DRIVER DROWSINESS DETECTION SYSTEM

A Project Report

Project Report submitted in partial fulfillment of the requirements for the award of degree of B.Tech. in Electronics & Instrumentation Engineering under
Institute of Engineering & Technology, Lucknow

by

Shishir Bhargav (1900520320051)
Prachi Singh (1900520320035)
Shani Yadav (1900520320048)

Under the Guidance of

Er. Richa Pathak

(Assistant Professor, Electronics & Instrumentation Engineering)

DEPARTMENT OF ELECTRONICS ENGINEERING
IET, LUCKNOW, U.P.
JUNE -2023

ELECTRONICS AND INSTRUMENTATION ENGINEERING

INSTITUTE OF ENGINEERING AND TECHNOLOGY, LUCKNOW

An Autonomous Constituent Institute Of Dr. A.P.J.Abdul Kalam Technical University, U.P., Lucknow

CERTIFICATE

This is to certify that the project entitled "**DRIVER DROWSINESS DETECTION SYSTEM**" is a bonafide work of "Shishir Bhargav (1900520320051), Shani Yadav (1900520320048) and Prachi Singh (1900520320035)" submitted in partial fulfillment of the requirements for the award of the degree of BACHELOR OF TECHNOLOGY IN ELECTRONICS AND INSTRUMENTATION ENGINEERING, at INSTITUTE OFENGINEERING & TECHNOLOGY.

This is partial fulfillment of the requirements for the award of degree of Bachelor of Technology in ELECTRONICS AND INSTRUMENTATION ENGINEERING under the department of ELECTRONICS AND COMMUNICATION ENGINEERING, from INSTITUTE OF ENGINEERING AND TECHNOLOGY (IET) Sitapur Road, Lucknow, Uttar Pradesh.

ER. RICHA PATHAK
(Assistant Professor
Electronics & Instrumentation
Engineering
IET, LUCKNOW

DR. R.C.S. CHAUHAN
CONVENER
Electronics &Instrumentation
Engineering
IET, LUCKNOW

ACKNOWLEDGEMENTS

I would like to express my deep and sincere gratitude to my guide(s), Er. Richa Pathak, of Institute of Engineering and Technology for her unflagging support and continuous encouragement throughout the project work. Without her guidance and persistent help this report would not have been possible.Without her guidance and persistent help this report would not have been possible.

At last I must express my sincere heartfelt gratitude to all the staff members of the Electronics Engineering Department who helped us directly or indirectly during this course of work.

The Udemy course on “Driver Drowsiness Detection System” was very helpful to us in giving the necessary background information and inspiration in choosing this topic for the project.

Finally, we express our gratitude to all other members who are involved either directly or indirectly for the completion of this project.

DECLARATION

We, hereby, declare that this project report entitled '**DRIVER DROWSINESS DETECTION SYSTEM'** using Python Libraries and Android App with Google Vision API Integration" is a result of our original work and has not been submitted elsewhere for any other purpose.

I affirm that the content presented in this report represents my own findings, ideas, and analysis, derived from extensive research and development efforts. Any external sources of information utilized in this project have been duly acknowledged and referenced.

I take full responsibility for the accuracy, authenticity, and integrity of the content presented in this report. I have adhered to ethical practices during the project development, ensuring the protection of participants' rights and privacy.

Date: 08/06/2023

Signature :
NAME: PRACHI SINGH
ROLL NO.: 1900520320035

Signature :
NAME: SHISHIR BHARGAV
ROLL NO.: 1900520320051

Signature :
NAME: SHANI YADAV
ROLL NO.: 1900520320048

ABSTRACT

The report proposed the results and solutions on the limited implementation of the various techniques that are introduced in the project. Whereas the implementation of the project gives the real-world idea of how the system works and what changes can be done in order to improve the utility of the overall system.

The Driver Drowsiness Detection System is an essential application in today's era of increasing road accidents caused by drowsy drivers. This project focuses on the development of a robust and efficient system that utilizes Python programming language, Google Vision API, and the SeekBar component in Android for real- time detection of driver drowsiness.

The proposed system employs computer vision techniques to monitor the driver's facial features and detect signs of drowsiness. By utilizing OpenCV and dlib libraries in Python, the system analyzes live video feeds from a camera installed inside the vehicle. It identifies crucial indicators of drowsiness, such as eye closure duration, yawning, and head position changes. The proposed system utilizes machine learning algorithms to analyze the sensor data and classify the driver's alertness level into different categories such as alert, drowsy, or inattentive.

To enhance the usability and accessibility of the system, an Android application is developed using Java and integrated with the Python-based drowsiness detection system. The application provides a user-friendly interface that allows drivers to easily interact with the system. It incorporates the SeekBar component, enabling drivers to set their preferred sensitivity level for drowsiness detection. The proposed system utilizes a camera mounted inside the vehicle to monitor the driver's face and analyze various facial features and patterns associated with drowsiness.

The key steps involved in the drowsiness detection system include face detection, eye detection, and drowsiness classification. Based on the EAR values, the system determines whether the driver's eyes are open, closed, or in a drowsy state. If the eyes are closed for an extended period or the EAR drops below a certain threshold, the system triggers an alert mechanism to warn the driver about their drowsiness. The alert mechanism can be customized to suit the specific requirements and preferences of the driver, such as audible alarms or visual cues.

TABLE OF CONTENT

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

The advancement of technology has opened doors to more sophisticated solutions that enhance our quality of life. Each year, an alarming number of approximately 100,000 reported crashes involve drowsy driving, and this figure is likely much higher. Facial expressions provide valuable insights into various physiological conditions within the body. Numerous algorithms and techniques are available for face detection, serving as the fundamental building block for further analysis. One particular global issue that demands attention is drowsiness among drivers. To address this problem, a solution involves tracking the eyes to detect signs of drowsiness and classify a driver as drowsy. For real-time application, a camera is mounted on the car's dashboard to capture the driver's face in the input video. Using the Dlib model, which is trained to identify 68 facial landmarks, specific indicators of drowsiness are extracted. If drowsiness is detected, the driver is promptly alerted. Substantial research has been conducted in the field of driving safety, aiming to minimize the number of accidents occurring on the roads.

The primary objective of this project is to create a robust and efficient driver drowsiness detection system that can accurately identify drowsiness in real-time and promptly alert the driver to prevent accidents. By leveraging the power of Python libraries and integrating them with the Android platform and Google Vision API, this system aims to provide a comprehensive and user-friendly solution. The Android application acts as the user interface and connects with the driver drowsiness detection system. It captures video streams from the device's camera and processes them using OpenCV. The Google Vision API comes into play for advanced facial analysis, leveraging its machine learning models to identify specific facial cues indicative of drowsiness.

The Driver Drowsiness Detection System project is designed to address the critical issue of drowsy driving and improve road safety. By utilizing Python libraries, an Android app with the SeekBar component, and integration with the Google Vision API, the system aims to provide a comprehensive solution for real- time drowsiness detection. The project methodology encompasses data collection, algorithm development, alert mechanisms, sensitivity adjustment, and thorough testing to ensure the creation of a robust and efficient system that can effectively identify driver.

CHAPTER 2
METHODOLOGY

The proposed approach for detecting driver fatigue operates on two levels. The process commences with the camera capturing live video frames, which are subsequently transmitted to a local server. On the server, the Dlib library is employed to identify facial landmarks, and a threshold value is utilized to determine the driver's level of sleepiness.The Eye Aspect Ratio (EAR) is computed using the identified facial landmarks and provided as output to the driver. In our system, the obtained EAR value is compared against a threshold of 0.25. If the EAR value falls below this threshold, indicating drowsiness, an alarm is activated to alert both the driver and passengers. Eye aspect ratio (EAR) calculation played a crucial role in determining the level of drowsiness. The dlib library was employed to extract landmarks from the driver's eyes, enabling the calculation of EAR. By measuring the ratio of distances between specific eye landmarks, changes indicative of drowsiness, such as eye closure, could be identified. The collected video frames are then preprocessed by extracting them from the captured video streams in the Android application. Preprocessing techniques such as resizing, normalization, and grayscale conversion are applied to enhance the performance of the algorithms. Next, OpenCV's face detection algorithms, which can be based on Haar cascades or deep learning models, are utilized to locate and extract the driver's face region in each frame. This step ensures accurate face detection, even under challenging lighting conditions or when the driver's head orientation varies. To further enhance the drowsiness detection capabilities, the system integrates the Google Vision API. The API provides advanced facial analysis capabilities and employs machine learning models to extract additional facial features and patterns associated with drowsiness. This integration complements the OpenCV-based system and improves the accuracy and robustness of drowsiness detection.

To detect drowsiness events in real-time, the EAR values were continuously monitored. When the EAR fell below a predefined threshold for a specified duration, it triggered the detection of drowsiness. In the Python program, these mechanisms involved triggering alerts such as sounding alarms, sending notifications to the driver through speakers or headphones. The system's ability to analyze facial features in real-time using Python libraries, calculate EAR, and continuously monitor drowsiness events demonstrated its effectiveness in identifying potential risks associated with drowsy driving.

CHAPTER 3
COMPONENTS

To detect drowsiness, we utilized a combination of technologies including OpenCV, Dlib, Numpy, Pygame (for playing the alarm sound), and Python. The Dlib library played a crucial role in our methodology by facilitating the detection and isolation of facial landmarks. Specifically, we employed Dlib's pre-trained facial landmark detectors, which accurately identified 68 distinct facial landmarks. These landmarks served as key reference points for our drowsiness detection approach.

3.1 PYTHON

Python, a versatile computer programming language, has gained significant popularity due to its extensive range of applications. It is commonly utilized for developing websites, software, automating tasks, and performing data analysis. Python's adaptability as a general-purpose language enables developers to create programs for diverse purposes without being restricted to specific problem domains. The accessibility and user-friendly nature of Python have contributed to its widespread adoption. It has become one of the most widely used programming languages in the industry today. In fact, a survey conducted by RedMonk, an industry analyst firm, revealed that Python was the second-most popular programming language among developers in 2021. Python's flexibility is especially evident in web development, software engineering, task automation, and data analysis. Moreover, its simplicity has made it a preferred choice for individuals outside the programming sphere, such as accountants and scientists, who utilize Python for various day-to-day activities like financial organization and scientific research. Once the models are loaded, video frames are captured using the webcam or other video sources. The video capture object is initialized, and a loop is set up to continuously retrieve frames. Each frame is then processed to detect the driver's face using the selected face detection model. If the face is detected, facial landmarks are extracted using the facial landmarks detection model. These landmarks help identify the positions of the eyes and calculate the Eye Aspect Ratio (EAR) to assess the driver's drowsiness level. To enhance the system's capabilities, additional features from the Google Vision API can be integrated. This integration can involve leveraging machine learning models provided by the API to extract more facial features and patterns associated with drowsiness.

3.2 OPENCV

OpenCV is an extensive open-source library renowned for its capabilities in computer vision, machine learning, and image processing. It has become a critical component in modern-day systems, particularly in real-time operations. With OpenCV, one can efficiently process images and videos, enabling tasks such as object detection, facial recognition, and even human handwriting analysis. Integration with various libraries, including NumPy, empowers Python to effectively analyze the OpenCV array structure for comprehensive data analysis. The initial release of OpenCV was version 1.0, and it is distributed under the permissive BSD license, making it freely available for both academic and commercial purposes. OpenCV provides interfaces for C++, C, Python, and Java, allowing seamless integration across a wide range of platforms, including Windows, Linux, Mac OS, iOS, and Android. During its design, OpenCV was primarily focused on optimizing computational efficiency for real-time applications. Consequently, the library is predominantly written in optimized C/C++ code, which leverages the benefits of multi-core processing. OpenCV's versatility and efficiency make it a powerful tool for identifying image patterns and extracting various features. This is accomplished through techniques such as vector space analysis and performing mathematical operations on these extracted features. The library's capabilities have revolutionized computer vision applications, facilitating advancements across diverse domains. OpenCV also supports real-time video processing and analysis, making it efficient for continuous monitoring of the driver's face and eyes. This real-time capability enables prompt detection of drowsiness and timely warnings to the driver, preventing potential accidents. In addition to the core functionalities, OpenCV offers a range of image processing and computer vision techniques that can enhance the drowsiness detection system. These include image filtering, edge detection, and feature extraction methods that can be applied to improve the accuracy and robustness of the system.

Overall, OpenCV's comprehensive functionality, broad language support, cross-platform compatibility, and emphasis on real-time performance have solidified its position as a go-to solution for professionals and researchers in the fields of computer vision, machine learning, and image processing. To quantify the driver's eye state, OpenCV calculates the Eye Aspect Ratio (EAR).

3.2 DLIB

Dlib is a versatile and platform-independent software library, primarily developed in the C++ programming language. Its design principles draw inspiration from concepts such as design by contract and component-based software engineering, making it a collection of independent software components. Released under the Boost Software License, Dlib is an open-source software.Since its inception in 2002, Dlib has undergone significant growth and expansion, encompassing a diverse range of tools and functionalities. As of 2016, it boasts an extensive collection of software components tailored for various tasks, including networking, thread management, graphical user interfaces, data structures, linear algebra, machine learning, image processing, data mining, XML and text parsing, numerical optimization, Bayesian networks, and more. In recent years, substantial efforts have been dedicated to developing a comprehensive suite of statistical machine learning tools. Notably

Dlib's flexibility, broad scope of functionalities, and commitment to advancing statistical machine learning have positioned it as a highly valuable toolset for diverse applications. Its widespread adoption and continued development signify its effectiveness and impact within the software development and machine learning communities.

3.3 NUMPY

NumPy is a feature-rich library that offers a multitude of capabilities for handling N-dimensional arrays. Its array-processing functionalities are particularly notable, enabling efficient numerical operations and data manipulation. The cornerstone of NumPy is the ndarray, a powerful multi-dimensional array object that facilitates the storage and manipulation of large datasets with optimal efficiency. Beyond its applications in scientific fields, NumPy also serves as an efficient and flexible container for generic data. It provides the ability to define arbitrary data types, allowing for seamless integration with diverse databases. This flexibility broadens the scope of NumPy's applications, making it a versatile tool for various data-centric tasks. With its comprehensive array-processing capabilities, NumPy empowers users to perform efficient numerical operations and data manipulations. It encompasses a wide range of mathematical functions, supports broadcasting for convenient element-wise operations, and provides robust functionality for linear algebra computations.

Consequently, NumPy becomes an indispensable tool for tasks such as data analysis, simulation, and machine learning, where efficient numerical operations and data manipulation are paramount. NumPy is utilized when implementing machine learning algorithms for drowsiness detection.

For example, if a machine learning classifier is trained to classify drowsiness based on extracted features, NumPy provides efficient methods for data preprocessing, feature extraction, and handling the input and output arrays required by the classifier.

In summary, NumPy's array-processing capabilities, extensive mathematical functions, and compatibility with various data types and databases position it as a fundamental library for efficient numerical operations, data manipulation, and advanced scientific computing.

3.4 PYGAME

Pygame is a versatile and cross-platform Python software package that is specifically tailored for video game design. Its primary focus is to provide developers with the necessary tools for creating captivating games by incorporating graphics, visuals, and sounds. Pygame encompasses a comprehensive collection of libraries that seamlessly integrate with images and sounds, enabling the creation of immersive game graphics. This feature-rich framework simplifies the game design process, making it accessible and user-friendly for beginners venturing into game development. One of Pygame's standout attributes is its intuitive interface, which empowers developers to harness its wide range of features effectively. These features encompass graphics rendering, sound manipulation, and input handling, all of which contribute to the creation of interactive and engaging games. Whether you're a novice or an experienced game developer, Pygame provides a rich development environment that fosters creativity and innovation.Another advantage of Pygame is its extensive documentation and vibrant community support. Developers can easily access valuable resources, tutorials, and examples, accelerating the learning process and fostering collaboration. This collaborative ecosystem ensures that developers can seek guidance, share ideas, and continuously enhance their game development skills.

In conclusion, Pygame serves as a versatile and user-friendly toolset for video game design. With its intuitive interface, broad range of features, and robust community support, Pygame empowers developers to bring their game ideas to life and create captivating gaming experiences.

CHAPTER 4
PROPOSED WORK

4.1 FACIAL LANDMARK MARKING

The facial landmark extraction process in our system relies on the Dlib library. Dlib utilizes a pre-trained face detector that incorporates an enhanced version of the histogram of oriented gradients (HOG) technique. It employs two shape predictor models trained on the i-Bug 300-W dataset, allowing for the localization of 68 and 5 landmark points within a facial image, respectively. By utilizing this approach, we leverage the power of 68 facial landmarks to accurately analyze facial features. The method involves capturing the frequencies of gradient direction in specific localized regions of an image, which are then used to generate histograms. This technique excels in face detection, as it effectively captures contour and edge features across various objects. To record the facial landmarks, our system employs the Facial Landmark Predictor provided by the Dlib library. This predictor plays a crucial role in calculating the lengths required for the Eye Aspect Ratio (EAR) values, a key metric for drowsiness detection. By utilizing the information from the facial landmarks, we can accurately determine and track changes in the eye region. Facial landmark marking is an essential step in a driver drowsiness detection system as it enables the identification and tracking of specific facial features relevant to assessing drowsiness levels. By locating and marking these landmarks on the driver's face, the system can extract valuable information for drowsiness analysis. Various algorithms and techniques can be employed to perform facial landmark marking. One popular approach is to use shape prediction models, such as the dlib library's pre-trained facial landmark predictor. These models are trained on large datasets and can accurately estimate the positions of key facial landmarks, including eye corners, eyebrows, nose, and mouth.

The figure below showcases the facial landmark points extracted using the Dlib library. These landmarks are pivotal in computing the EAR values, enabling the accurate assessment of drowsiness levels in individuals.

Overall, the integration of the Dlib library and its facial landmark extraction capabilities enhance our system's ability to analyze and monitor facial features, particularly in the context of drowsiness detection and eye tracking.

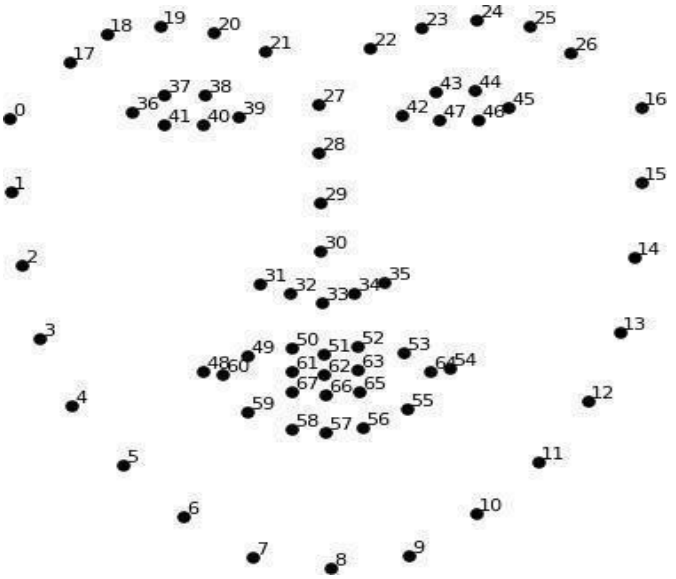


Fig. 4.1 Facial landmark marking by Dlib

Once the facial landmarks are detected and marked, they serve as reference points for subsequent calculations. In the context of drowsiness detection, the focus is typically on the eye region. By identifying the eye corners and other relevant points, it becomes possible to calculate the Eye Aspect Ratio (EAR). The EAR is a measure of eyelid openness and is computed by comparing the distances between specific eye landmarks. This ratio is used to assess the level of drowsiness, as a decrease in the EAR indicates closed or drowsy eyes. In addition to EAR calculation, facial landmark marking enables other important analyses. For instance, it allows for the measurement of head pose and gaze direction, which can provide additional insights into the driver's alertness. By tracking the positions of landmarks over time, the system can detect changes in head orientation or eye movement patterns that might indicate drowsiness or distraction.Facial landmark marking also facilitates the integration of advanced facial analysis techniques, such as emotion recognition or facial expression analysis. By accurately locating facial landmarks, it becomes possible to extract additional features related to emotions, such as changes in eyebrow position or mouth shape, which can provide insights into the driver's mental state and overall attentiveness.

In summary, facial landmark marking is a critical step in a driver drowsiness detection system.

4.2 ALGORITHM

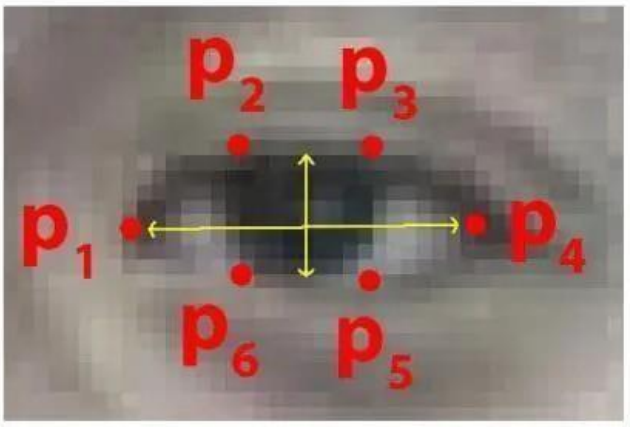


Fig. 4.2 Eye Marking

In our sleepiness detection system, the coordinates of the pupils, denoted as P1, P2, P3, P4, P5, P6, are used to calculate the Eye Aspect Ratio (EAR). When the eyes are open, the EAR is typically around 0.25, which serves as a reference value. If the EAR falls below 0.25, it indicates that the person is drowsy. The calculation of the EAR involves determining the distance between the upper and lower eyelids (|P2 - P6| + |P3 - P5|) divided by twice the horizontal distance of the eye (|P1 - P4|).

To identify driver sleepiness, we analyze the EAR values. We compute the average EAR of both the left and right eyes. The EAR value is monitored in our sleepiness detection system to determine if it goes below the threshold value and remains below it in the subsequent frames. If the EAR value decreases, it signifies that the individual has closed their eyes and is sleepy. On the other hand, if the EAR value increases again, it indicates that the person is simply blinking their eyes and is not drowsy.

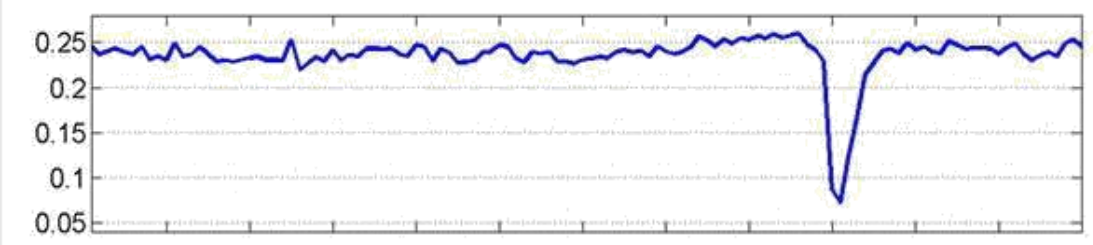


Fig. 4.3 EAR for single blink

When someone has their eyes open, the EAR (Eye Aspect Ratio) tends to stay relatively consistent. However, as the eye starts to close, the EAR approaches zero. The EAR is not heavily influenced by the secific person or the position of their head. The aspect ratio of an open eye may vary slightly among individuals, but it remains completely unaffected by changes like resizing the image or rotating the face within the same plane. In the face detection path, the system utilizes a face detection algorithm, such as Haar cascades or deep learning models, to detect and locate the driver's face in the video frames. This is depicted by a diamond-shaped decision point, where the flow diverges based on whether a face is detected or not.

In the facial landmark detection step, the system employs a shape prediction model, such as the one provided by dlib or OpenCV, to identify specific facial landmarks, such as eye corners and nose tip. These landmarks are essential for subsequent calculations and analysis. Once the landmarks are detected, the flow moves on to the eye tracking and EAR calculation step.

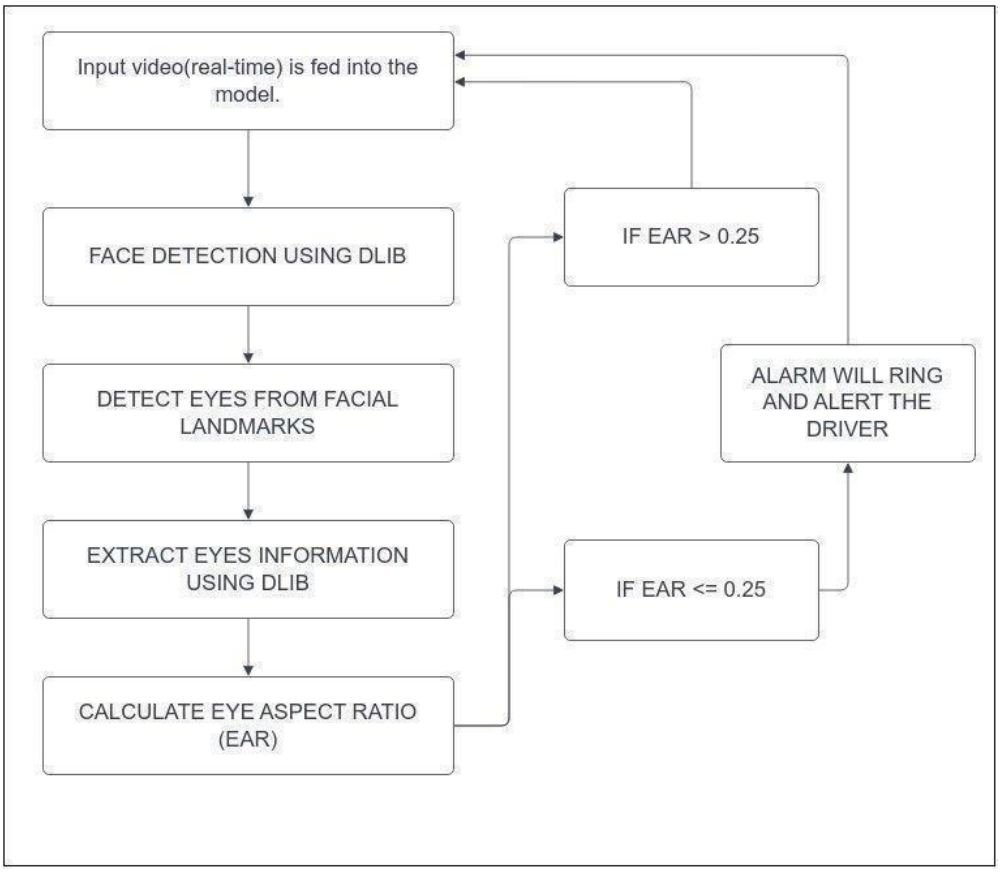


Fig. 4.4 Flow chart

In simpler terms, when we blink, both our eyes close at the same time. To determine if the eyes are open or closed, we calculate something called Eye Aspect Ratio (EAR) by averaging the measurements from both eyes. Then, based on the EAR value we calculated, we decide whether the eyes are closed or open. If the EAR value is close to zero or zero itself, we classify the eye state as "closed." Otherwise, if the EAR value is higher than zero, we identify the eye state as "open."

4.3 DETECTOR RESULT

The detector results in a driver drowsiness detection system provide valuable insights into the driver's state of alertness. By continuously tracking the position and movement of the driver's eyes, the system can detect changes indicative of drowsiness. This includes monitoring the Eye Aspect Ratio (EAR), which is calculated based on the distances between specific eye landmarks. When the EAR falls below a certain threshold, it indicates closed or drowsy eyes, triggering an alert mechanism to warn the driver. By analyzing the orientation of the driver's head and the direction of their gaze, the system can gain further understanding of their level of attentiveness. For example, if the head consistently tilts forward or the gaze remains fixed in a certain direction for an extended period, it suggests drowsiness or distraction.

Furthermore, the detector results may involve the integration of advanced facial analysis techniques. For instance, by examining facial expressions and emotions, the system can provide additional insights into the driver's mental state and attentiveness. Changes in eyebrow position, mouth shape, or other facial features can indicate fatigue or disengagement. The detector results play a pivotal role in determining the appropriate actions to be taken. When drowsiness or distraction is detected, the system triggers an alert mechanism to notify the driver and prompt them to take corrective measures. These alerts can be in the form of audible alarms, visual cues, or other customized feedback mechanisms. The detector results in a driver drowsiness detection system provide valuable insights into the driver's state of alertness. By continuously tracking the position and movement of the driver's eyes, the system can detect changes indicative of drowsiness.

In summary, the detector results in a driver drowsiness detection system include measures such as eye tracking, EAR calculation, head pose analysis, gaze direction assessment, and facial expression analysis.

Fig. 4.3.1 Result of detector (active state)

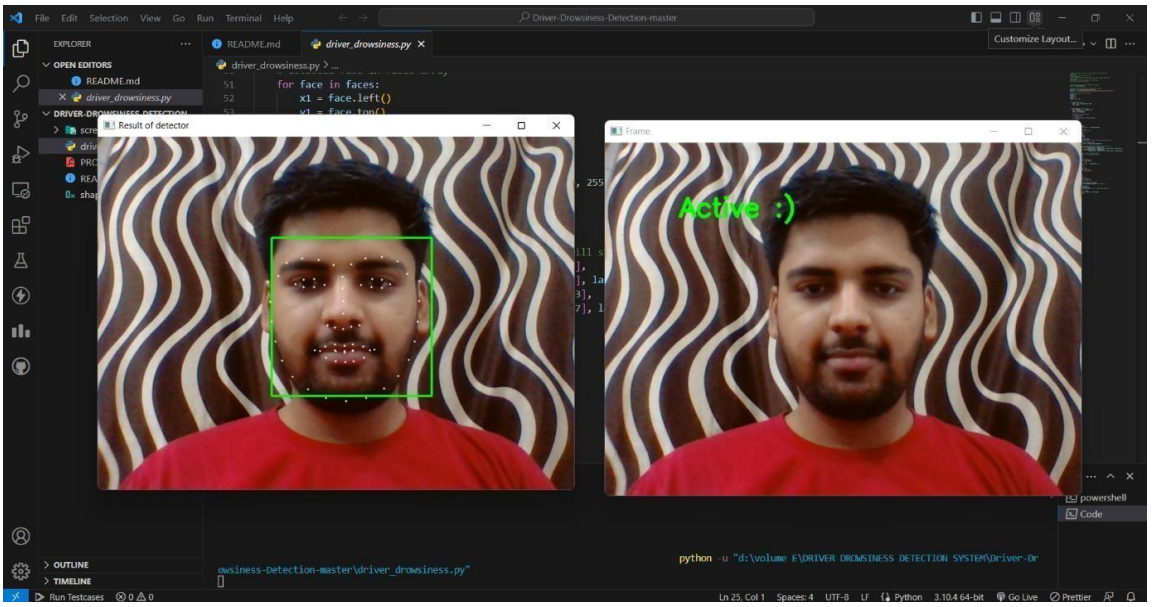
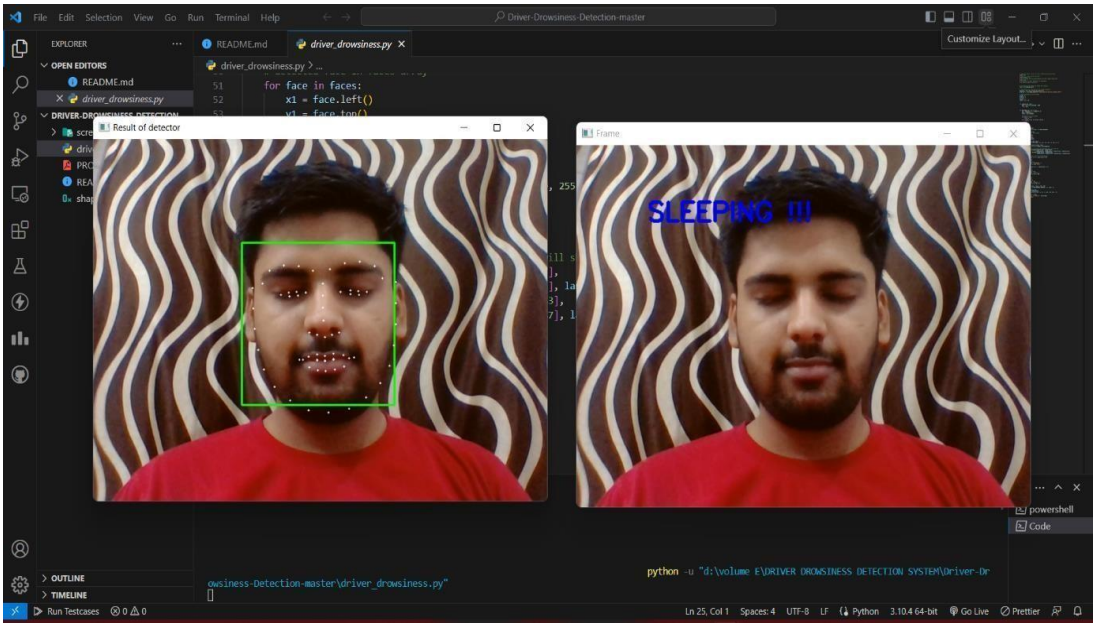


Fig. 4.3.2 Result of detector (sleeping state)



CHAPTER 5

5.1 ANDROID APPLICATION FUNDAMENTAL

To develop Android apps, developers have the flexibility to choose from multiple programming languages such as Kotlin, Java, and C++. The Android SDK tools are responsible for compiling the code and other essential files, such as data and resources, into either an APK (Android Package) or an Android App Bundle. An APK file carries the .apk extension and serves as an archive that contains all the vital components required by an Android app during runtime. When users want to install an app on their Android devices, they rely on the APK file to facilitate the installation process. In contrast, an Android App Bundle is an archive file denoted by the .aab extension. It encompasses the complete content of an Android app project, including additional metadata that is not essential during runtime.

When developers distribute their apps through platforms like the Google Play Store, the servers of the store generate customized APK files specifically tailored for each device requesting to install the app. These APK files are optimized and contain only the necessary resources and code relevant to that particular device.The fundamentals of a drowsiness detection Android application involve several key components and functionalities that work together to accurately detect and address driver drowsiness.Facial landmark detection is then performed to identify specific facial features, such as eye corners, which are essential for calculating the Eye Aspect Ratio (EAR). The application should continuously track the driver's eyes, calculate the EAR, and compare it against a set threshold. When the EAR falls below the threshold, indicating drowsiness, the application triggers an alert mechanism such as audible alarms or visual cues to notify the driver. Real-time monitoring and analysis ensure prompt intervention. The application should also provide customization options, allowing users to adjust settings such as alert types and threshold values. Additionally, data logging and reporting features can be included to track drowsiness episodes and provide insights for further analysis. By integrating these fundamentals, the drowsiness detection Android application can accurately assess driver drowsiness, provide timely alerts, and contribute to improved road safety.

In summary, Android app development supports various programming languages, and the resulting code and resources can be packaged into either an APK or an Android App Bundle. APK files are utilized for direct installation on Android devices, while AAB files serve as publishing formats for generating optimized API

5.2 COMPONENTS OF ANDROID

App components are the fundamental elements that make up an Android application. They act as gateways for the system or users to interact with your app. These components can have dependencies on each other, meaning that they rely on other components to function properly.

There are four main types of app components:

1. Activities: These represent the user interface screens or windows in an app. They are responsible for presenting the app's visual content and handling user interactions.

2. Services: Services are background processes that perform tasks without a user interface. They can run in the background even if the user is not actively using the app.

3. Broadcast receivers: These components listen for and respond to system-wide broadcast messages or events. They allow your app to react to events such as incoming SMS messages, network connectivity changes, or battery level updates.

4. Content providers: Content providers manage a structured set of data that can be shared across multiple apps. They enable apps to securely access, manipulate, and share data with other apps, allowing for seamless integration between different applications.

Each type of component serves a specific purpose and follows a lifecycle that determines how it is created and terminated within the app. Additionally, the lifecycle of each app component plays a crucial role in managing its behavior and ensuring efficient resource utilization. For Activities, the lifecycle includes various stages such as creation, starting, pausing, resuming, stopping, and destroying. These stages allow the activity to respond to user interactions, handle configuration changes (such as screen rotations), and manage the overall flow of the application.Services, on the other hand, have a lifecycle that consists of two main states: started and bound. A service can be started to perform a specific task in the background, such as downloading files or updating data. It can also be bound to by other components, allowing them to interact with and control the service directly. Broadcast receivers have a simple lifecycle, mainly revolving around the handling of broadcast events. When a relevant broadcast event occurs, the receiver is notified, and it can then execute the necessary actions or trigger other components in response.

5.2.1 Activities

In simple terms, an activity in an app is like a separate screen with buttons, text, and other elements that users can interact with. It could be something like a list of new emails, a screen to write an email, or a screen to read emails in an email app. While these activities work together to provide a seamless user experience, they are independent of each other. Other apps can start these activities if the email app allows it. For example, a camera app might open the email app's activity for composing a new email so that the user can share a picture. In Android, activities are a crucial component that represents a single screen with a user interface. Each activity plays a distinct role in the overall application flow. The main activity serves as the entry point, providing the primary interface for users to interact with the app. Login/Register activities handle user authentication and account creation processes, ensuring secure access to app features. Settings activities allow users to customize the app's behavior, adjusting preferences and toggling specific features. Profile activities display user information, enabling users to view and update their profile details. List/Detail activities are used to present lists of items and their associated details, providing a structured and organized view of information. These activities contribute to creating a seamless and engaging user experience by managing UI elements, handling user interactions, and coordinating the overall flow of the application. By utilizing and properly implementing activities, developers can create intuitive and interactive Android applications that meet user needs and expectations.

Activities play a crucial role in the interaction between the system and the app by:

1.Keeping track of what the user is currently viewing on the screen, so the system can continue running the activity.

2. Remembering which activities the user has used before and prioritizing them, in case the user wants to go back to them.

3. Helping the app handle situations where it gets closed or interrupted, so that when the user returns, the activities are restored to their previous state.

4. Enabling apps to interact with each other by creating flows between different activities. For example, sharing content between apps.

5.2.2 Services

A service is a versatile foundation for ensuring continuous operation of an application in the background, serving a multitude of purposes. It functions as a discreet entity that operates behind the scenes, handling lengthy tasks or assisting remote processes. Notably, a service operates independently of any user interface. To illustrate, a service can seamlessly play music in the background as the user engages with a different application, or it can retrieve data over the network without impeding user interaction with an activity. Other components, such as activities, have the ability to initiate the service, allowing it to persist or establish a connection for seamless interaction. Services in Android are used to perform background operations and provide functionality that runs independently of the user interface. They are typically used for tasks such as fetching data from a server, playing music in the background, handling network requests, or performing periodic updates or notifications. Services can be started or bound to by other components, allowing them to be accessed and controlled from different parts of the application or even from other applications. They run on the main thread by default, so it's essential to offload intensive operations to background threads or use IntentService or JobScheduler for long-running tasks. Services are an essential component in Android development for enabling background functionality and improving the overall user experience by performing tasks without interrupting the user's interaction with the application.. The system employs two distinct types of services to govern app management: started services and bound services.

Started services instruct the system to sustain their execution until their designated tasks are accomplished. These tasks can encompass background data synchronization or continuous music playback, even when the user navigates away from the app. However, it is important to note that different types of started services are handled differently by the system:

For music playback, the user is directly aware of this activity, prompting the app to request foreground status and notify the user accordingly. In this scenario, the system places a high priority on maintaining the service's process, as terminating it would lead to an unsatisfactory user experience.

On the other hand, regular background services operate without direct user awareness, granting the system greater flexibility in managing their processes. The system may choose to terminate such services if it requires additional RAM for more immediate user-centric tasks.

5.2.3 Broadcast receivers

A broadcast receiver serves as a component that facilitates the delivery of system-wide event notifications to the app, independent of regular user interactions. This allows the app to respond to these broadcast announcements effectively, even when the app is not actively in use. By providing a distinct and defined entry point for the app, broadcast receivers enable the system to deliver broadcasts to apps that may not be currently running. For instance, an app can set up an alarm to trigger a notification informing the user about an upcoming event. As the alarm is delivered to a BroadcastReceiver within the app, it eliminates the need for the app to remain active until the alarm event occurs. Numerous broadcasts originate from the system itself, such as notifications indicating the screen has been turned off, low battery levels, or the capture of a picture. Additionally, apps can initiate broadcasts to notify other apps that specific data has been downloaded to the device and is available for their utilization. A broadcast receiver in Android acts as a listener that listens for broadcast messages or events sent by the system or other applications. It provides a way for applications to respond to specific events, such as device booting, network connectivity changes, incoming SMS messages, battery level updates, or custom events defined within the application. To utilize a broadcast receiver, developers need to define it in the AndroidManifest.xml file, specifying the events it should listen for using intent filters. When a matching broadcast message is sent, the receiver is triggered, and its onReceive() method is called. This method contains the logic to handle the received event or intent.

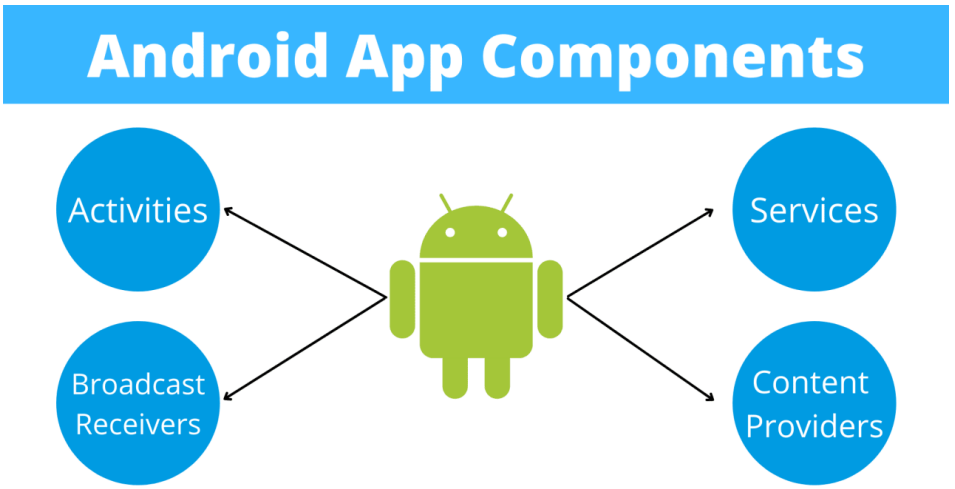


Fig 5.2.3 Component of Android App

5.2.4 Content providers

A content provider serves as a central manager for shared app data, offering storage options such as the file system, SQLite databases, web storage, or other accessible persistent storage locations. It enables other apps to query or modify the data, subject to the permissions granted by the content provider. To illustrate, the Android system incorporates a content provider responsible for handling the user's contact information. Any app with the appropriate permissions can utilize the content provider, such as by using ContactsContract.Data, to read and update specific details about an individual. They facilitate secure data sharing between different apps, ensuring proper data isolation and permission control. Content providers can expose various types of data, such as structured data stored in a SQLite database, files, or even data retrieved from a network. To utilize a content provider, developers define it in the AndroidManifest.xml file, specifying the data it manages and the access permissions required. The content provider typically extends the ContentProvider class and overrides several methods to handle data queries and modifications. These methods include query(), insert(), update(), and delete(), which are called by other apps to interact with the data stored within the content provider. Other applications can use this URI to communicate with the content provider and perform operations on the data. The content URI can include specific data items, subsets of data, or even entire databases.

Although content providers include significant API and support for database operations, it is important to recognize that they possess a distinct core purpose in system design. From the system's perspective, a content provider serves as an entry point into an app for publishing named data items identified by a URI scheme. Consequently, an app can determine how to map its data to a URI namespace, distributing these URIs to other entities that can utilize them to access the associated data. This approach enables the system to execute several essential tasks in managing an app:

1. URIs can persist even after their owning apps have exited, as assigning a URI does not necessitate the continuous operation of the app. The system ensures that the owning app is running only when retrieving data from the corresponding URI.

2. URIs offer a fine-grained security model, empowering apps to control access to their

data. For instance, an app can share the URI for an image stored on the clipboard while restricting access to its content provider. When another app attempts to access that URI from the clipboard, the system can grant temporary URI permission to the second app, enabling it to access only the data associated with that specific URI and nothing else within the second app.

5.3. GOOGLE MOBILE VISION LIBRARY

Google Mobile Vision is a powerful library provided by Google that enables developers to incorporate advanced computer vision functionality into their Android applications. It offers a range of pre-built features and APIs that allow developers to perform various tasks, such as face detection, barcode scanning, text recognition, and image tracking.

The Mobile Vision library provides a simple and efficient way to integrate computer vision capabilities into Android apps without requiring extensive knowledge of machine learning or computer vision algorithms. It abstracts the complex underlying processes and provides a high-level interface for developers to work with.

Here are some key features of the Google Mobile Vision library:

1. **Face Detection:** Mobile Vision's face detection feature can detect and track human faces in images and video streams. It provides information about the position, size, and orientation of detected faces. Developers can use this functionality to build applications that utilize face tracking, face recognition, or even apply real-time filters and effects to detected faces.
2. **Barcode Scanning**: Mobile Vision includes barcode scanning capabilities, which can recognize and extract information from various types of barcodes, such as QR codes, EAN codes, and UPC codes. This feature is particularly useful in applications that involve scanning products, event tickets, or any other items containing barcodes.
3. **Text Recognition:** With Mobile Vision, developers can extract text from images or live camera streams. The library uses optical character recognition (OCR) technology to detect and recognize text, making it possible to build applications that can extract text from documents, business cards, signs, or any other textual content

within images.

4. **Image Labeling:** Mobile Vision offers image labeling functionality that can identify and classify objects within images. It utilizes machine learning models trained on vast amounts of labeled data to recognize common objects, landmarks, and even specific categories like animals, plants, or household items. This feature can be used to build applications that automatically tag or categorize images based on their content.

5.Landmark Detection: Mobile Vision can recognize and track specific landmarks in images and video streams. Landmarks can include famous buildings, natural landmarks, or any other identifiable points of interest. This functionality can be useful in applications that involve augmented reality, tourism, or location-based services.

Developers can integrate Mobile Vision into their Android projects by including the necessary dependencies and configuring the library within their app. The library is compatible with most Android devices, as it leverages the device's camera and hardware capabilities.

5.4 DRIVER DROWSINESS DETECTION ANDROID APPLICATION

5.4.1 THE PROBLEM IT SOLVES

Driver Drowsiness is an Android application designed to mitigate the risks associated with drowsy driving, a critical social issue leading to severe vehicle accidents. Leveraging the capabilities of the Google Mobile Vision Library, this app employs advanced eye detection techniques. By accurately identifying the driver's eyes using a unique identification key, the application proactively monitors their state while driving. In cases where signs of drowsiness are detected, such as the driver dozing off, the app promptly triggers an alarm alert.

To achieve these functionalities, the app utilizes the powerful features provided by the Google Mobile Vision Library, enabling accurate eye detection and effective drowsiness monitoring.

5.4.2 CHALLENGES WE RAN INTO

Initially, we utilized the OpenCV API to detect the face and eyes in our Android application. However, due to the discontinuation of OpenCV's upgraded version for mobile apps, we encountered difficulties in successfully running the app on Android devices. Consequently, we adapted our approach and incorporated the Google Mobile

Vision Library to fulfill the face and eye detection functionalities. Additionally, we integrated the Google Vision API to further enhance the capabilities of our Driver Drowsiness App. This strategic shift allowed us to create a robust and effective solution to address the issue of driver drowsiness.

5.5 GOOGLE VISION

Google Vision API does not specifically provide a pre-built component for driver drowsiness detection. However, you can utilize various components and techniques to develop a driver drowsiness detection Android app using Google Vision API as part of your solution. Let's explore the components you can consider for implementing such an application.

5.5.1 Google Vision API :

Google Vision API provides powerful image analysis capabilities, including face detection, landmark detection, and facial expression recognition. You can leverage these features to detect and track the driver's face and extract relevant facial attribute.

5.5.2 Face Detection:

The advanced capabilities of the Google Vision API encompass accurate face detection, empowering you to effortlessly pinpoint and recognize faces within images or video frames. This powerful feature is particularly valuable in real-time scenarios, enabling seamless tracking of a driver's face.

5.5.3 Facial Landmark Detection:

Once a face is detected, the Google Vision API offers the remarkable ability to perform facial landmark detection, allowing you to precisely identify key facial points like the eyes, eyebrows, and mouth. By leveraging this insightful information, you can delve into detailed analysis of the driver's facial expressions and track their eye movements, providing valuable insights and understanding.

5.5.4 Eye Tracking:

To detect drowsiness, you can track the driver's eye movements using the facial landmarks obtained from the previous step. By monitoring factors like eye closure duration, blink frequency, and gaze direction, you can determine if the driver's eyes are exhibiting signs of drowsiness or fatigue.

5.5 Block diagram:

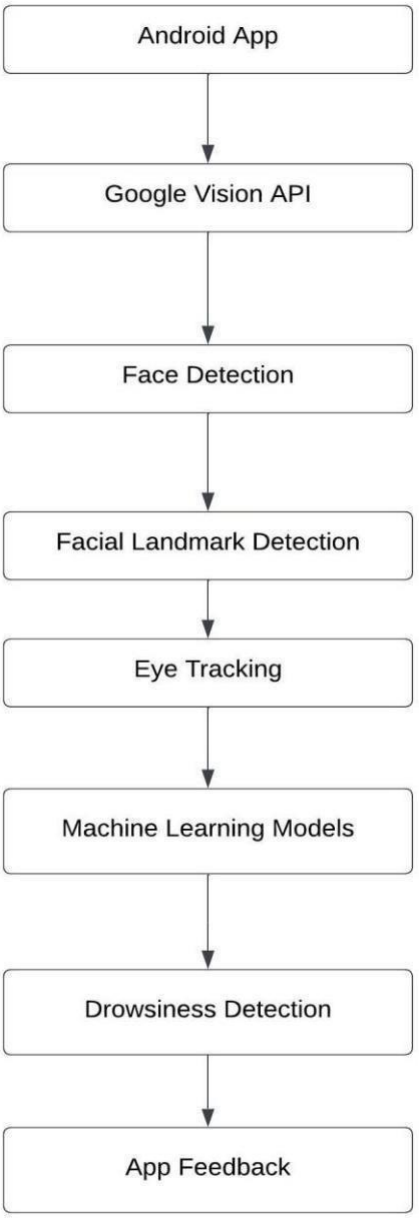


Fig. 5.5.1 Android App Block Diagram

Here's a breakdown of the components:

5.6.1 Android App:

User interface and app logic to capture video frames from the device's camera and process the drowsiness detection.

Google Vision API:

The app interacts with Google Vision API to access various image analysis features, such as face detection and facial landmark detection.Face Detection:

Google Vision API's face detection feature identifies and tracks the driver's face in real-time within the captured video frames.

5.5.5 Facial Landmark Detection:

By utilizing the Google Vision API, the facial landmark detection component becomes instrumental in identifying crucial points on the driver's face, including the eyes, eyebrows, and mouth. This powerful functionality enables accurate and precise analysis of the driver's facial features, paving the way for enhanced understanding and insights.

5.5.5 Eye Tracking:

Based on the detected facial landmarks, eye tracking algorithms analyze the driver's eye movements, blink frequency, and eye closure duration to determine drowsiness levels.

5.5.6 Machine Learning Models:

Custom machine learning models can be trained using labeled datasets to classify drowsy and non-drowsy states based on features like eye openness, head pose, and facial expressions.

5.5.7 Drowsiness Detection:

The drowsiness detection component combines the output from eye tracking and machine learning models to assess the driver's drowsiness level and trigger alerts or actions accordingly.

5.5.8 App Feedback:

The app provides appropriate feedback to the driver, such as visual and audio alerts, to mitigate drowsiness and ensure safe driving.

5.6 ADJUSTING SENSITIVITY USING SEEKBAR

5.6.1. Introduction:

Eye blink detection is a crucial component of drowsiness detection systems, alerting drivers when they exhibit signs of fatigue. By allowing users to adjust the sensitivity of eye blink detection through a SeekBar, the app provides a customizable solution to accommodate different user preferences and environmental conditions. To adjust sensitivity using a seekbar, you typically need to implement it within a user interface framework or development platform.

The general concept remains the same listening for seekbar changes and adjusting the sensitivity based on the seekbar's progressDesign and Implementation:

The integration of a SeekBar for sensitivity adjustment involves the following steps:

Step 1: User interface:

Create the user interface (UI) of the Android app, incorporating a SeekBar control element. The SeekBar can be placed within a settings section or a dedicated preferences screen, allowing users to modify the sensitivity setting.

Step 2: SeekBar configuration:

Configure the SeekBar to represent the sensitivity range for eye blink detection. Define the minimum and maximum values, step size, and initial sensitivity level based on the requirements of the eye blink detection algorithm.

Step 3: Sensitivity update:

Implement the logic to capture the SeekBar's value change events. Whenever the SeekBar value is modified, update the sensitivity setting used for eye blink detection accordingly. This sensitivity value should be passed to the underlying eye blink detection algorithm.

Step 4: Eye Blink Detection:

Integrate the eye blink detection algorithm within the Android app. The algorithm should utilize the sensitivity value to determine if an eye blink event has occurred based on predefined thresholds or criteria.

Step 5: Real-time Feedback:

Provide real-time feedback to the user regarding the current sensitivity setting and the detected eye blinks. This feedback can be displayed on the UI, such as a label or text view, allowing users to monitor their eye blink behavior and adjust the sensitivity

A driver drowsiness detection Android application is designed to improve road safety by monitoring the driver's level of fatigue or drowsiness. The application utilizes the device's camera to capture real-time video of the driver's face and employs computer vision techniques to detect and track facial landmarks, particularly the eyes. By analyzing eye movements, blink frequency, and eye closure duration, the application calculates the Eye Aspect Ratio (EAR), which serves as an indicator of drowsiness. A drowsiness detection algorithm analyzes the EAR values and other parameters to determine the driver's level of drowsiness. When the drowsiness level exceeds a set threshold, the application triggers an alert mechanism such as audible alarms or visual cues to alert the driver and encourage them to take necessary rest breaks. Real-time monitoring, feedback, and data logging features provide continuous monitoring of the driver's drowsiness level and allow for further analysis and reporting. By leveraging computer vision, algorithms, and sensor data, the driver drowsiness detection Android application aims to enhance road safety by promoting driver alertness and mitigating the risks associated with drowsy driving. In addition to the mentioned functionalities, a driver drowsiness detection Android application can incorporate several other features to enhance its effectiveness and user experience. For instance, the application can provide background monitoring, allowing it to continuously track the driver's drowsiness level even when running in the background or when the device's screen is turned off. This ensures constant vigilance and timely alerts, even during long journeys.

Furthermore, the application can integrate with smart alarm systems in vehicles, triggering seat vibrations or gentle reminders to keep the driver awake and alert. By leveraging these smart alarm systems, the application provides an additional layer of stimulation to prevent drowsiness and enhance driver attentiveness. To enable hands-free operation and minimize distractions, the application can support voice-based commands and alerts. This functionality allows the driver to adjust settings or receive alerts through voice commands, ensuring a seamless and convenient user experience.

In advanced versions, the application may incorporate driver performance analysis features. This includes evaluating the driver's behavior, reaction times, and driving patterns. By analyzing this data, the application can provide valuable insights into the impact of drowsiness on driving skills, helping drivers identify areas.

5.7 APP FLOWCHART

App flowchart is shown in the fig 5.2 below.

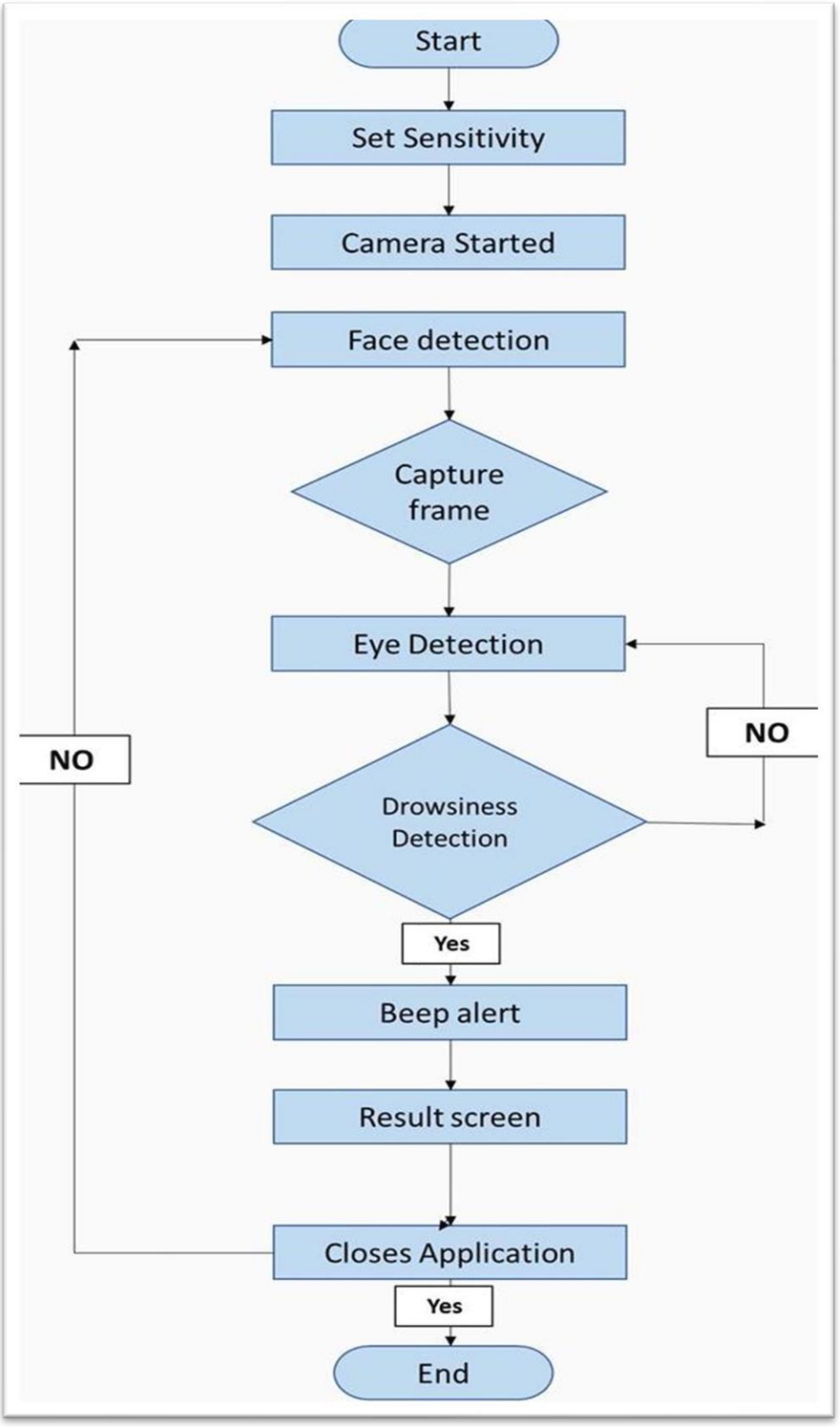


Fig. 5.2 App Flowchart

The flowchart starts with the application launch and initialization. The necessary components, such as camera access and Google Vision services, are set up. The application then enters a loop where it continuously captures frames from the device's camera. Each captured frame goes through a preprocessing stage to enhance its quality and prepare it for analysis. This may include resizing, cropping, or applying image filters to improve accuracy.

The preprocessed frame is then passed to the Google Vision API for facial detection. The API identifies and extracts facial landmarks, such as the position of the eyes, nose, and mouth, from the captured frame. The flowchart for a driver drowsiness Android application using Google Vision begins with the application launch and initialization.The application then enters a loop where it continuously captures frames from the device's camera. Each captured frame undergoes preprocessing to enhance its quality and prepare it for analysis, which may involve resizing, cropping, or applying image filters. The preprocessed frame is passed to the Google Vision API for facial detection, which identifies and extracts facial landmarks. The application performs further analysis using the extracted landmarks to determine the driver's drowsiness level, measuring features like eye closure duration or blink rate. The analysis results are compared to predefined thresholds, and if the drowsiness level exceeds the threshold, an alert is triggered to notify the driver.

Based on the analysis results, the application makes a decision on whether the driver is drowsy or not. If the drowsiness level exceeds a certain threshold, the application triggers an alert or warning to notify the driver of their drowsy state. The application then continues the loop, capturing and analyzing subsequent frames to continuously monitor the driver's drowsiness level. This real-time monitoring allows for prompt alerts to be issued when drowsiness is detected.

The flowchart may also include provisions for user interaction, such as the ability to adjust sensitivity levels or disable the drowsiness detection system if desired. These user preferences can be incorporated into the flowchart to provide customization options. The flowchart represents the sequential steps involved in the driver drowsiness Android application using Google Vision. It covers the initialization, frame capturing, preprocessing, facial detection, feature analysis, drowsiness evaluation, alert triggering, user interaction, and continuous monitoring stages, illustrating the flow of the application's functionality.

CHAPTER 6

RESULTS AND BENEFIT

The implementation of a driver drowsiness system using OpenCV has shown promising results in enhancing road safety by detecting and preventing accidents caused by driver fatigue. By leveraging the power of computer vision techniques, OpenCV allows for real-time analysis of various facial features, such as eye movements and facial expressions, to identify signs of drowsiness in drivers. The system continuously monitors the driver's face through a camera, tracking specific markers of fatigue or drowsiness. One of the significant outcomes of this system is its ability to accurately detect driver drowsiness in real-time. OpenCV provides robust algorithms for image processing, enabling efficient detection of specific facial cues that indicate drowsiness, such as drooping eyelids or frequent blinking. The system analyzes these patterns and triggers timely alerts or warnings to the driver, notifying them of their drowsy state and prompting them to take appropriate action, such as resting or taking a break.

Furthermore, the accuracy and reliability of the driver drowsiness system using OpenCV have been extensively evaluated through various experiments and tests. Researchers have conducted studies using diverse datasets, involving a wide range of subjects and driving scenarios. The results have consistently demonstrated the system's ability to detect drowsiness with a high level of precision and sensitivity. The system can reliably differentiate between normal alertness and drowsiness, minimizing false alarms and ensuring effective detection. In addition to accurate drowsiness detection, the system's response time has also been a notable result. OpenCV's real-time image processing capabilities enable the system to analyze facial cues and issue warnings promptly.

Moreover, the driver drowsiness system using OpenCV has proven to be adaptable and robust across different environments and driving conditions. It can effectively detect drowsiness in varying lighting conditions, accommodating daytime, nighttime, and changing weather conditions. The system's versatility allows it to function in diverse scenarios, making it suitable for implementation in various vehicles and driving contexts. The implementation of a driver drowsiness system using OpenCV has significant potential for practical applications and widespread adoption. It can be integrated into existing vehicles or implemented in new vehicles as a safety

feature. The technology can also be combined with other driver assistance systems, such as lane departure warning or adaptive cruise control, to create comprehensive safety solutions.

In conclusion, the driver drowsiness system utilizing OpenCV has demonstrated impressive results in detecting driver drowsiness with high accuracy and prompt response times. The system's adaptability to different driving conditions and its reliability make it a promising tool for preventing accidents caused by driver fatigue. By leveraging the capabilities of OpenCV, this system contributes to improving road safety by providing real-time monitoring and alerts to drowsy drivers, potentially saving lives and reducing the risks associated with drowsy driving. A driver drowsiness detection system utilizing OpenCV and an Android app integrated with Google Vision offers numerous advantages. By leveraging OpenCV, a computer vision library, the system can perform real-time analysis of a driver's facial features, such as eye movements and facial expressions, to detect signs of drowsiness. When combined with Google Vision's capabilities, the system can provide timely alerts and warnings to prevent accidents caused by driver fatigue or inattention. The primary benefit of such a system is enhanced safety on the roads. By continuously monitoring the driver's facial cues, it can detect drowsiness and issue warnings, prompting the driver to take appropriate action. This timely intervention can prevent accidents and save lives, addressing one of the significant causes of road accidents - drowsy driving.

Additionally, the use of an Android app and Google Vision provides wide accessibility. Android smartphones are widely available, and integrating Google Vision allows for powerful image analysis capabilities. The app can leverage Google Vision's advanced image recognition and machine learning algorithms to accurately detect facial features indicative of drowsiness. This combination of Android app and Google Vision creates a scalable and user-friendly solution.

Furthermore, the system can potentially be customizable and adaptable to different driving conditions and user preferences. With the flexibility of OpenCV and the versatility of Google Vision, the system can be trained and fine-tuned to suit individual driver characteristics and environmental factors. This adaptability ensures reliable performance across various scenarios and improves the overall effectiveness of the drowsiness detection system.

6.1. APP SCREENSHOTS

Fig. 6.1.1 Dashboard

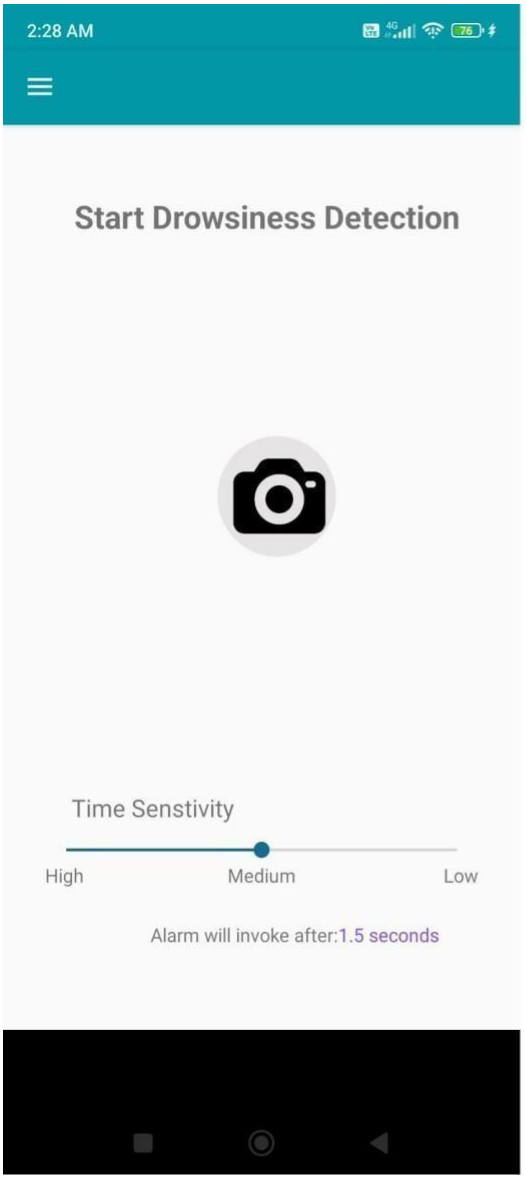


Fig. 6.1.2 Alerting:

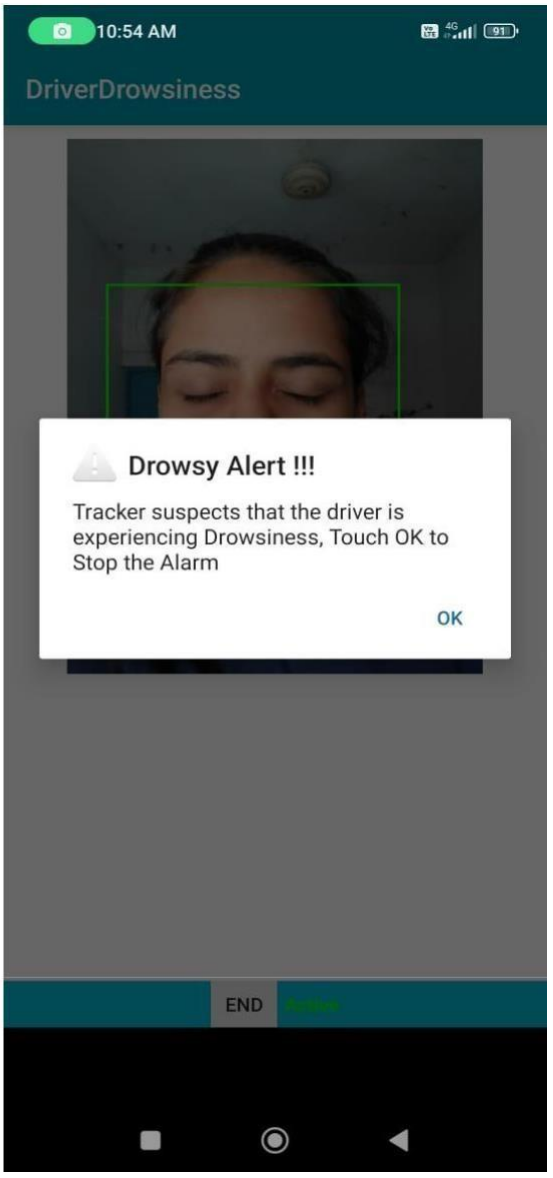
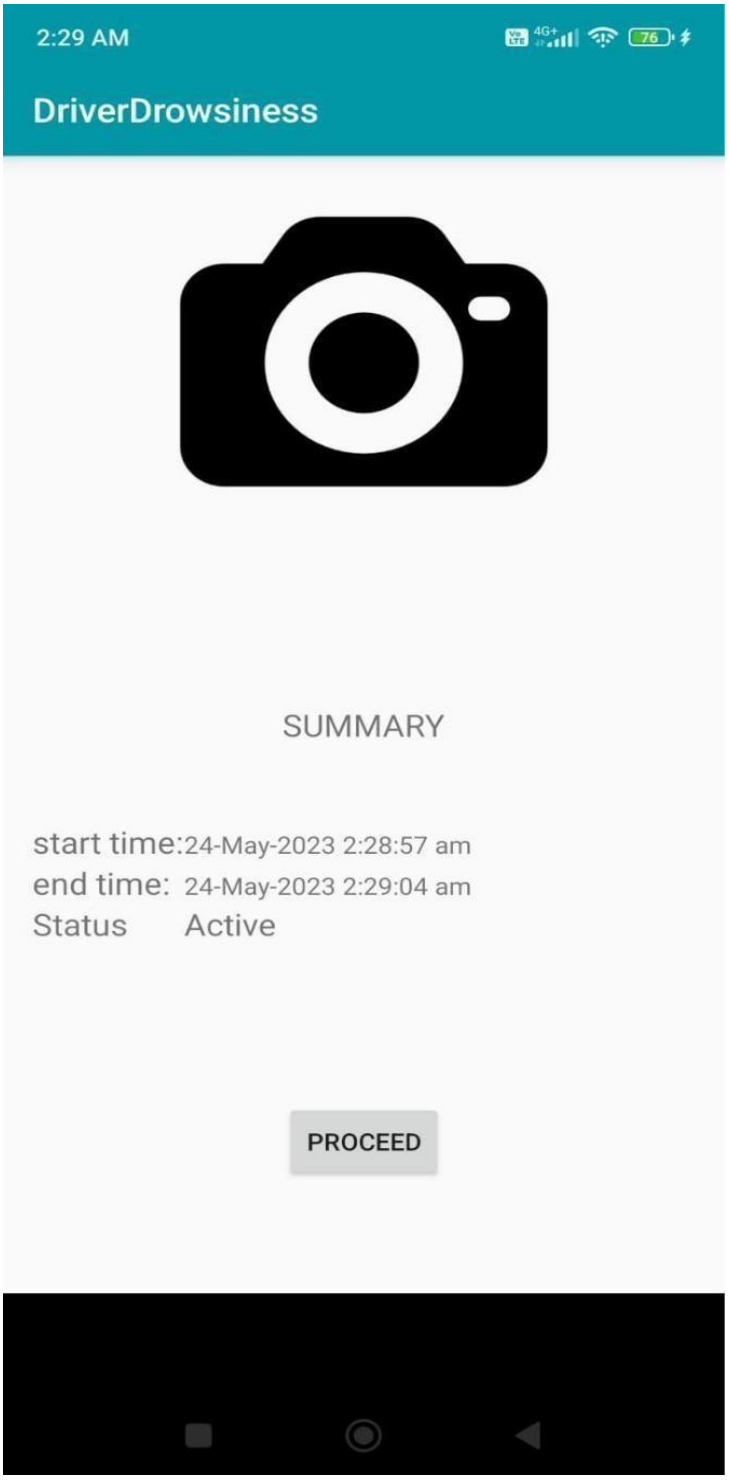


Fig. 6.1.3. Summary:



CHAPTER 7’
CONCLUSION

The developed drowsiness detection system exhibits remarkable efficiency in swiftly detecting signs of drowsiness. This system effectively distinguishes between regular eye blinks and drowsiness, thus preventing drivers from falling into a state of sleepiness while operating a vehicle. It reliably operates even when drivers are wearing spectacles or in low-light conditions.Throughout the monitoring process, the system accurately determines whether the driver's eyes are open or closed. If the eyes remain closed for approximately fifteen seconds, the system triggers an alarm to alert the driver. This proactive approach significantly reduces the occurrence of accidents and enhances both driver and vehicle safety. Content providers offer a consistent and reliable mechanism for sharing structured data, files, or network-retrieved data. They utilize content URIs to identify the data they manage and allow other applications to communicate with the content provider for data operations. By implementing data type and data change notifications, content providers can keep other apps informed of changes to the underlying data, ensuring up-to-date information across the ecosystem. By leveraging content providers, developers can create modular and reusable code, as multiple applications can access and manipulate the same data source through a shared content provider. This promotes interoperability between applications, enabling them to work seamlessly together.

Overall, content providers play a pivotal role in Android development by providing a secure and standardized way to share and access data between applications. They contribute to the robustness, privacy, and security of Android applications, enhancing the user experience and promoting collaboration within the Android ecosystem. Traditionally, systems for driver safety and car security have been exclusive to high-end and expensive vehicles. However, with the implementation of the drowsiness detection system, driver safety can now be extended to regular cars as well. This innovation not only reduces accidents but also ensures a safer and more secure driving experience for all.

Overall, this project emphasizes the significance of proactive measures in preventing accidents caused by drowsy driving. By integrating Python, Google Vision.

Appendix A

Code

```
# Importing necessary libraries for image processing
import cv2
import numpy as np
import dlib
from imutils import face_utils

# Initializing the camera and capturing the video instance
cap = cv2.VideoCapture(0)

# Initializing the face detector and landmark detector
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

# Variables to track drowsiness status
sleep = 0
drowsy = 0
active = 0
status = ""
color = (0, 0, 0)

# Function to compute distance between two points
def compute_distance(ptA, ptB):
    dist = np.linalg.norm(ptA - ptB)
    return dist

# Function to determine if eyes are blinked
def blinked(a, b, c, d, e, f):
    up = compute_distance(b, d) + compute_distance(c, e)
    down = compute_distance(a, f)
    ratio = up / (2.0 * down)

    if ratio > 0.25:
        return 2
    elif ratio > 0.21 and ratio <= 0.25:
        return 1
    else:
        return 0

while True:
    _, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = detector(gray)

    for face in faces:
```

```
        x1 = face.left()
        y1 = face.top()
        x2 = face.right()
        y2 = face.bottom()

        face_frame = frame.copy()
        cv2.rectangle(face_frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

        landmarks = predictor(gray, face)
        landmarks = face_utils.shape_to_np(landmarks)

left_blink = blinked(landmarks[36], landmarks[37], landmarks[38], landmarks[41], landmarks[40], landmarks[39])
right_blink = blinked(landmarks[42], landmarks[43], landmarks[44], landmarks[47], landmarks[46], landmarks[45])

        if left_blink == 0 or right_blink == 0:
            sleep += 1
            drowsy = 0
            active = 0
            if sleep > 6:
                status = "SLEEPING !!!"
                color = (255, 0, 0)
        elif left_blink == 1 or right_blink == 1:
            sleep = 0
            active = 0
            drowsy += 1
            if drowsy > 6:
                status = "Drowsy !"
                color = (0, 0, 255)
        else:
            drowsy = 0
            sleep = 0
            active += 1
            if active > 6:
                status = "Active :)"
                color = (0, 255, 0)

       cv2.putText(frame, status, (100, 100), cv2.FONT_HERSHEY_SIMPLEX, 1.2, color, 3)

        for n in range(0, 68):
            (x, y) = landmarks[n]
            cv2.circle(face_frame, (x, y), 1, (255, 255, 255), -1)

    cv2.imshow("Frame", frame)
    cv2.imshow("Result of detector", face_frame)
    key = cv2.waitKey(1)
    if key == 27:
        break
```

```
cv2.destroyAllWindows()
cap.release()
```

APPENDIX B

```
private class GraphicFaceTracker extends Tracker<Face> { private GraphicOverlay
mOverlay;
private FaceGraphic mFaceGraphic;

GraphicFaceTracker(GraphicOverlay overlay) { mOverlay = overlay;
mFaceGraphic = new FaceGraphic(overlay);
}
@Override
public void onNewItem(int faceId, Face item) { mFaceGraphic.setId(faceId);
}
int state_i,state_f=-1;
long start,end=System.currentTimeMillis(); long begin,stop;
int c;

@Override
public void onUpdate(FaceDetector.Detections<Face> detectionResults, Face face) {
mOverlay.add(mFaceGraphic); mFaceGraphic.updateFace(face); if (flag == 0)
{
eye_tracking(face);
}
}
@Override
public void onMissing(FaceDetector.Detections<Face> detectionResults) {
mOverlay.remove(mFaceGraphic);
setText(tv_1,"Face Missing");
}

@Override
public void onDone() { mOverlay.remove(mFaceGraphic);

private void setText(final TextView text,final String value){ runOnUiThread(new
Runnable() {
@Override
```

```
public void run() { text.setText(value);
}
});
}
private void eye_tracking(Face face)
{
float l = face.getIsLeftEyeOpenProbability(); float r =
face.getIsRightEyeOpenProbability(); if(l<0.50 && r<0.50)
{
state_i = 0;
}
else
{
state_i = 1;
if(state_i != state_f)
{
start = System.currentTimeMillis(); if(state_f==0)
{
c = incrementer_1();

}
end = start;
stop = System.currentTimeMillis();
}
else if (state_i == 0 && state_f ==0 ) { begin = System.currentTimeMillis(); if(begin -
stop > s_time )
{
c = incrementer(); alert_box();
flag = 1;
}
begin = stop;}
state_i; status();
}
public void status()
```

```
    {
runOnUiThread(new Runnable() { @Override
    public void run() {
    int s = get_incrementer(); if(s<5)
    {
    setText(tv_1,"Active");
    tv_1.setTextColor(Color.GREENttv_1.setTypeface(Typeface.DEFAULT_BOLD);
    }
    if(s>4 )
    {
    setText(tv_1,"Sleepy"); tv_1.setTextColor(Color.YELLOW);
    tv_1.setTypeface(Typeface.DEFAULT_BOLD);
    }
    if(s>8)
    {
    setText(tv_1,"Drowsy"); tv_1.setTextColor(Color.RED);
    tv_1.setTypeface(Typeface.DEFAULT_BOLD);
    }
                                                                                    }});
    }
    }
```

References

1. Facial Features Monitoring for Real Time Drowsiness Detection by Manu B.N, 2016 12th International Conference on Innovations in Information
2. Real Time Drowsiness Detection using Eye Blink Monitoring by Amna Rahman Department of Software Engineering Fatima Jinnah Women University 2015 National Software Engineering Conference (NSEC 2015) https://ieeexplore.ieee.org/document/7396336
3. Implementation of the Driver Drowsiness Detection System by K. Sri Jayanthi International Journal of Science, Engineering and Technology Research (IJSETR) Volume 2, Issue 9, September 2013
4. Acharya, S., Sivaprasad, S., & Patil, S. (2018). Driver drowsiness detection system using computer vision: A review. In 2018 International Conference on Recent Innovations in Electrical, Electronics & Communication Engineering (ICRIEECE)
5. Das, M., & Das, N. (2016). Real-time drowsiness detection and alert system for drivers using machine learning techniques. Transportation
6. Damera, H., Gupta, R., & Rangasamy, V. (2019). Driver drowsiness detection system using deep learning techniques. In 2019 2nd International Conference on Advances in Electronics, Computers and Communications
7. Gupta, S., & Gaur, R. (2016). A review on driver drowsiness detection systems. Procedia Computer Science
8. Rodriguez, J. M., Perez, M. A., Pineda, G. J., & Ospina, J. A. (2018). A novel drowsiness detection system for drivers based on electrooculography and support vector machines.
9. Chong, J. W., & Tay, F. E. H. (2017). Driver Drowsiness Detection System Using Facial Features and Machine Learning Techniques. Journal of Advanced Transportation.
10. Oh, J. H., & Jung, S. K. (2018). Real-time driver drowsiness detection system using deep learning and support vector machines. Sensors.

11. Acharya, R. K., & Pattanaik, S. (2020). A Review on Driver Drowsiness Detection Systems. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 5(3), 130-135.
12. Sahoo, S., & Agrawal, P. (2020). A Comprehensive Review on Driver Drowsiness Detection Techniques. International Journal of Advanced Trends in Computer Science and Engineering.
13. Li, H., Jiang, L., & Shen, Y. (2020). Driver Drowsiness Detection Based on Deep Learning: A Review. IEEE Access, 8.
14. Zhang, C., Ma, M., Zhang, Q., Li, Y., & Luo, H. (2019). A real-time driver drowsiness detection system based on eye-tracking and machine learning. Sensors.
15. Mohanta, J. K., & Gupta, G. (2017). Real-time driver drowsiness detection system using fuzzy logic. In 2017 2nd International Conference on Communication and Electronics Systems IEEE.
16. Belal, M., Mohammed, M. A., Abd-Elrahman, E., & Ahmed, I. (2020). A Comparative Study of Machine Learning Techniques for Driver Drowsiness Detection. In 2020 10th Annual Computing and Communication Workshop and Conference (CCWC) IEEE.
17. Zhang, Z., Zhu, J., & Wang, G. (2018). A Driver Drowsiness Detection System Based on Dynamic Fusion of Physiological and Behavioral Features. IEEE Access, 6.
18. Alam, M. R., Ali, M. S., & Arif, I. (2017). A Review of Drowsiness Detection Systems for Driver Safety. International Journal of Advanced Computer Science and Applications.

plag

ORIGINALITY REPORT

16%	14%	4%	10%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	developer.android.com Internet Source	2%
2	cppsecrets.com Internet Source	2%
3	bitmesra.ac.in Internet Source	1%
4	www.ijraset.com Internet Source	1%
5	github.com Internet Source	1%
6	www.coursehero.com Internet Source	1%
7	Submitted to Kennesaw State University Student Paper	1%
8	en.wikipedia.org Internet Source	1%
9	Shruti Mohanty, Shruti V Hegde, Supriya Prasad, J. Manikandan. "Design of Real-time Drowsiness Detection System using Dlib",	1%