely automated which is used in self-driving cars

In this paper we are developing Warning based ADAS where the information is collected from a single camera (avoiding High-cost sensors in order to reduce the cost of this entire system) and using some Computer vision concepts to calculate the real time values and alert the driver in case of emergency situation.

1.4 METHODOLOGY


Processing the video
Applying Sliding window technique
Implementation of Algorithm
Interfacing with Hardware and Camera
Real time testing

1.5 LITERATURE SURVEY

Table 1.2 Literature survey

TITLE	AUTHOR / COMPANY	PUBLICATION & YEAR	DESCRIPTION
Lane Detection with Lane Departure Warning System	Pranali Rajkapur Nagrale, Dr.V.P. Kshirsagar	International Journal of Scientific Development and Research (IJSDR), Published on August 2019	1. Camera-based Real World Lane detection with lane departure warning system. 2. Canny edge detection and Hough transformation algorithms are used in tracing the lane 3. Some faded gray regions will appear in the output of Hough transform we need to distinguish between the lane and the faded regions.
A study on Lane Departure Warning System and Object Detection on Roads for Forward Collision Avoidance	Faiz Habeeb. K. N. Kunan, N. Tuturaja	International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering, Published on May 2018	1. There are many different types of accidents, including rear end collisions, accidents while changing lanes, and driving off the road 2. Seat belts and air bags can reduce damage but they won’t prevent accidents. With this Collision warning system, we can eliminate accidents 3. The proposed method uses yolo models to detect other cars and alert the driver
Real-time drowsiness detection system for an intelligent vehicle	Marco Javier Flores, Jose Maria Armingol, Arturo de la Escalera	IEEE, published on June 2008	1. A non-intrusive driver's drowsiness system based on computer vision and Artificial Intelligent has been presented. 2. This system uses advanced technologies for analysing and monitoring driver's eye state.

TITLE	AUTHOR / COMPANY	PUBLICATION & YEAR	DESCRIPTION
Driver's Fatigue Detection Based on Yawning Extraction	Nawal Alioua, Aouatif Amine, and Mohammed Rziza	International Journal of Vechicular Technology, Published on August 2014	1. Based on CHT, a new algorithm has been developed for yawning detection. 2. The whole driver's fatigue detection system uses three steps: face detection, mouth region localization, and wide-open mouth detection.
Visual Based approach for drowsiness detection	Belhassen AKROUT and Walid MAHDI	IEEE Intelligent Vehicles Symposium, published June 2013	1. This system uses eyes state by the video analysis of several topics. It locates the face and eye driver using the HaarCascade feature. Circular Hough Transform allows the detection of the iris centre and the points of intersection for the two lids. 2. The result of signal after the extraction of two geometric descriptors is non-linear and non-stationary. The results of this approach show a good accuracy rate. To improve them we need to produce hybrid systems which use 3D force estimation

1.6 LANGUAGE AND LIBRARIES USED

Python 3.8.5

Python is interpreted as a general purposed language. It is easy to learn and use. It is also supported in various fields like computer vision, machine learning, neural networks etc. Many Libraries are available in this language and they are very easy to understand. It is a built-in, high-level data structure that combines dynamic typing and dynamic binding. The main reason to use this language is that it is completely open source.

OpenCV 4.4.0.46

OpenCV is a library used for implementing computer vision concepts in the real world. It is supported in python language. It is an open-source computer vision library.

NumPy 1.18.5

NumPy is a numeric python which deals with the array where the images are stored as an array of pixels and to perform certain operations on that array to modify and extract information from those images. It has functions like linear algebra, fourier transform and matrices to work on images.

Matplotlib 3.3.3

Matplotlib is a data visualization library which is supported by python. This library is the most interactive, animated visualization. The distribution of that array of pixels corresponding to particular images is visualized for better understanding.

TensorFlow 2.4.1

TensorFlow library is used in developing advanced topics like Neural network and Machine Learning Architectures. Developed by Google and it is an open-source library. This TensorFlow library is used to load pretrained models and is used for prediction.

Dlib 19.21.1

Dlib is a open-source C++ library which is widely used for computer vision and machine learning topics. In this paper, the dlib library is used to detect the mouth and eye of the driver from the input frame to perform and analyse the drowsiness of the driver and alarms if he falls asleep.

CHAPTER 2

OBJECTIVE

2.1 EXPECTED OUTCOME

The main objective is to trace the lane, adapt in all conditions (low light, smoggy road, rainy road) and to alert the drowsy driver. Lane detection is one of the most important features of the Advanced Driver assistant system. Several algorithms have been developed with the differences in their image processing techniques. However, it is difficult to achieve a higher rate of accuracy in situations like the presence of shadows, varying illuminations, bad quality of road painting and physical barriers. To detect the drowsiness of the driver we use certain computer vision concepts. Analysing the driver's face should predict the condition of the driver. It should alert the driver in case of carelessness and sleepy condition.

2.2 PROPOSED SOLUTION

The first Solution is the traditional method called Hough transform which is used to detect the lines from an image. Hough transform is a computer vision-based feature extraction technique used in image analysis. The Hough transform can be used to detect line circles or parametric curves.

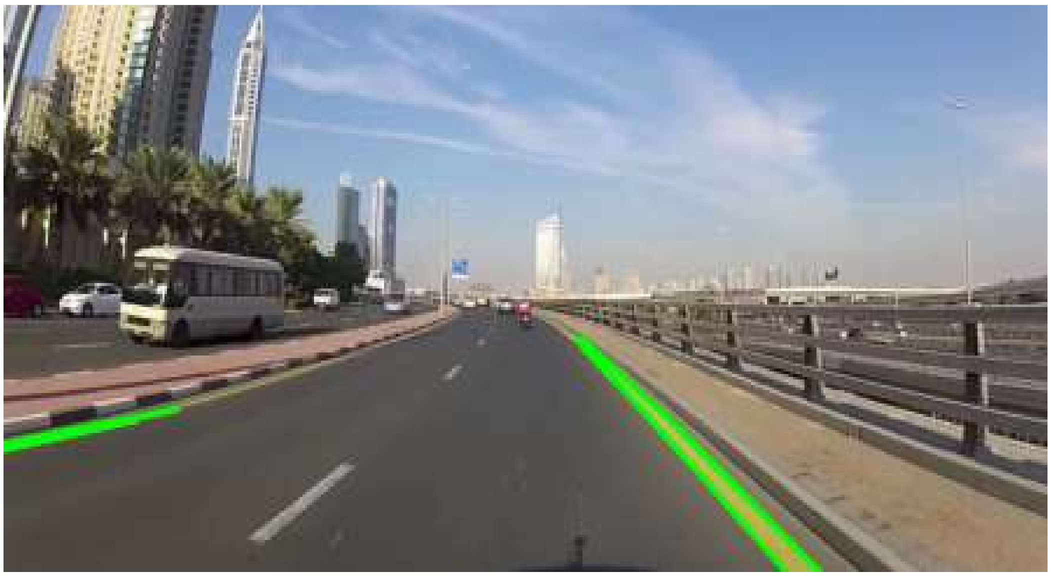


Fig 2.1 Hough tranform

Another method is the Sliding Window technique, which starts at the bottom of the image and scans all the way to the top of image. Each time we search within a sliding window, we add the potential lane line pixels to a list. If we have enough lane pixels in the window. The mean position of these pixels becomes the centre of the next sliding window. Once we have identified the pixels that correspond to the left and right lane lines, we draw a polynomial best fit line through the pixels. This line represents our best estimate of the lane lines.

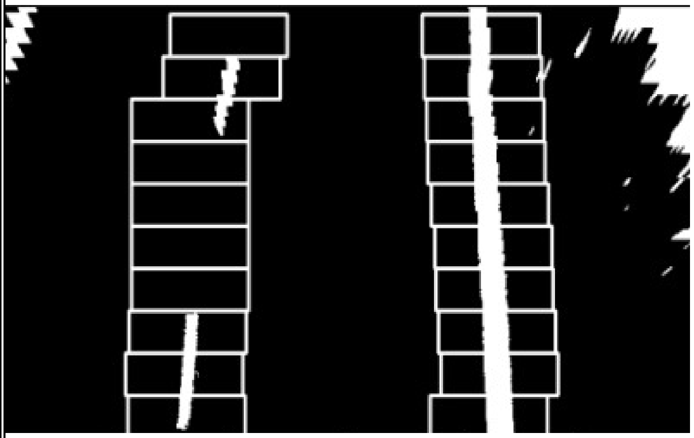


Fig 2.2 Warped frame with sliding windows

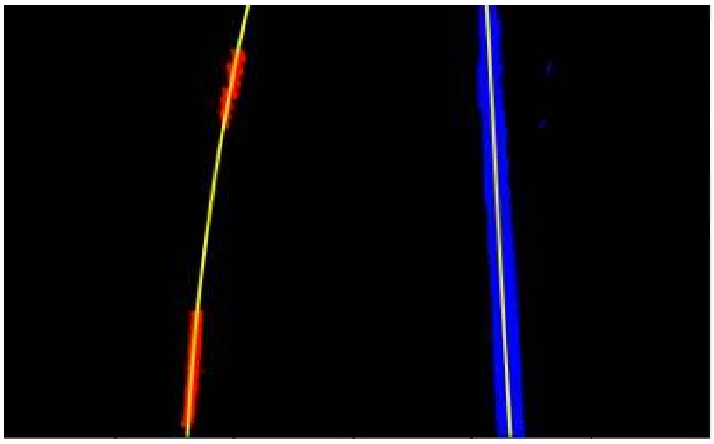


Fig 2.3 Detecting lane lines with sliding windows

The second solution for both drowsiness and yawning is a simple OpenCV approach. The expected performance is not met. We moved on to segmentation using haarcascade and detecting with neural networks.

2.3 EFFICIENCY OF PROPOSED SOLUTION

Comparing both methods, the Sliding window Method seems to be more accurate and gives the desired output. Using Hough transform sometimes misclassified the lane lines. It can give unexpected results when objects appear to be aligned like a straight line by chance. Detected lines are infinite lines described by their parameter values, rather than finite lines with defined end points. So, we go with the Sliding Window technique which calculates the average of pixels known to the base of of the next window above, we can repeat this over and over until we get to the top the lane. So, this will give better results compared to Another transform. Using only OpenCV can lead to misclassification and the processing time is comparatively higher than other methods. This algorithm does not satisfy your needs and meets your expected accuracy.

2.4 FINAL SOLUTION

The final implementation of the algorithm will be based on the Sliding window technique. We saw that the Efficiency of Sliding window algorithm is quite accurate compared to Hough transform. Since the mean of these pixels are calculated there is less possibility of error occurrence. So that we are able to estimate the best fit line for the lane. We go for implementing Sliding window techniques. Yawning detection uses landmarks of the face and detects mouth position. So, we are sticking with the method based on the NN as its accuracy level and response time are matching our requirement.

CHAPTER 3

ALGORITHM IMPLEMENTATION PROCESS OF LDWS

3.1 IMPORTING NECESSARY LIBRARIES

```
import cv2
import numpy as np
from timeit import default_timer as timer
```

We imported the required libraries to implement some computer vision concepts and operations on array. OpenCV and NumPy are the libraries used in this project. Along with this we used a timer library to calculate fps.

3.2 PROCESSING

```
def Processing(frame):
```

To process the video captured by the camera, we have to set the threshold value for the lane to be recognized. There are 2 different types of lanes. White for the roads in public places and yellow color for the roads in private places. We have to write the code for both the possibilities as the car is going to run in both the places.

```
    lower_white = np.array([0, 160, 10])

    upper_white = np.array([255, 255, 255])

    mask = cv2.inRange(frame, lower_white, upper_white)

    hls_result0 = cv2.bitwise_and(frame, frame, mask = mask)
```

Starting from the first line we are giving the lower value and the upper value of the white to be recognized. We are providing these values as arguments for the mask variable in order to detect them. Then we have created a variable called hls_result0 and performing the bitwise_and operation with the frames and mask variable as the arguments.

```
    lower_yellow = np.array([10, 0, 100])

    upper_yellow = np.array([40, 255, 255])

    mask1 = cv2.inRange(frame, lower_yellow, upper_yellow)

    mask2 = cv2.bitwise_or(mask,mask1)

    hls_result1 = cv2.bitwise_and(frame, frame, mask = mask2)
```

We are doing the same process for the yellow lanes as well except for the thing we are performing a bitwise_or operation with the mask and mask1 variables and storing it in the mask2 variable, and using that for finding hls_result1.

```
    hls_result = cv2.bitwise_or(hls_result0,hls_result1)
```

And the next step is performing OR operation with both hls_result0 and hls_result1 in order to make the code work for all possibilities.

```
    gray = cv2.cvtColor(hls_result, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(gray, 160, 255, cv2.THRESH_BINARY)
```

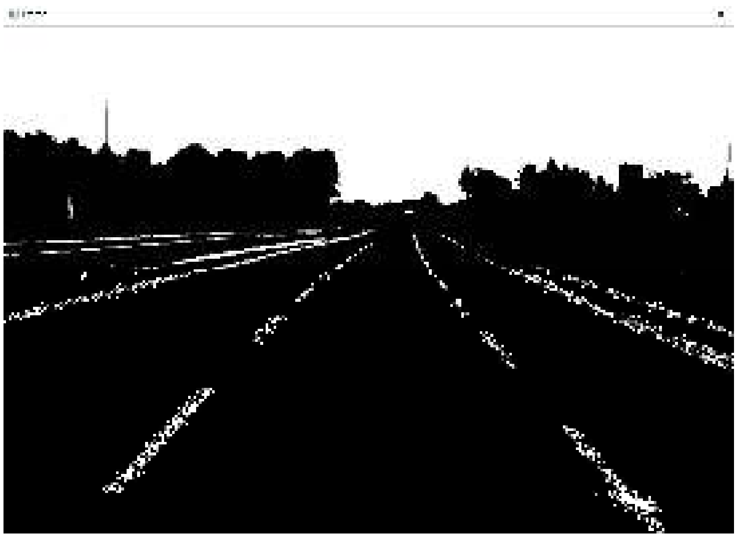


Fig 3.1 Masked binary image

Then the masked image is converted to gray. The gray converted image is passed into threshold and here we are using cv2.THRESH_BINARY and have set the threshold value at 160 and maximum value at 255.

```
    blur = cv2.GaussianBlur(thresh,(3, 3), 0)
```

The next step we have done is apply gaussian blur to the threshold image stored in the variable thresh and the second argument to pass in the cv2.GussianBlur is the kernel size. Here we have taken the kernel size as 3x3 and sigma value as 0.

```
    canny = cv2.Canny(blur, 40, 60)
```

We are performing canny edge detection for the blurred image by passing the arguments blur, and the 2 threshold values.

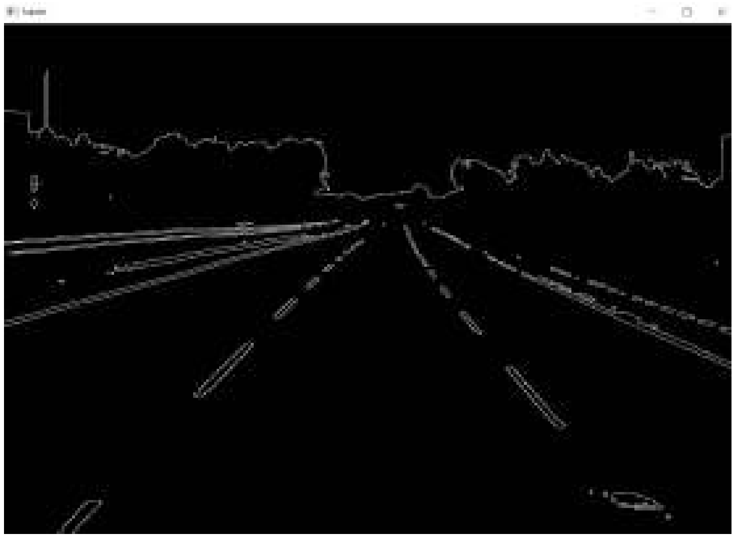


Fig 3.2 Canny edge detection

```
    return canny,thresh
```

Finally, we are returning the variables canny and thresh where the canny edge, detected image and threshold image are stored.

3.3 PERSPECTIVE TRANSFORMATION

```
def Perspective_Transformation(binary_img):
```

The perspective transformation of the image frame should be calculated on calling this function providing the image to be processed as the argument.

```
    img_size = (frame1.shape[1], frame1.shape[0])
    width,height = img_size
```

For finding the perspective transformed image we need the width and height of the image, so we are creating a tuple named “img_size” which gets the value of the width and height from the line frame1.shape.

```
    dst = np.float32([[0,0],[width,0],[0,height],[width,height]])
    x1 = width*0.35
    y1 = height*0.5
    x2 = width*0.6
    y2 = height*0.5
    src = np.float32([[x1,y1],[x2,y2],[0,height],[width,height]])
```

The variable “src” indicates the four ends of the source image, the points x1, y1, x2, y2 are calculated using trial and error and the variable “dst” holds the four ends of the destination image. We are converting all points into float variable using np.float32.

```
    matrix = cv2.getPerspectiveTransform(src, dst)
```

We are passing those variables as the arguments into the function cv2.getPerspectiveTransform which returns the values as the matrix stored in the variable named “matrix”.

```
    birdseye = cv2.warpPerspective(binary_img, matrix, (width,height))
```

In the function cv2.warpPerspective we are passing the canny edge detected image, matrix and the img_size. It returns the image to the variable “birdseye”.

```
    minv = cv2.getPerspectiveTransform(dst, src)
```

The variable “minv” is just the inverse of matrix.

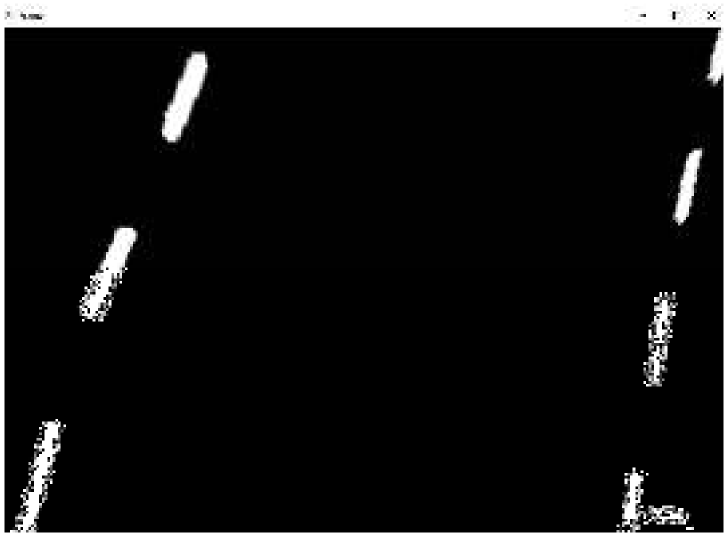


Fig 3.3 Perspective transformation

```
    return birdseye,minv
```

Finally, we are returning the birdseye transformed image and matrix inverse (“minv”) obtained from the perspective transformation.

3.4 WINDOW SEARCH

Sliding window technique is an advanced concept where some rectangle passes through the entire image to identify the particular region. Here the lane part is converted to birdeye view in order to do specific operation only on lane part. Further the transformed image is converted to binary image so that it would be easy to identify the lane line. Sometimes the lane line is yellow in colour so we have to consider this lane line too. We have applied a mask to the image to get that yellow line.

```
Def window_search(binary_warped):
    hist = np.sum(binary_warped[int(binary_warped.shape[0]/2):,:], axis=0)
    result = np.dstack((binary_warped, binary_warped, binary_warped))*255
    mid = np.int(hist.shape[0]/2)
    leftlane = np.argmax(hist[:mid])
    rightlane = np.argmax(hist[mid:]) + mid
    slidingwindows = 20
    window_height = np.int(binary_warped.shape[0]/slidingwindows)
```

Here the binary image is fed to the window search function and this image is stacked in three channels. Then the midpoint of lane is calculated and the position of left and right lane pixels are also calculated using the argmax function. For the stacked binary image, 20 sliding windows is considered as it depends on the programmer. The height of the sliding window is calculated from the image size and number of sliding windows used.

```
    Nonzero = binary_warped.nonzero()
    nonzeroy = np.array(nonzero[0])
    nonzerox = np.array(nonzero[1])
    current_left = leftlane
    current_right = rightlane
    margin = 70
    minpix = 20
    correct_left = []
    correct_right = []
```

The next nonzero function is used to get the position of pixels having more value. In the input binary image only the lane part has more pixel values and positions of that pixels are stored in nonzero variable. Current_left and current_right variable stores the current values of left lane and right lane pixels. Margin is fixed as 70 and minimum pixel is taken as 20 where the inside the rectangle of sliding window should have minimum of 20 pixels should have high pixel values.

```
For window in range(slidingwindows):
        y_low = binary_warped.shape[0] - (window+1)*window_height
        y_high = binary_warped.shape[0] - window*window_height
        w_xleft_low = current_left - margin
        w_xleft_high = current_left + margin
        w_xright_low = current_right - margin
        w_xright_high = current_right + margin
        cv2.rectangle(result,(w_xleft_low,y_low),(w_xleft_high,y_high),(0,
255,0), 2)
        cv2.rectangle(result,(w_xright_low,y_low),(w_xright_high,y_high),(
0,255,0), 2)
        goodleftlane_pos = ((nonzeroy >= y_low) & (nonzeroy < y_high) & (n
onzerox >= w_xleft_low) & (nonzerox < w_xleft_high)).nonzero()[0]
        goodrightlane_pos = ((nonzeroy >= y_low) & (nonzeroy < y_high) & (
nonzerox >= w_xright_low) & (nonzerox < w_xright_high)).nonzero()[0]
        correct_left.append(goodleftlane_pos)
        correct_right.append(goodrightlane_pos)
```

The sliding window is currently looped throughout the image used for looping. Now the coordinates of the rectangle of the sliding window are changing in each iteration and it is updated for both the left and right lanes. Using these coordinates, the rectangle has been drawn and checked for conditions that, inside a rectangle, should a at least have minimum of 20 pixels of high value (white pixels). Then the updated lane position is checked for correction and only good lane position of left and right lane is appended in a list.

```
    If len(goodleftlane_pos) > minpix:
            current_left = np.int(np.mean(nonzerox[goodleftlane_pos]))
        if len(goodrightlane_pos) > minpix:
            current_right = np.int(np.mean(nonzerox[goodrightlane_pos]))
    correct_left = np.concatenate(correct_left)
    correct_right = np.concatenate(correct_right)
    leftx = nonzerox[correct_left]
    lefty = nonzeroy[correct_left]
    rightx = nonzerox[correct_right]
    righty = nonzeroy[correct_right]
```

After getting the good lane positions mean is calculated to avoid misclassification of lane pixels in sliding window. In the " if " condition good lane positions are checked for minimum pixel value.

```
    Left_fit = np.polyfit(lefty,leftx,2)
    right_fit = np.polyfit(righty,rightx,2)
    line=np.linspace(0, binary_warped.shape[0]-1, binary_warped.shape[0] )
    fit_left = left_fit[0]*line**2 + left_fit[1]*line + left_fit[2]
    fit_right = right_fit[0]*line**2 + right_fit[1]*line + right_fit[2]
    result[nonzeroy[correct_left],nonzerox[correct_left]]=[255, 255, 255]
    result[nonzeroy[correct_right],nonzerox[correct_right]]=[255, 255,255]
    right = np.asarray(tuple(zip(fit_right, line)), np.int32)
    left = np.asarray(tuple(zip(fit_left, line)), np.int32)
    cv2.polylines(result, [right], False, (1,1,0), thickness=5)
    cv2.polylines(result, [left], False, (1,1,0), thickness=5)
    return line,mid,leftlane,rightlane,left_fit,right_fit,fit_left,
           fit_right,result,leftx,rightx
```

Polyfit, linspace, fit_left and fit_right function is used to create polynomial function of degree 2 from the position of left lane and right lane. Finally, Polylines are drawn on stacked binary images and certain values are returned to perform further operations.

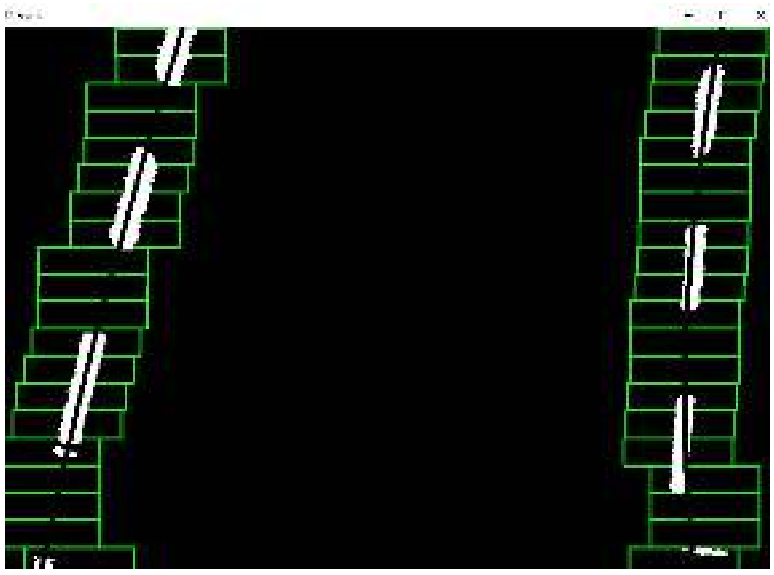


Fig 3.4 Sliding windows

3.5 RADIUS OF CURVATURE

Radius of curvature of lane is calculated from the returned values of pixels in window search function. Now the calculated values are in the form of pixels, so we have to convert them into real world values. We already knew the width of lane as 3.7 meters and it is divided by number of pixels.

```
def radius_of_curvature(line,fit_left,fit_right):
    max_y = np.max(line)
    left_fit_cr = np.polyfit(line*ym_per_pix, fit_left*xm_per_pix, 2)
    right_fit_cr = np.polyfit(line*ym_per_pix, fit_right*xm_per_pix, 2)

    left_curved_rad  = ((1 + (2*left_fit_cr[0]*max_y*ym_per_pix +
            left_fit_cr[1])**2)**1.5) / np.absolute(2*left_fit_cr[0])

    right_curved_rad = ((1 + (2*right_fit_cr[0]*max_y*ym_per_pix +
            right_fit_cr[1])**2)**1.5) / np.absolute(2*right_fit_cr[0])

    if fit_left[0] - fit_left[-1] > 60:
        curve_direction = 'Right Curve'

    elif fit_left[-1] - fit_left[0] > 60:
        curve_direction = 'Left Curve'

    else:
        curve_direction = 'Straight'

    curve_rad = (left_curved_rad + right_curved_rad) / 2.0
```

Similar to the window search method, we use a polyfit function to develop polynomial equations of lane lines and are converted in terms of real world values. We have a formula to calculate the Radius of curvature.

$$R = \frac{(1+(\frac{dy}{dx})^2)^{3/2}}{|\frac{d^2y}{dx}|}$$

After calculating the radiation of curvature, we were able to say whether there is a left turn or a right turn.

3.6 DRAWING LANE LINES

Now we have all the calculated values which we have to implement in the real output frame to make it visible to the user.

```
def draw_lane_lines(original_image, warped_image, letfx,rightx,fitx_left,
                    fitx_right,ploty,offcenter):
   warp_zero = np.zeros_like(warped_image).astype(np.uint8)
   color_warp = np.dstack((warp_zero, warp_zero, warp_zero))

   pts_left = np.array([np.transpose(np.vstack([fitx_left, ploty]))])

   pts_right=np.array([np.flipud(np.transpose(np.vstack([fitx_right,ploty]
)))])

   pts = np.hstack((pts_left, pts_right))
   mean_x = np.mean((fitx_left, fitx_right), axis=0)
   pts_mean=np.array([np.flipud(np.transpose(np.vstack([mean_x,ploty])))])
   if round(offcenter) > threshold:
       cv2.fillPoly(color_warp, np.int_([pts]), (0, 0, 255))
   else:
       cv2.fillPoly(color_warp, np.int_([pts]), (0,255, 0))

   newwarp =cv2.warpPerspective(color_warp,minv,(original_image.shape[1],
                                original_image.shape[0]))

   result = cv2.addWeighted(original_image, 1, newwarp, 0.3, 0)
   return pts_mean, result
```

In this function we get the original frame and binary image. We stacked the binary image into 3 channels. Then the left and right lane points are to be arranged properly in order to collaborate it with the original frame. Finally, both the points are stacked horizontally into a single list. Since we are developing a lane departure warning system, we have to be alert if the vehicle goes or moves away from the offset with some tolerance. We used offset value and fillpoly function which fills the space between left and right lane in different colours to distinguish when vehicle maintains the offset and deviates from the offset. Finally, we do warp perspective and add weight to maintain pixel clarity in case of perspective transformation. Then return the final output image.

3.7 CALCULATING OFFSET

From the window search method, we have returned lots of values which are used in further operations and functions. We have left and right lane values and positions. We use this parameter to calculate the center of both the lanes and find the offset values from the center of the lane.

```
def offCenter(mid,left,right):
    a = mid - left
    b = right - mid
    width = right - left
    if a >= b:
        offset = a / frame_width * Lane_width - Lane_width /2.0
    else:
        offset = Lane_width /2.0 - b / frame_width * Lane_width
    direction = "Right" if offset < 0 else "Left"
    return offset,direction
    return deviation, direction
```

We use this value and put it in if condition to calculate the offset according to frame width, From the calculated value we say whether the vehicle is offset to left or right from the center. Here the values are real world values so we multiplied the lane width value to offset.

3.8 ADDING TEXT

AddText function is used to add certain instructions. These instructions will be displayed in the output frames.

```
Def addText (img, radius, direction, deviation, devDirection):
```

This function receives five arguments. Image, direction, radius, deviation, deviation direction.

```
    font = cv2.FONT_HERSHEY_TRIPLEX
```

Initially we have declared a variable font and stored the font type. So wherever the font type is needed, we can just use the variable font instead of cv2.FONT_HERSHEY_SIMPLEX.

```
    if (direction == 'Straight'):
        text1 = 'Stay Straight'
    elif(direction == 'Left Curve' and deviation<2):
        text1 = 'Slightly Turn Left '
    elif(direction == 'Right Curve' and deviation<2):
        text1 = 'Slightly Turn Right '
    elif(direction == 'Left Curve' and deviation>2):
        text1 = 'Turn Left ahead'
    elif(direction == 'Right Curve' and deviation>2):
        text1 = 'Turn Right ahead '
    else:
        text1 = str (direction)
```

Based on the value of the direction, the . if else ' conditional statement will save the instruction on the variable "text1".

```
    if round(deviation) > threshold:

        text2='Stay in Lane'

    elif round(deviation) > 1.2:,

        text2='Changing Lane'

    else:

        text2='Good Lane Keeping'
```

The same process of comparing with the conditional statement happens for the deviation also. We are storing the instruction in the variable "text2".

```
    fps_text = str(round(fps)) + 'fps'
```

We are also showing fps on the screen, but before that we are rounding the values.

```
    cv2.putText(img,'LANE STATUS : ',(50,100),font,0.8,(255,255,255),2)

    cv2.putText(img,text2,(260,100),font,0.8,(85,34,214),1,cv2.LINE_AA)

    cv2.putText(img,'DRIVER ASSISTANCE :',(50,150),font,0.8,(255,255,255),2)

    cv2.putText(img,text1,350,150), font, 0.8, (153,0,0), 1, cv2.LINE_AA)
```

We are showing the text on the screen using this cv2.putText providing the image frame and text to be printed with specific font size, font face, color and thickness. We have already stored the instructions on the variable text1 and text2.

```
    cv2.putText(img,"OFFCENTRE : ",(50,200),font,0.8,(255,255,255),2)

    deviation_text =  str(round(abs(deviation), 3)) + 'm' + ' to the ' + d
evDirection

    cv2.putText(img, deviation_text, (230, 200), cv2.FONT_HERSHEY_TRIPLEX,
 0.8, (153,0,0), 1, cv2.LINE_AA)
```

The next instruction to be printed on the screen is the offcentre which is indicating the centre of the road and calculating the value to store it in the deviation_text and print it on the screen.

```
    return img
```

Finally, we are returning the image frame with the instruction texts printed to the function call.

3.9 CALLING ALL THE DEFINED FUNCTIONS

Now we have created all the functions which are necessary to detect the lane line and warning system. Address all the functions in the main program and display the final output.

```
while True:

    ret,frame = video.read()
    if not ret:
        video = cv2.VideoCapture("F:/lane9.mp4")
        continue

    start = timer()
    frame1 = cv2.resize(frame,(1000,700),interpolation=cv2.INTER_AREA)
    img_size = (frame1.shape[1], frame1.shape[0])
    width,height = img_size
    frame_width,frame_height = (frame.shape[1],frame.shape[0])
    canny,thresh = Processing(frame1)
    birdeye,minv = Perspective_Transformation(thresh)

    line,mid,leftlane,rightlane,left_fit,right_fit,fit_left,fit_right,
output,leftx,rightx = window_search(birdeye)

    ym_per_pix = 30 / (frame_height/input_scale)
    xm_per_pix = 3.7 / (700/input_scale)

    curve_radius,direction_1=radius_of_curvature(line,fit_left,fit_right)
    deviation,direction_2 = offCenter(mid,leftlane,rightlane)
    mean,result = draw_lane_lines(frame1,thresh,leftx,rightx,fit_left,fit_
right,line,deviation)

    end = timer()
    fps = 1/(end - start)

    final = addText(result,curve_radius,direction_1,deviation,direction_2)

cv2.imshow("Frame",result)
    if cv2.waitKey(1) == ord('q'):
        break
video.release()
cv2.destroyAllWindows()
```

Using all functions and displaying the output in the original frame. We also calculated the frame per second rate to see how fast the program executes.

3.10 FINAL SOLUTION

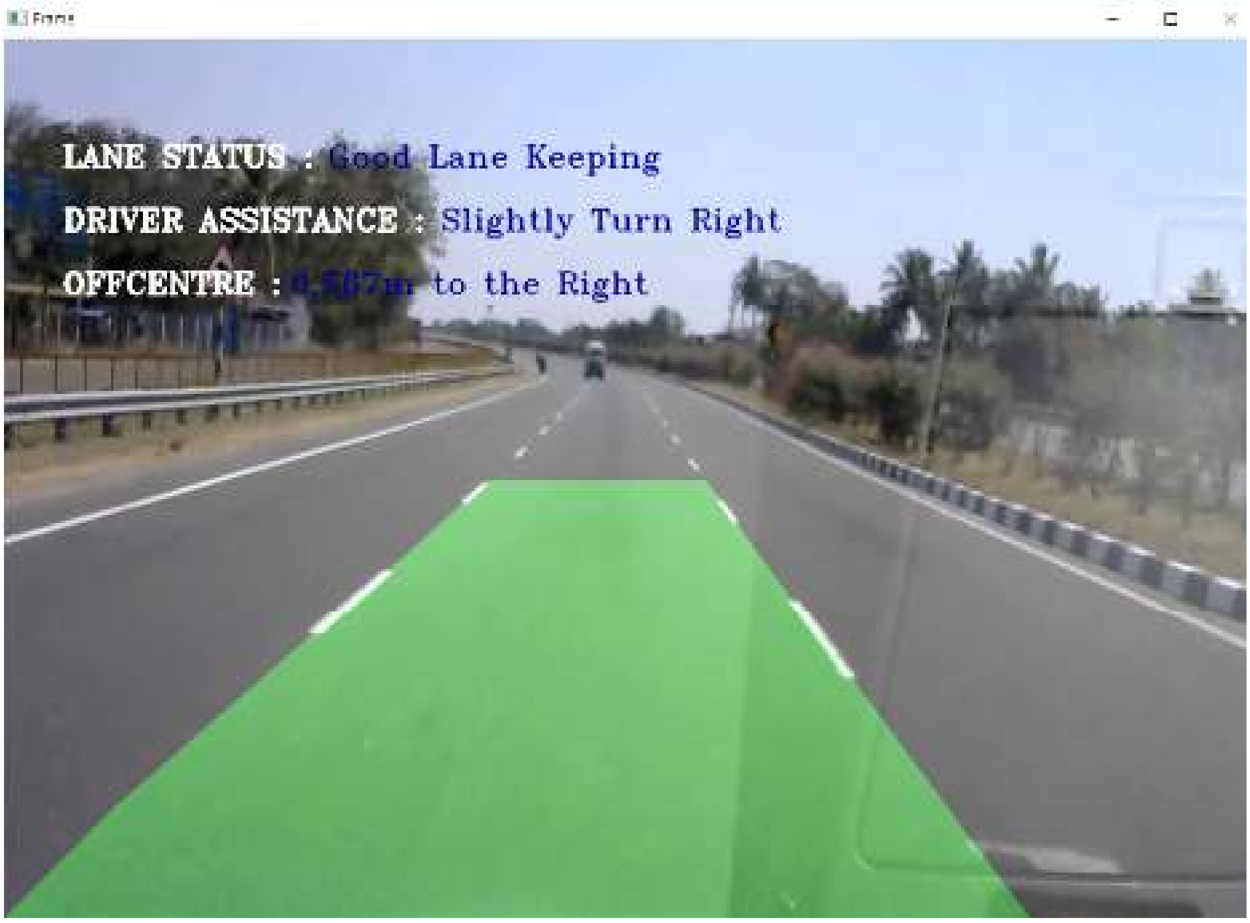

Frame
LANE STATUS : Good Lane Keeping
DRIVER ASSISTANCE : Slightly Turn Right
OFFCENTRE : [illegible]m to the Right

Fig 3.5 Real time output (LDWS)

CHAPTER 4

ALGORITHM IMPLEMENTATION PROCESS OF EDAS

4.1 IMPORTING NECASSARY LIBRARIES

```
import cv2
import os
from keras.models import load_model
import numpy as np
from pygame import mixer
import time
import dlib
from imutils import face_utils
```

We are importing all the required libraries to be used in the entire program.

4.2 SETTING PARAMETERS

```
mixer.init()
sound = mixer.Sound('alarm.wav')
```

Mixer is a predefined module in pygame library, init() function will initialize the mixer module. mixer.Sound() will create a new Sound object from ‘alarm.wav’ file.

```

face = cv2.CascadeClassifier('haar cascade files\haarcascade_frontalface_a
lt.xml)
leye = cv2.CascadeClassifier('haar cascade files\haarcascade_lefteye_2spli
ts.xml)
reye = cv2.CascadeClassifier('haar cascade files\haarcascade_righteye_2spl
its.xml')
```

Haarcascade files for face, left and right eyes have been downloaded and imported.

```

predictor=dlib.shape_predictor('models/shape_predictor_68_face_landmarks.dat')

```

The dlib.shape_predictor function will identify the 68 face landmarks from the 'models/shape_predictor_68_face_landmarks.dat' file.

```
lbl=['Close','Open']
```

The labels for the two states are open and close of the eye.

```
model = load_model('models/cnncat2.h5')
```

4.3 PROCESSING

The pre-trained model for predicting eye status is imported.

```
cap = cv2.VideoCapture("1.mp4")
```

Input video is being read frame by frame and stored in ‘cap’.

```
codec = cv2.VideoWriter_fourcc(*'XVID')
```

Videowriter_fourcc is used to save output video along with fps, width and height of frame.

```
video_fps =int(cap.get(cv2.CAP_PROP_FPS))
video_width,video_height = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)), int(cap
.get(cv2.CAP_PROP_FRAME_HEIGHT))
```

The cap.get(cv2.CAP_PROP_FPS) function will return the fps of video.The cap.get(cv2.CAP_PROP_FRAME_WIDTH) and cap.get(cv2.CAP_PROP_FRAME_HEIGHT) will return the width and height of the frame respectively.

```
out = cv2.VideoWriter('results.avi', codec, video_fps, (video_width, video
_height))
```

The output video is saved as 'results.avi' using cv2.VideoWriter.

4.4 DROWSINESS DETECTOR

```
font = cv2.FONT_HERSHEY_COMPLEX_SMALL
```

The font of the text to be printed.

```
score=0
```

Scores are the variable that stores the status of the eye.

```
thicc=2
rpred=[99]
lpred=[99]
```

These are the variables to store model output for right and left eye status.

```
YAWN_THRESH = 30
while(True):
    ret, frame = cap.read()
    height,width = frame.shape[:2]
```

The white loop for reading input video. The input video is read and stored in a variable. You can find the height and width of the frame is found.

```
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

The input frame is converted to gray scale.

```
      faces = face.detectMultiScale(gray,minNeighbors=5,scaleFactor=1.1,minSi
ze=(25,25))
      left_eye = leye.detectMultiScale(gray)
      right_eye =  reye.detectMultiScale(gray)
```

The haarcascade files are fitted to the input.

```
cv2.rectangle(frame, (0,height,50) , (200,height) , (0,0,0) , thickness=cv
2.FILLED )
```

A rectangle is drawn to the entire frame.

```
for (x,y,w,h) in faces:
        cv2.rectangle(frame, (x,y) , (x+w,y+h) , (100,100,100) , 1 )
```

A rectangle is drawn to the detected face using the output of the haarcascade_ frontalface_alt.

```
for (x,y,w,h) in right_eye:
        r_eye=frame[y:y+h,x:x+w]
```

The coordinates of the right eye are extracted from the output of Haarcascade_righteye_2splits.

```
        r_eye = cv2.cvtColor(r_eye,cv2.COLOR_BGR2GRAY)
        r_eye = cv2.resize(r_eye,(24,24))
        r_eye= r_eye/255
        r_eye=  r_eye.reshape(24,24,-1)
```

The obtained pixel coordinates are converted to gray scale, resized ,normalized and then reshaped according to the input shapes required by the CNN model.

```
        r_eye = np.expand_dims(r_eye,axis=0)
```

The (24,24,-1) image will be reshaped into (1,24,24,-1) so that it can be inputted into the model.

```
        rpred = model.predict_classes(r_eye)
        if(rpred[0]==1):
            lbl='Open'
        if(rpred[0]==0):
            lbl='Closed'
        break
```

The processed image will be passed to the model using model.predict and the output will be stored in rpred. If rpred is 1 then eye is open else if rpred is 0 then eye is closed. The program will come out of the loop.

```
       for (x,y,w,h) in left_eye:
        l_eye=frame[y:y+h,x:x+w]
        count=count+1
        l_eye = cv2.cvtColor(l_eye,cv2.COLOR_BGR2GRAY)
        l_eye = cv2.resize(l_eye,(24,24))
        l_eye= l_eye/255
        l_eye=l_eye.reshape(24,24,-1)
        l_eye = np.expand_dims(l_eye,axis=0)
        lpred = model.predict_classes(l_eye)
        if(lpred[0]==1):
            lbl='Open'
        if(lpred[0]==0):
            lbl='Closed'
        break
```

The above steps are repeated to predict the status of the left eye.

4.5 YAWN DETECTOR

Two constants ie.. score and threshold. score is incremented or decremented based on the lpred and rpred values.

```
if(score<0):
        score=0
    cv2.putText(frame,'Score:'+str(score),(100,height-
20), font, 1,(255,255,255),1,cv2.LINE_AA)
    if(score>8):
        try:
            sound.play()
        except:  # isplaying = False
            pass
```

The eye aspect ratio is tested to see if it is less than the “blink/closed” eye ratio. If it is, we increase the score (the total number of consecutive frames where the person has had their eyes closed). We assume the person is starting to doze off if the score exceeds the threshold value (ie..8). Another search was made to see if the alarm is switched on; if it wasn't, it is turned on.

```
    rects = detector.detectMultiScale(gray, scaleFactor=1.1,minNeighbors=5
, minSize=(30, 30),flags=cv2.CASCADE_SCALE_IMAGE)
```

The haarcascade_frontalface_alt file is used to detect the coordinates of the face.

```
        for (x, y, w, h) in rects:

        rect = dlib.rectangle(int(x), int(y), int(x + w), int(y + h))
```

Using the obtained coordinates a rectangle is drawn to the entire face.

```
shape1 = predictor(gray, rect)
```

This line will return to the facial landmarks within the given rectangle.

```
shape = face_utils.shape_to_np(shape1)
```

Convert the dlib shape object to a NumPy array with shape *(68, 2)*.

```
        top_lip = shape[50:53]
        top_lip = np.concatenate((top_lip, shape[61:64]))
```

The first line will give the coordinates of the upper part of the top lip, in the next line the coordinates of the lower part of the top lip will be added.

```
        low_lip = shape[56:59]

        low_lip = np.concatenate((low_lip, shape[65:68]))
```

Similarly, the coordinates for the lower lip will be obtained.

```
        top_mean = np.mean(top_lip, axis=0)
        low_mean = np.mean(low_lip, axis=0)
```

Mean of both top lip and bottom lip will be calculated.

```
        distance = abs(top_mean[1] - low_mean[1])
```

The distance between the top lip and bottom lip will be calculated.

```
        score_lip = distance + 10
        lip = shape[49:60]
        cv2.drawContours(frame, [lip], -1, (0, 255, 0), 1)
```

The first line will return the entire outer boundary of the lip.The next line will draw the contours around the detected lip pixels.

```
if (score_lip > YAWN_THRESH):
            cv2.putText(frame, "Yawn Alert", (50, 30),cv2.FONT_HERSHEY_SIM
PLEX, 0.7, (0, 0, 255), 2)

            if (thicc < 16):
                thicc = thicc + 2
            else:
                thicc = thicc - 2
                if (thicc < 2):
                    thicc = 2
            cv2.rectangle(frame, (0, 0), (width, height), (0, 0, 255), thi
cc)
```

There are also two constants included: score lip and yawn threshold. score lip for the eye, and a yawn threshold for the number of consecutive frames the eye must be below the threshold before the alarm is activated. We measure the difference between the upper and lower lips to see if it crosses the yawn threshold. If this is the case, the yawn warning message is shown.

```
 cv2.imshow('frame',frame)
    out.write(frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
out.release()
cv2.destroyAllWindows()
```

Displaying the output frame on our screen.

4.6 FINAL SOLUTION

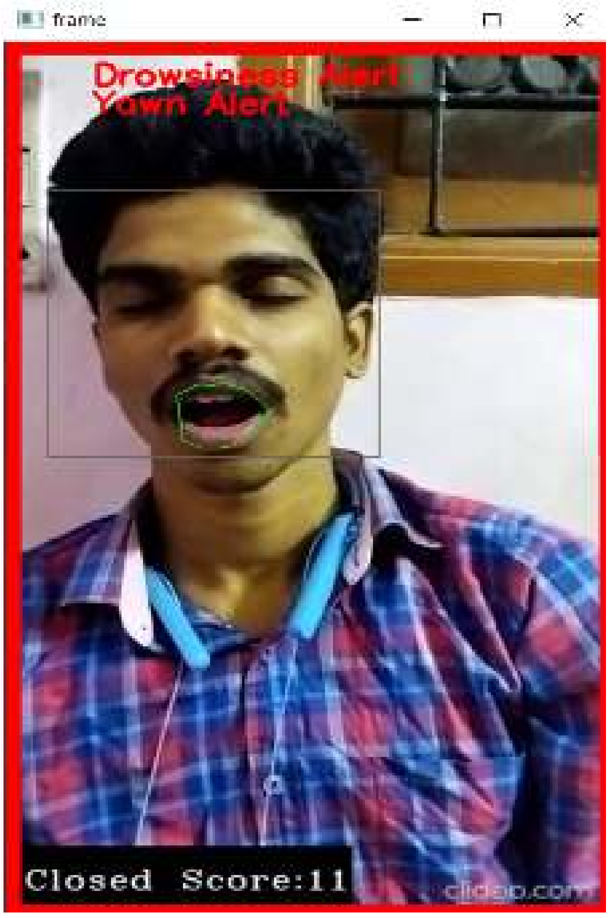


Fig 4.1 Real time output (EDAS)

CHAPTER 5

HARDWARE USED

5.1 RASPBERRY PI 4

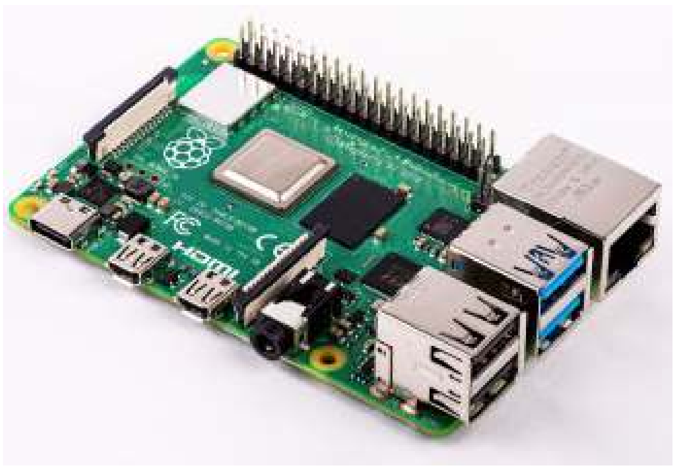


Fig 5.1 Raspberry Pi

Table 5.1 Specifications of Raspberry Pi

Dimensions	49 x 85
Processor	Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit soc @ 1.5ghz
Ram	4GB
USB Ports	2 X 3.0 Ports, 2 X 3.0 Ports
GPIO	40 pin GPIO header
HDMI Ports	2 X micro – HDMI Ports
Audio Ports	1 X 4-pole stereo audio
Graphics	Opengl ES 3.0
Power	5V DC via USB-C connector
Operating temperature	0 – 50 degrees C ambient

5.2 CAMERA

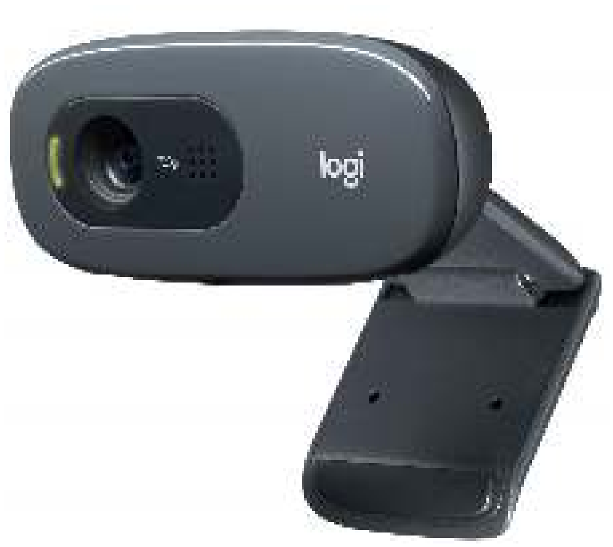


Fig 5.2 Camera

Table 5.2 Specifications of Camera

Dimensions	13.6 x 8.1 x13.6
Connector Type	USB
Video Capture Resolution	720p / 30FPS
Batteries Required	No
Batteries Included	No
Focus	Fixed
FoV	60°

5.3 DISPLAY

Fig 5.3 Display

Table 5.3 Specifications of Display

Dimensions	7 inch (194 x 110 x 20)
Viewable Screen Size	155mm x 86mm
Screen Resolution	800 x 480 Pixels
Backlight lifetime	20000 hrs
Operating temperature	-20 to +70 degree centigrade
Contrast Ratio	300
Average Brightness	250 cd/m^2

5.4 HARDWARE IMPLEMENTATION

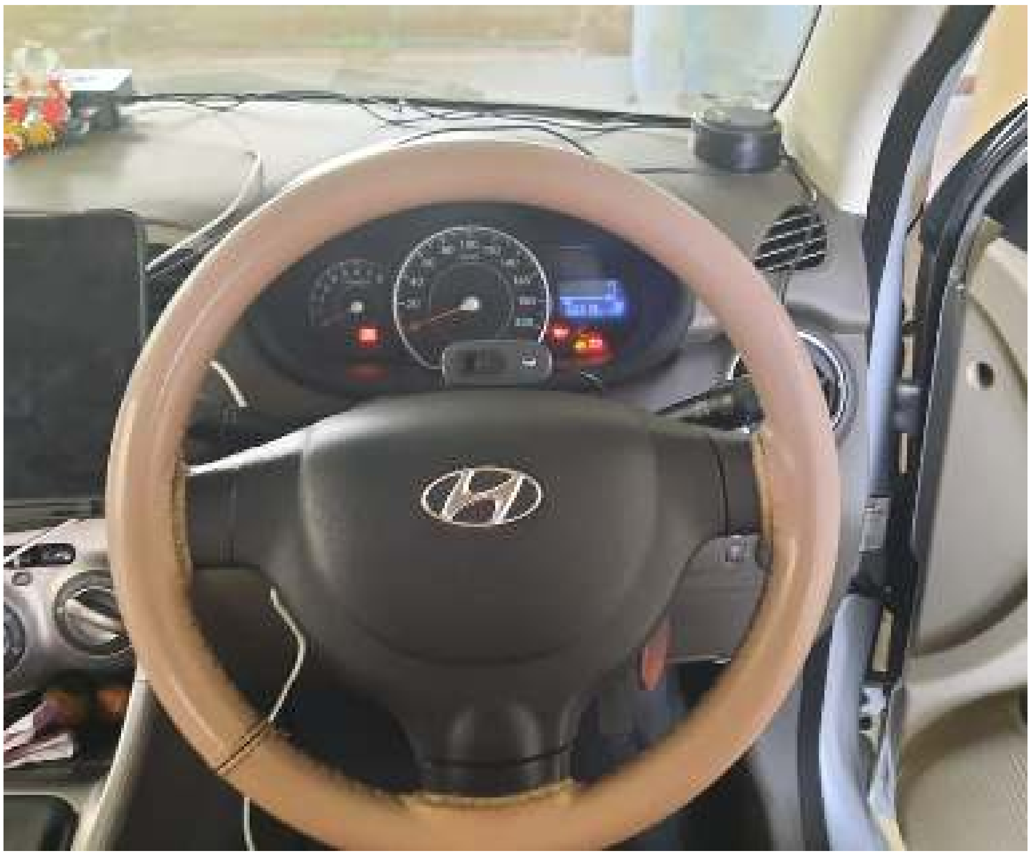


Fig 5.4 EDAS monitoring setup

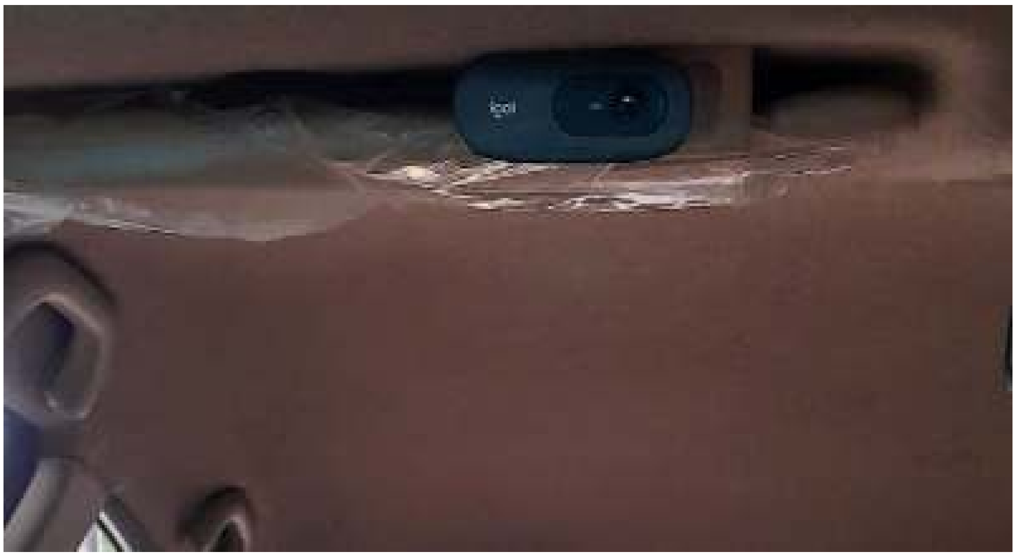


Fig 5.5 Lane finder setup

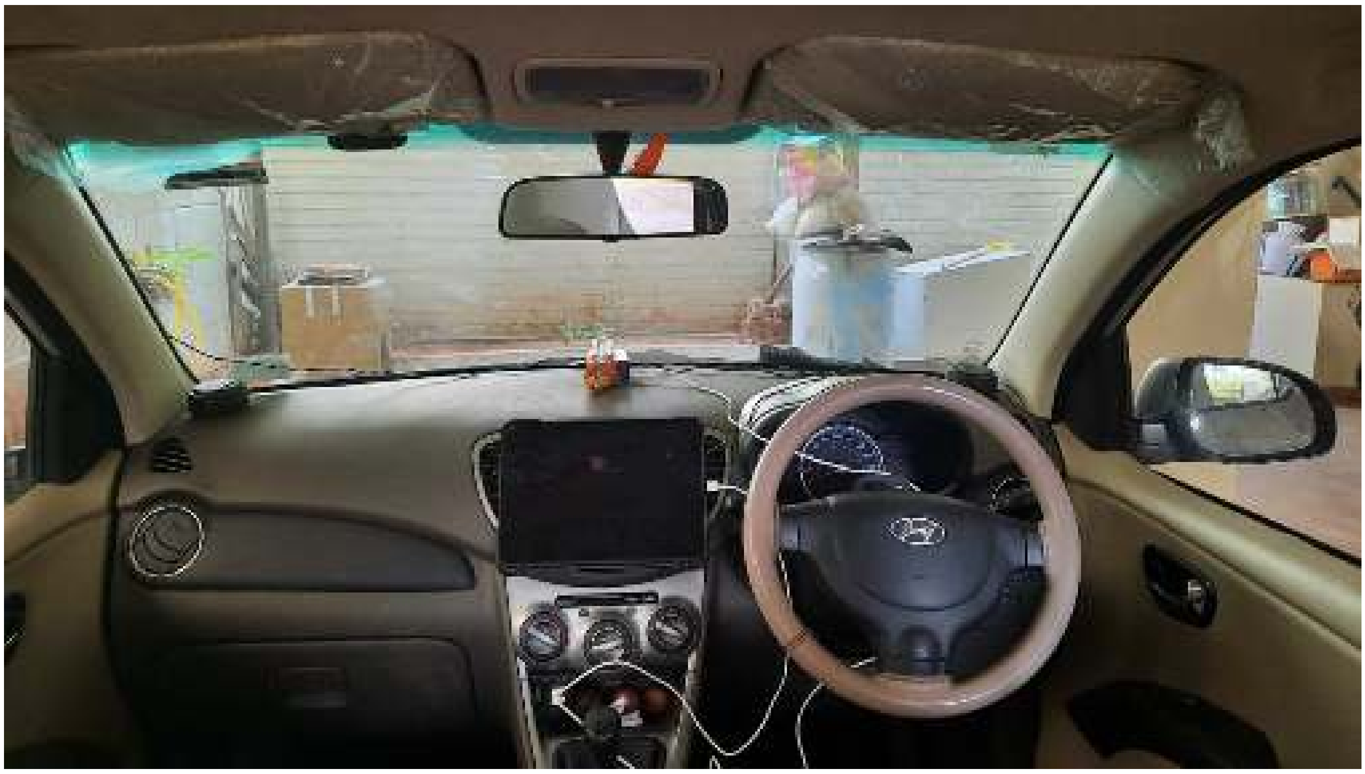


Fig 5.6 Whole setup of implementation

CHAPTER 6

CONCLUSION

We will learn the ins and outs of working in through this project. It provided us with an understanding of how a self-driving car operates as well as in-depth knowledge of the field. We were able to put the Lane Departure Alert System and EDAS into effect. The most significant takeaway was that we were able to turn our ideas into a real-time solution, gaining skills to automate the car in the process.

BIBLIOGRAPHY

1. [Nawal Alioua], [Aouatif Amine, and Mohammed Rziza],[driver's fatigue detection based on yawning extraction], [International Journal of Vechicular Techology], [August],[2014].
2. [Belhassen AKROUT], [Walid MAHDI],[a visual based approach for drowsiness detection], [IEEE Intelligent Vehicles Symposium],[June],[2013].
3. [Pranali Rajkapur Nagrale], [Dr.V.P. Kshirsagar], [lane detection with lane departure warning system], [International Journal of Scientific Development and Research (IJSDR)], [August], [2019].
4. [Marco Javier Flores], [Jose Maria Armingol, Arturo de la Escalera], [real-time drowsiness detection system for an intelligent vehicle], [IEEE],[June],[2008]
5. [Faiz Habeeb]. [K, N Kunan, N Tuturaja], [a study on lane departure warning system and object detection on roads for forward collision avoidance],[International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering],[May],[2018].