IBM

edunet foundation

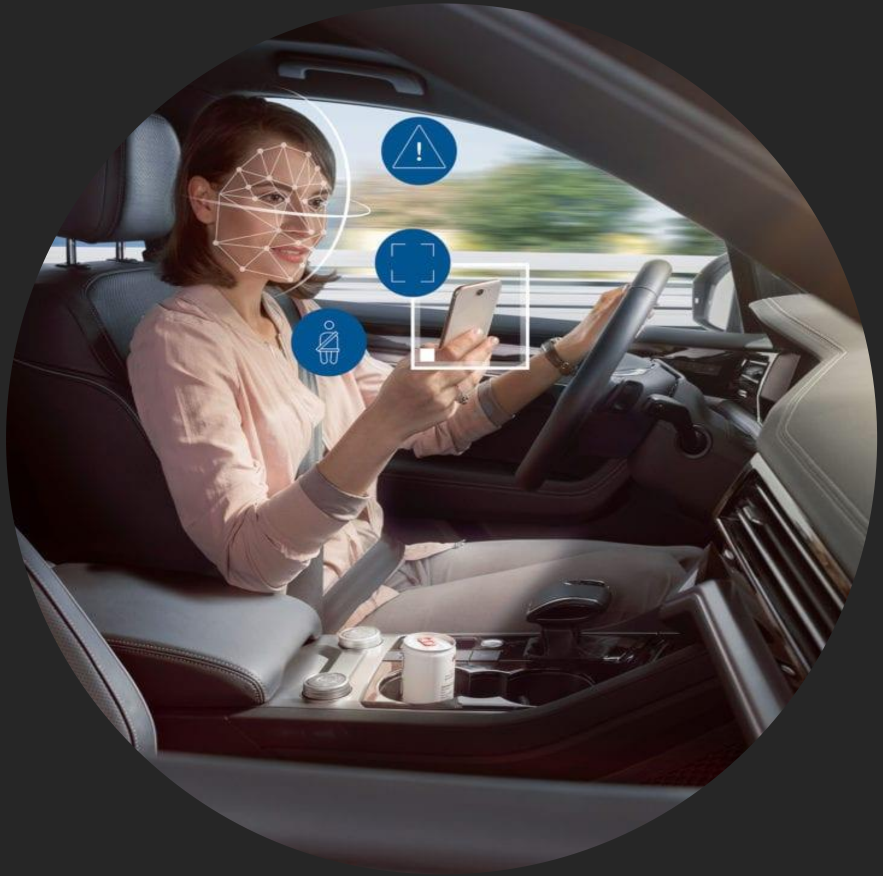
IBM Internship Project

AI-ML

Driver Attetiveness System

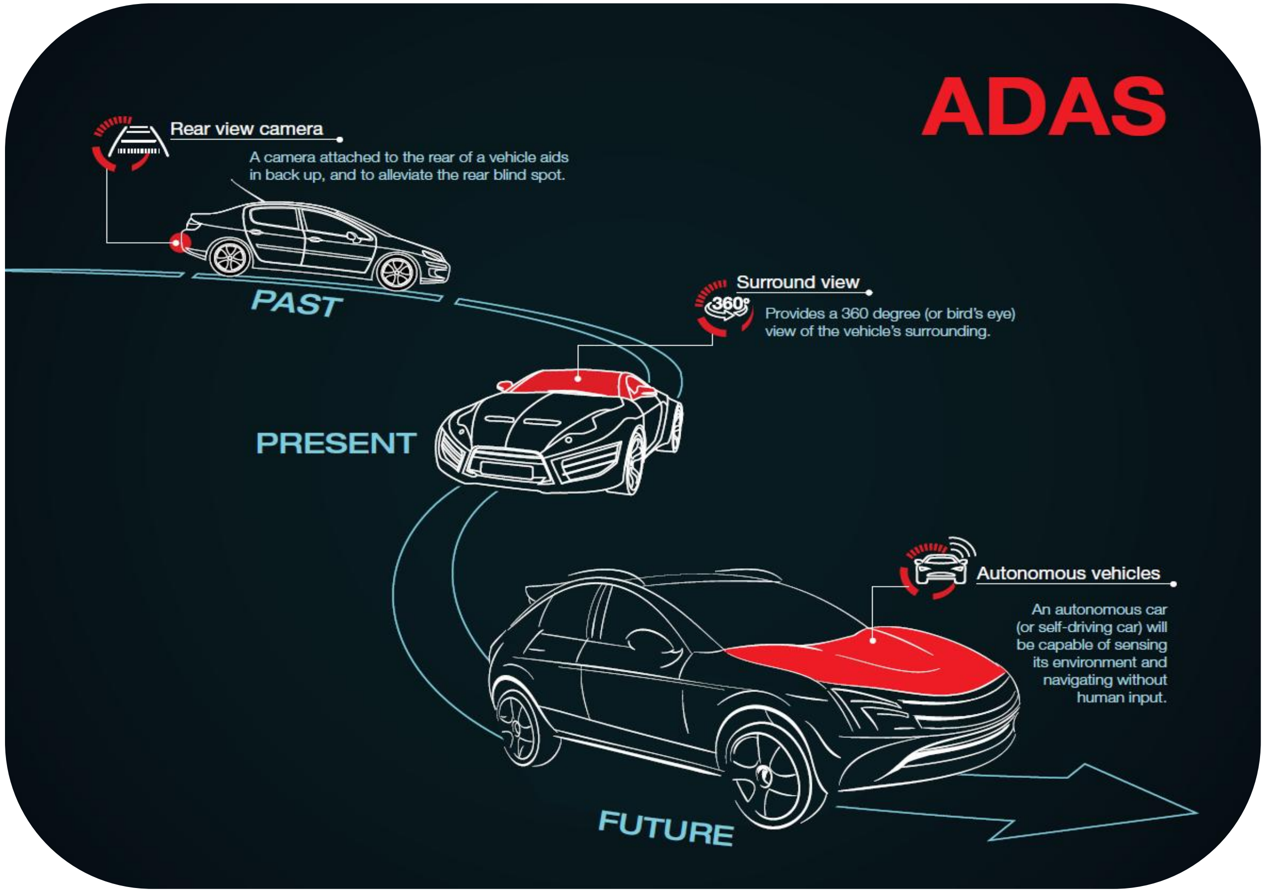
Atharva Taras
TE - AISSMS Institute of Information Technology
Email - atharvataras26@gmail.com
AICTE ID - STU61529c5a893791632803930

GitHub Reopsitory - https://github.com/AtharvaTaras/Driver-Attentiveness-System
Documentation - https://github.com/AtharvaTaras/Driver-Attentiveness-System#readme



CONTENTS

ABOUT THE PROJECT



Agenda

The agenda for a driver assistance system is an introduction to the topic, a discussion of current technologies and features, an update on research and development, a review of use cases, and a discussion of future developments.

Overview

Advanced Driver Assistance Systems (ADAS) are technologies designed to improve the safety and comfort of vehicle operation. These systems use a combination of sensors, such as cameras, radar, and lidar, to detect and respond to various driving scenarios. These technologies can help drivers avoid accidents, , and make the driving experience more enjoyable.

END USERS

Drowsiness detection is a specific feature of Advanced Driver Assistance Systems (ADAS) that is designed to detect when a driver is becoming fatigued and alert them to take a break. This feature can be beneficial for all types of drivers, regardless of their age, experience level, or occupation.

For Example

Novice drivers: New drivers may not have the experience to recognize when they are becoming drowsy and may not know when to take a break. Drowsiness detection can alert them to take a break before they become too fatigued to drive safely.

Elderly drivers: As we age, our ability to stay alert decreases, and drowsiness detection can help older drivers recognize when they need to take a break.

Distracted drivers: Drowsiness detection can help drivers stay alert even if they are becoming distracted by other tasks or activities while driving.

Professional drivers: Long-distance truck drivers or delivery drivers may spend many hours on the road and may need reminders to take a break and rest. Drowsiness detection can help them stay safe and avoid accidents caused by fatigue.

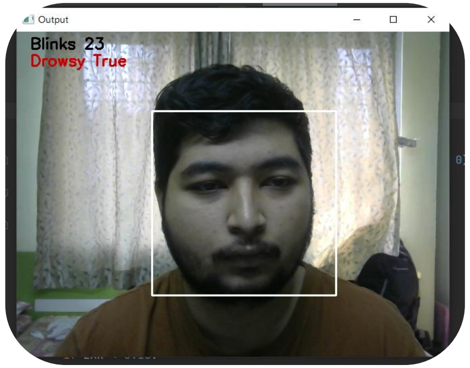
People with disabilities: Drowsiness detection can help people with mobility or cognitive impairments stay alert and safe on the road, especially if they need to drive for long periods of time.

HOW IT WORKS

Flow Modelling

Step II

Track and exctract the facial region from the video

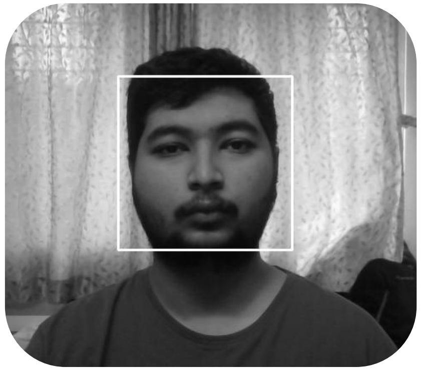


Step IV

Use euclidean distance between points to check if eyes are close and conclude accordingly

Step I

Identify and detect face from video feed



Step III

Identify features and plot landmarks on the face

DLib

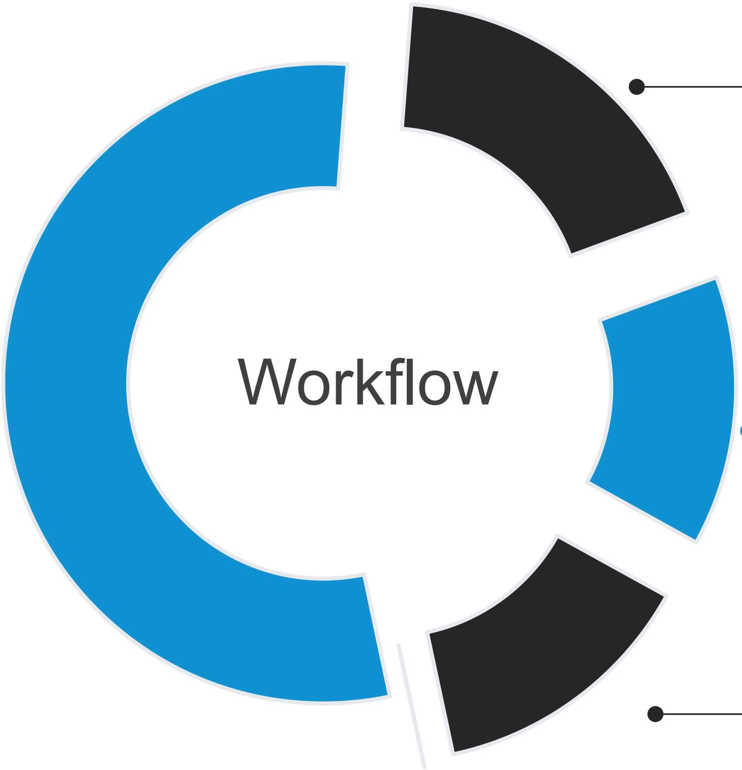
NumPy

OpenCV

SciPy

MODELLING

How it works



60%

OpenCV

Used to get input video, add objects on top of the video and convert video into different sizes and colour spaces

20%

HAAR Cascades

Fastest method to detect and track a face accurately in a video

15%

Dlib

Uses pre-trained model to plot landmarks on the face

15%

SciPy

Calculates average euclidean sidtance between landmarked pints to detect if eyes are closed

```python
def find_face() -> list:

    ret, frame = VID.read()

    if ret:
        grayscale = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        face_coordinates = HAAR_DATA.detectMultiScale(grayscale)

        if len(face_coordinates) == 1:
            x, y, w, h = [each for each in face_coordinates[0]]
            # return [x, y, x + w, y + h]  # Face Mask

            subframe = grayscale[y: y + h, x: x + w]

        else:
            x, y = 0, 0
            w = grayscale.shape[1]
            h = grayscale.shape[0]
            subframe = grayscale


        if DEBUG_FACE:
            print(len(face_coordinates), face_coordinates)
            # print(f'X {x} Y {y}, W {w}, H {h}')
            cv2.imshow('Raw Input', frame)
            cv2.rectangle(img=grayscale,
                          pt1=(x, y),
                          pt2=(x+h, y+w),
                          thickness=2,
                          color=(255, 255, 255))
            cv2.imshow('Grayscale', grayscale)
            cv2.imshow('Extracted Face', subframe)
            cv2.waitKey(1)
            # print(face_coordinates)

        return [frame, subframe, [x, y, w, h]]
```

```python
def calculate_EAR(eye) -> float:
    A = distance.euclidean(eye[1], eye[5])
    B = distance.euclidean(eye[2], eye[4])
    C = distance.euclidean(eye[0], eye[3])
    ear_aspect_ratio = (A + B) / (2.0 * C)
    return ear_aspect_ratio


def check_blink(grey_frame) -> bool:
    blink = False
    faces = DETECTOR(grey_frame)

    for face in faces:

        face_landmarks = PREDICTOR(grey_frame, face)
        leftEye = []
        rightEye = []

        for n in range(36, 42):
            x = face_landmarks.part(n).x
            y = face_landmarks.part(n).y
            leftEye.append((x, y))
            next_point = n + 1
            if n == 41:
                next_point = 36
            x2 = face_landmarks.part(next_point).x
            y2 = face_landmarks.part(next_point).y

            if DEBUG_BLINK:
                cv2.line(grey_frame, (x, y), (x2, y2), (150, 150, 0), 2)

        for n in range(42, 48):
            x = face_landmarks.part(n).x
            y = face_landmarks.part(n).y
```

```
        x2 = face_landmarks.part(next_point).x
        y2 = face_landmarks.part(next_point).y

        if DEBUG_BLINK:
            cv2.line(grey_frame, (x, y), (x2, y2), (150, 150, 0), 2)

    for n in range(42, 48):
        x = face_landmarks.part(n).x
        y = face_landmarks.part(n).y
        rightEye.append((x, y))
        next_point = n + 1
        if n == 47:
            next_point = 42
        x2 = face_landmarks.part(next_point).x
        y2 = face_landmarks.part(next_point).y

        if DEBUG_BLINK:
            cv2.line(grey_frame, (x, y), (x2, y2), (150, 150, 0), 2)

    if DEBUG_BLINK:
        cv2.imshow('Eyes', grey_frame)
        cv2.waitKey(1)

    left_ear = calculate_EAR(leftEye)
    right_ear = calculate_EAR(rightEye)

    EAR = (left_ear + right_ear) / 2
    EAR = round(EAR, 2)

    if EAR < 0.18:
        blink = True
        sleep(0.05)

    return blink
```

```
def calibrate(duration: float) -> list:
    ret, frame = VID.read()

    if ret:
        grey = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = DETECTOR(grey)
        # frame = cv2.resize(frame, dsize=(frame.shape[1] * 2, frame.shape[0

        lt_eye = []
        rt_eye = []

        start = time()

        while True:

            for face in faces:
                landmarks = PREDICTOR(grey, face)

                points = []

                for n in range(0, 68):

                    x = landmarks.part(n).x
                    y = landmarks.part(n).y
                    # Appending Y coordinates only since we need vertical eu
                    points.append(y)
                    # print(points)

                    if ((n > 35) and (n < 48)) or ((n > 47) and (n < 68)):

                        if DEBUG_LANDMARKS:
                            cv2.circle(frame,
                                       center=(x, y),
                                       radius=2,
```

WOW FACTORS

What makes this solution different?

Realtime

This system can run at realtime, 30 FPS on a single CPU core

Modularity

The code is highly reusable and modular. The parameters are easy to set, update and configure.

Sensors

There is no need for any extra sensors, the whole system can work using a camera only.

THANK YOU