

VEHIGUARD

-A Comprehensive Vehicular Safety System

A CAPSTONE PROJECT REPORT

*Submitted in partial fulfillment of the requirement
for the award of the
Degree of*

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING

by

**GONUGUNTLA ROHINI (20BCD7174)
PRATHAPANI SATWIKA (20BCD7160)
MEKALA POORNIMAI (20BCD7167)
GUTTI MANOJ SAI (20BCR7066)**

Under the Guidance of

DR. VISALAKSHI ANNEPU



**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
VIT-AP UNIVERSITY
AMARAVATI- 522237**

DECEMEBR 2023

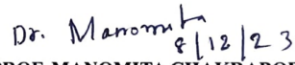
CERTIFICATE

This is to certify that the Capstone Project work titled “**VEHIGUARD**” that is being submitted by **Gonuguntla Rohini (20BCD7174), Prathapani Satwika (20BCD7160), Mekala Poornimai (20BCD7167), Gutti Manoj Sai (20BCR7066)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.



PROF. DR. VISALAKSHI ANNEPU
Guide

The thesis is satisfactory / unsatisfactory


PROF. DR. VIKASH KUMAR SINGH
Internal Examiner 1


PROF. MANOMITA CHAKRABORTY
Internal Examiner 2

Approved by


Dr. G Muneeswari
HoD, Data Science and Engineering
School of Computer Science and Engineering

ACKNOWLEDGEMENTS

We would like to extend our heartfelt gratitude and acknowledgment to Dr. Visalakshi Annepu from VIT AP for her invaluable guidance and support throughout our project journey. Her immense knowledge and expertise have considerably shaped our understanding and enabled us to navigate complex challenges.

Furthermore, we would also like to express our sincere appreciation to the esteemed panels who evaluated our project presentation. Their insightful comments and constructive feedback provided us with a fresh perspective and helped us to enhance the quality of our work. We are truly thankful for their unwavering commitment to fostering our academic growth.

In addition to Dr. Visalakshi Annepu and the evaluation panels, we would also like to convey our gratitude to the entire university. The institution's resources, facilities, and academic environment have played a critical role in facilitating our project's success. We are thankful for the continuous support and inspiration we have received from the university community.

Once again, we extend our sincere thanks to Dr. Visalakshi Annepu, the evaluation panels, and the university for their immense contribution to our project's development.

ABSTRACT

Transportation is playing a vital role in our daily life and its development has made many of our chores much easier. But in recent years, driver drowsiness, distractions, which leads to the road accidents. Slumbering, dozing, alcohol consumption cause intrusiveness which needs to alert the driver before a mishap happens. In the realm of advancing automotive technologies, ensuring the safety of drivers and passengers remains a critical concern. Vehiguard is a Raspberry Pi-based project designed to enhance vehicular safety through a multifaceted approach. The system integrates various sensors, machine learning models, and web communication to monitor and respond to different safety parameters in real-time. Several lives are saved by alerting the driver with help of a sound system that is deemed to prevent any distractions before happen. The project encompasses features such as accident detection through an accelerometer, alcohol presence sensing, seat belt status monitoring, and driver drowsiness detection using facial landmarks. Leveraging a GPS module, the system also provides continuous location tracking. Communication with ThingSpeak facilitates remote monitoring and data visualization. The code employs machine learning models, including K-Nearest Neighbors (KNN), Support Vector Classifier (SVC), and Logistic Regression (LR), for classifying and predicting various conditions. These models utilize a dataset loaded from a CSV file containing parameters relevant to vehicular safety. The project's novelty lies in its comprehensive approach, integrating both hardware and software components to create a robust vehicular safety system. Future enhancements may include refining drowsiness detection algorithms, employing more advanced machine learning techniques, and expanding the system's capabilities for additional safety features. Vehiguard stands as a testament to the potential of leveraging emerging technologies to address critical safety concerns in the automotive industry, paving the way for more secure and intelligent vehicles.

TABLE OF CONTENTS

S. No.	Chapter	Title	Page Number
1.		Acknowledgement	2
2.		Abstract	3
3.		List of Figures and Table	5
4.	1	Introduction	7
	1.1	Objectives	8
	1.2	Background and Literature Survey	8
	1.3	Organization of the Report	9
5.	2	Vehiguard -A Comprehensive Vehicular Safety System	10
	2.1	Proposed System	10
	2.2	Working Methodology	13
	2.3	Standards	14
	2.4	System Details	15
	2.4.1	Software	15
	2.4.2	Hardware	17
6.	3	Cost Analysis	24
	3.1	List of components and their cost	24
7	4	Results and Discussions	25
8	5	Conclusion and Future Work	31
9	6	Appendix	33
10	7	References	48

List of Tables

Table No.	Title	Page No.
1.	Cost Analysis	24

List of Figures

Figure No.	Title	Page No.
1.	General Architecture	10
2.	Behavioral measures	11
3.	Flowchart of Alcohol Detection	11
4.	Monitoring driver's face	12
5.	Detection of eye and mouth landmarks	12
6.	Ear	13
7.	ThingView app	17
8.	Location	17
9.	Raspberry pi	17
10.	Alcohol Sensor	18
11.	GPS Antenna	18
12.	GPS Module	19
13.	Motor Driver	19
14.	Accelerometer Sensor	20
15.	Switch	20
16.	Buzzer	20
17.	VNC Viewer	21
18.	VNC Viewer Address	22
19.	VNC Server	23
20.	VNC Properties Window	23
21.	Drowsiness alert 1	25
22.	Drowsiness alert 2	25

23.	Yawn alert 1	26
24.	Yawn alert 2	26
25.	Yawn alert 3	27
26.	GPS location1	28
27.	GPS location2	28
28.	KNN Confusion Matrix	29
29.	SVC Confusion Matrix	29
30.	Logistic Regression Confusion Matrix	30
31.	Hardware Setup	30

CHAPTER I

1. INTRODUCTION

In an era where road safety is of paramount concern, technological advancements have paved the way for innovative solutions aimed at accident prevention and driver safety. One such groundbreaking development is the integration of Raspberry Pi and Pi Camera technology into a comprehensive safety assistance device for vehicles. This device employs a sophisticated approach by continuously capturing and analyzing facial landmarks through the Pi Camera, utilizing Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) algorithms for drowsiness detection.

The system vigilantly monitors the driver's facial expressions, promptly identifying signs of drowsiness by detecting variations in the EAR and MAR ratios. Once a predetermined threshold is surpassed, signaling potential driver fatigue, the system issues an instant alert through a sound system, providing a crucial warning to the driver. In an extraordinary demonstration of safety commitment, the system takes proactive measures by automatically halting the vehicle when it determines that the driver is in a drowsy state, mitigating the risk of accidents caused by driver fatigue.

Recognizing the pervasive issue of driving under the influence, the proposed system integrates alcohol sensors as a robust preventive measure. By accurately measuring alcohol levels, the system ensures that the vehicle remains inoperable if the alcohol concentration exceeds a predefined limit, effectively curbing instances of drunk driving and enhancing overall road safety.

Furthermore, the device addresses the critical aspect of seat belt compliance, a key factor in minimizing injuries during accidents. In instances where the driver neglects to fasten their seat belt, the system activates an alarm, delivering a timely reminder to prioritize safety.

In the unfortunate event of an accident, the device goes beyond conventional safety features by sending immediate alerts to designated contacts, such as the victim's home. These alerts contain precise information, including the latitude and longitude of the accident site, enabling swift and accurate emergency responses and details on the impact severity.

In essence, this innovative accident prevention and safety assistance device stand as a testament to the fusion of cutting-edge technology and a proactive safety ethos. With its multifaceted approach encompassing drowsiness detection, alcohol sensing, seat belt reminders, and emergency alert systems, the device sets a new standard for vehicular safety, heralding a future where accidents are not just mitigated but prevented altogether.

1.1 Objectives

The following are the objectives of this project:

- Develop a robust accident detection algorithm that can quickly identify potential accidents based on vehicle dynamics, sudden changes in speed, and collision-like events.
- Implement a real-time notification system that sends alerts to mobile devices in the event of an accident, providing information about the severity of the impact.
- Integrate advanced sensors and machine learning techniques to detect whether the driver is wearing a seat belt, and issue warnings if the seat belt is not fastened.
- Employ a reliable alcohol detection mechanism that can identify if the driver is under the influence of alcohol, triggering appropriate warnings and notifications.

1.2 Background and Literature Survey

The motivation behind the development of VehiGuard stems from a poignant incident involving cricketer Rishabh Pant, highlighting the severe consequences of accidents caused by preventable factors like driver drowsiness. Such incidents underscore the critical need for advanced technological interventions in vehicular safety. Vehicular accidents, often attributed to driver fatigue and distractions, can lead to devastating outcomes. Recognizing the potential to address this issue, our project, VehiGuard, endeavors to create an innovative accident detection and prevention system. By leveraging cutting-edge technologies, we aim to mitigate the risks associated with driver fatigue and other factors, ultimately contributing to the reduction of accidents and the preservation of lives on the road.

The foundation of VehiGuard's development is grounded in a comprehensive exploration of existing literature on IoT-based accident detection systems. Alvi et al. (2020) conducted an extensive study on IoT-based accident detection systems for smart vehicles, providing valuable insights into the integration of Internet of Things (IoT) technologies to enhance vehicular safety.

Their work serves as a foundational reference for understanding the landscape of accident detection systems and informs our approach in creating an effective and intelligent system.

In the realm of sensor-based technologies, Putra et al. (2017) proposed an event-triggered machine learning approach for accelerometer-based fall detection. While their focus is on fall detection, the principles of event-triggered machine learning and accelerometer-based sensing align with the objectives of VehiGuard in detecting abrupt changes in vehicle dynamics. This work contributes to our understanding of sensor technologies and inspires the development of effective algorithms for accident detection.

Furthermore, Goyal et al. (2022) delve into the realm of real-time accident detection systems over IoT. Their work provides insights into the challenges and opportunities associated with implementing real-time accident detection mechanisms. By examining their findings, we gain a deeper understanding of the practical considerations and potential solutions that can inform the design and implementation of VehiGuard.

In amalgamating these foundational studies, VehiGuard draws from the collective knowledge to forge an innovative system that goes beyond traditional safety measures. The literature survey not only informs our technical approach but also highlights the importance of interdisciplinary collaboration, showcasing how advancements in IoT, machine learning, and sensor technologies can converge to create a transformative solution for vehicular safety.

1.3 Organization of The Report

The remaining chapters of the project report are described as follows:

- Chapter 2 contains the proposed system, methodology, hardware and software details.
- Chapter 3 gives the cost involved in the implementation of the project.
- Chapter 4 discusses the results obtained after the project was implemented.
- Chapter 5 concludes the report.
- Chapter 6 consists of codes.
- Chapter 7 gives references.

CHAPTER 2

2. Vehiguard -A Comprehensive Vehicular Safety System

This Chapter describes the proposed system, working methodology, software and hardware details.

2.1 Proposed System

General architecture

This system works by live monitoring of the driver. The general architecture is shown in Fig1. It comprises of the security features such as monitoring driver face, alcohol detection, seatbelt detection and detecting change in accelerometer values.

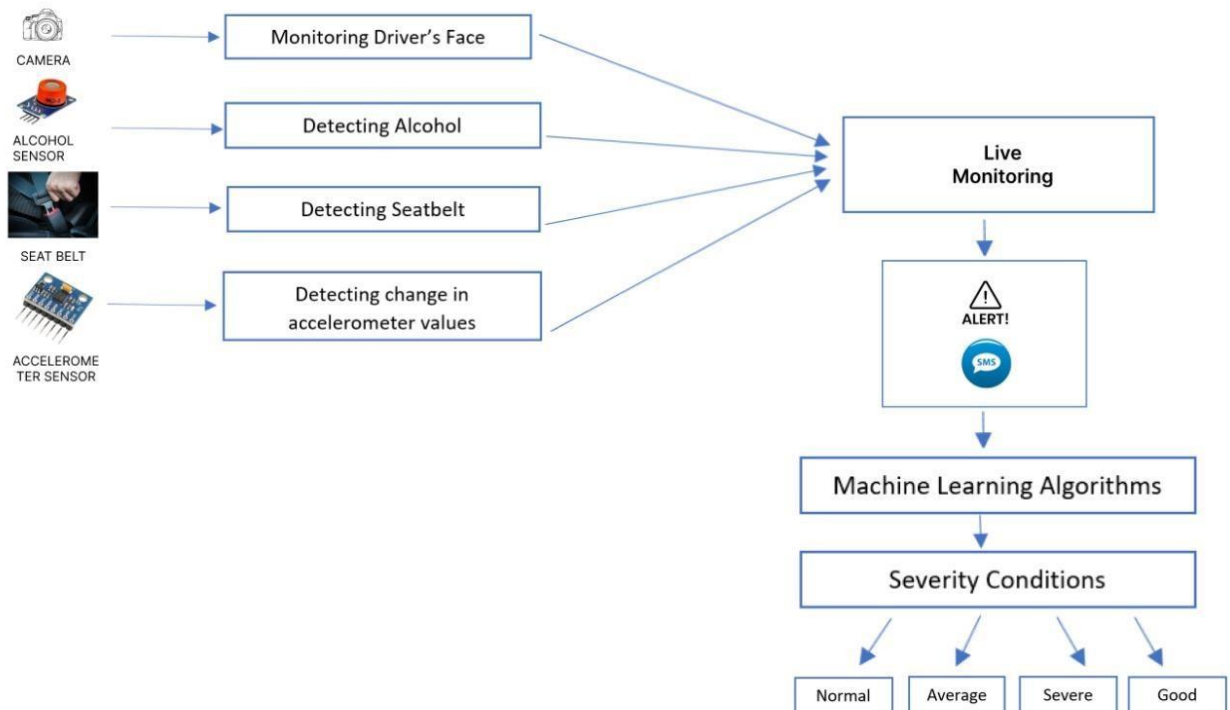


Fig1. General Architecture

DROWSINESS AND YAWN DETECTION

This study proposes accident prevention and safety aid. The current IoT-based technology requires drivers to wear Sensors such as smart glasses, eye detectors, pulse detectors, and heartbeat sensors can detect the driver's behavior with the help of IoT. Because these devices are external, it is not assured that the drivers will constantly wear the same; it also causes distractions when driving. The suggested system is built on IoT, however it does not require drivers to wear any electronic gadgets to detect their driving behaviors. This method also considers other potential causes of accidents, such as driving while intoxicated, as well as seatbelt safety measures.

This system is designed mainly to detect driver drowsiness, consumption of alcohol, seatbelt detection, and accident detection. As illustrated in Fig2, the proposed approach handles accidents induced by behavioral variation.

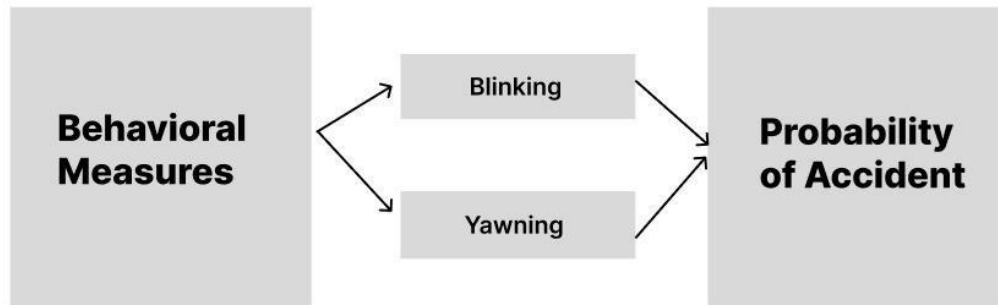


Fig2 Behavioral measures

Since various measures available in drowsiness detection, Behavioral measures depend on driver's behavior to detect whether the driver is fatigue or not. This method focuses on the driver's eye blinking ratio which is measured in terms of EAR called Eye Aspect ratio and mouth opening ratio which is termed as yawning and measured through mouth aspect ratio (MAR).

ALCOHOL DETECTION

Alcohol consumption can be detected in various ways. One way is to detect blood alcohol levels through a breath analyzer or through a sensor for sensing the alcohol intake. As we need a compatible testing mode for vehicles, MQ3 sensor is used in the implementation. It is compatible, easy to implement, and has a good sense of alcohol. This sensor starts working when it senses alcohol in the air. It alerts the driver if he is an alcoholic.

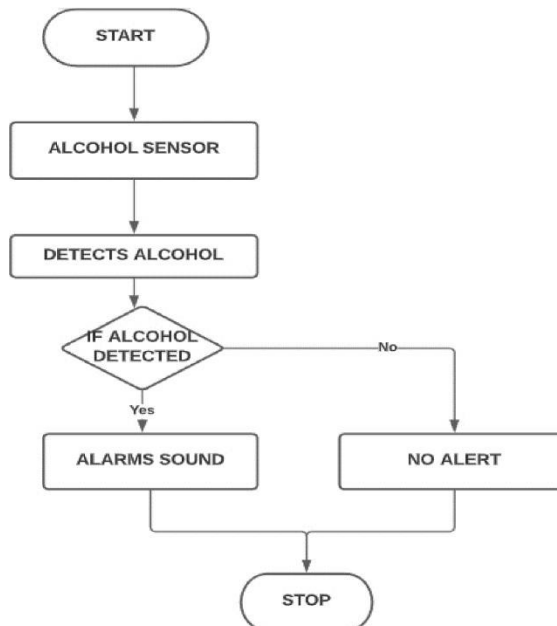


Fig3. Flowchart of Alcohol Detection

Monitoring drivers' face

The drivers' face is continuously monitored as given in Fig.4 The steps are explained as follows:

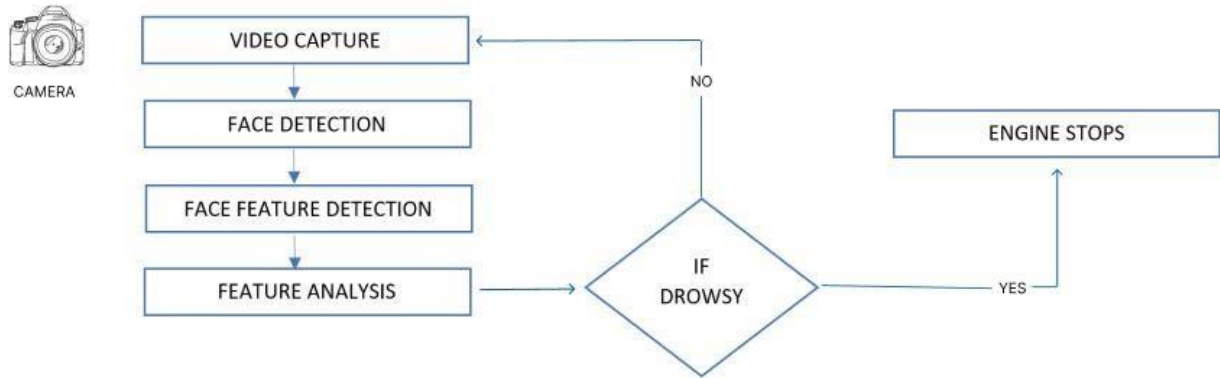


Fig4 Monitoring driver's face Video capture

The video is captured through the Raspberry Pi camera which is incorporated with the raspberry pi. The captured videos are segmented into continuous frames for face detection. Our proposed system fixes 15 continuous frames to detect the EAR and MAR to classify the drowsy driver.

Face feature extraction

Face Feature extraction is a prominent model next to face detection. It helps in detecting the useful landmarks of our face like eyes, nose, and mouth. In our work Eyes and Mouth identification are the next significant step in drowsiness detection.



Fig5 Detection of eye and mouth landmarks

Feature analysis

The structures of the selected face are then further processed into Percentage of eyelid closure (Percentage of eyelid closure) or Eye Aspect Ratio (EAR). The RGB form of images is converted into grayscale or gradient image for selecting the portions of eyes and mouth for detecting eye closing and mouth opening ratio. It also calculates the rate of speed of eye closure.

The EAR algorithm determines a calculation based on the ratio of the distances between various facial landmarks of the eyes as shown in Fig6.

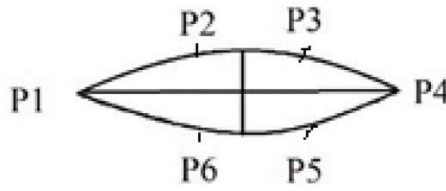


Fig6 EAR

2.2 Working Methodology

The working methodology of the proposed Vehiguard system involves a seamless integration of hardware components, software algorithms, and communication protocols to create a comprehensive vehicular safety solution. The system's core functionalities encompass accident detection, driver drowsiness monitoring, alcohol sensing, and seat belt compliance, all aimed at preventing potential road hazards.

Firstly, the hardware components, including a Raspberry Pi, Pi Camera, sensors for monitoring eye movements, and an accelerometer for accident detection, are strategically installed within the vehicle. The Pi Camera continuously captures facial landmarks, and sophisticated algorithms, such as the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR), are employed to analyze these landmarks for signs of driver drowsiness. The system vigilantly monitors variations in these ratios, promptly identifying potential fatigue.

Simultaneously, the accelerometer detects sudden changes in vehicle dynamics and collision-like events, triggering the accident detection algorithm. This multifaceted approach ensures a robust system capable of identifying diverse safety parameters in real-time. Additionally, alcohol sensors are integrated into the system to accurately measure alcohol levels, acting as a preventive measure against drunk driving.

The software component utilizes machine learning models, including K-Nearest Neighbors (KNN), Support Vector Classifier (SVC), and Logistic Regression (LR). These models are trained on a dataset containing parameters relevant to vehicular safety, allowing the system to classify and predict conditions such as drowsiness, alcohol influence, and seat belt status. The realtime data generated by the hardware components are processed by these models, enabling quick and accurate decision-making.

The system employs IoT technology to transmit the driver's behavior and driving patterns to the cloud in real-time. This enables remote monitoring and quick response under emergency situations. Communication with platforms like ThingSpeak facilitates data visualization, enhancing the ability to track and analyze driving patterns over time.

In the event of a detected risk, such as drowsiness or alcohol influence surpassing predefined thresholds, the system issues immediate alerts. For drowsiness, an alert is sent through a sound system, providing a crucial warning to the driver. In the case of alcohol influence, the system ensures the vehicle remains inoperable, preventing potential accidents. Seat belt noncompliance activates an alarm to remind the driver to prioritize safety.

Moreover, in the unfortunate event of an accident, the system goes beyond immediate alerts by sending detailed notifications to designated contacts, including precise location information. This ensures swift and accurate emergency responses, minimizing the impact severity. The proposed VehiGuard system, with its multifaceted approach and innovative use of technology, stands as a pioneering solution for enhancing vehicular safety in the modern era.

2.3 Standards

Various standards used in this project are:

1. ISO 26262 - Road Vehicles - Functional Safety:

This standard is crucial for ensuring functional safety in automotive systems. Given that VehiGuard involves safety-critical functions like accident detection and prevention, adhering to ISO 26262 principles would be beneficial.

2. IEEE 802.11 - Wireless LAN Standards:

If VehiGuard involves wireless communication for transmitting data, compliance with IEEE 802.11 standards is essential for ensuring secure and reliable wireless communication protocols.

3. ISO 11898 - Controller Area Network (CAN):

CAN standards might be relevant for the communication between different components within the vehicle, especially if VehiGuard integrates with the vehicle's existing CAN bus system.

4. ISO 13485 - Medical devices - Quality management systems:

Depending on the specific sensors used in VehiGuard, particularly if they are biometric or healthrelated, ISO 13485 might be relevant to ensure quality management in the development of medical device components.

5. ISO 9001 - Quality Management System:

Adhering to ISO 9001 standards ensures best practices in quality management across the entire project development, contributing to the reliability of VehiGuard.

6. Cybersecurity Standards (e.g., ISO/IEC 27001):

Given the integration of IoT and cloud-based systems, adherence to cybersecurity standards such as ISO/IEC 27001 is crucial to ensure the security of data transmission and protect the system from potential cyber threats.

7. Automotive Ethernet Standards (e.g., IEEE 802.3bw):

If VehiGuard involves wired communication within the vehicle, compliance with Automotive Ethernet Standards, such as IEEE 802.3bw, might be relevant for ensuring compatibility and reliability in wired communication protocols.

It's important to note that the specific standards applicable to the VehiGuard project may depend on the detailed specifications, technologies used, and the intended deployment region. Always check and comply with the latest revisions of these standards to ensure ongoing compatibility and adherence to industry norms.

2.4 System Details

This section describes the software and hardware details of the system:

2.4.1 Software Details

i) Android Application

For the visualization of real-time data and seamless monitoring of the Vehiguard system, the Android application "ThingView" was employed. ThingView serves as a dedicated platform for interfacing with ThingSpeak channels, offering an intuitive and user-friendly interface. The application allows for the effortless visualization of data from the various sensors integrated into the Vehiguard system.

ThingView Overview:

ThingView is an Android application designed to facilitate the visualization of ThingSpeak channels with ease. It simplifies the process of monitoring data generated by the Vehiguard system in real-time, offering a streamlined user experience.

Integration Process:

To integrate ThingView into our system, the following steps were undertaken:

1. ThingSpeak Channel Setup:

- A ThingSpeak account was created and configured to host the data generated by the Vehiguard system.

- A dedicated ThingSpeak channel was set up, defining specific fields to represent data from sensors such as drowsiness detection, alcohol levels, seat belt status, and accident detection.

2. Raspberry Pi Configuration:

- The Raspberry Pi, a key component of the Vehiguard system, was configured to collect data from various sensors.
- Custom code was developed on the Raspberry Pi to process and transmit the collected data to the ThingSpeak channel.

3. ThingView App Configuration:

- ThingView was installed on Android devices designated for monitoring the Vehiguard system.
- The application was configured by entering the unique ThingSpeak channel ID associated with our Vehiguard project.

4. Real-Time Visualization:

- Upon configuration, ThingView seamlessly visualized real-time data from the Vehiguard system.
- The app respected channel settings, displaying data according to configured preferences for public channels.

User Interface:

ThingView provides an intuitive user interface, allowing users to monitor critical parameters at a glance. The app supports various chart types, ensuring a comprehensive representation of the Vehiguard system's performance.

Benefits:

The integration of ThingView enhances the accessibility and user experience of the Vehiguard system. With a focus on simplicity and functionality, ThingView contributes to the project's objective of ensuring vehicular safety through effective data visualization.

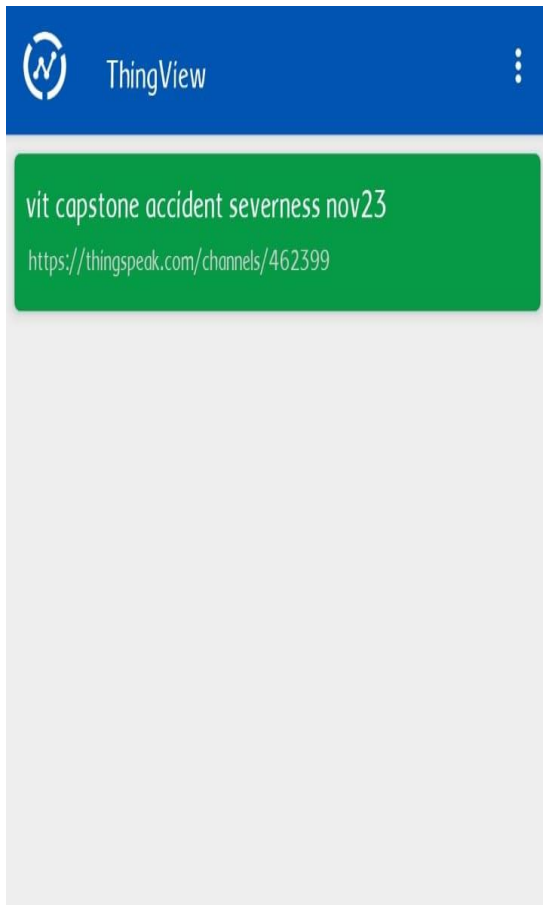


Fig7 ThingView App



Fig8 Location

2.4.2 Hardware Details

i) Raspberry Pi

The Raspberry Pi 3, a versatile single-board computer, serves as the central computing unit in the Vehiguard system. With a quad-core ARM Cortex-A53 processor and built-in wireless connectivity, it efficiently processes data from integrated sensors, facilitating real-time monitoring of driver behavior and vehicular safety parameters. Its compact form factor and GPIO (General Purpose Input/Output) pins make it an ideal platform for the seamless integration of hardware components in the project.



Fig9 Raspberry Pi

ii) **Alcohol Sensor**

The MQ-3 alcohol sensor is a semiconductor-based gas sensor capable of detecting alcohol vapors in the air. It operates on the principle of resistance changes in response to the presence of alcohol. As the concentration of alcohol increases, the sensor's resistance alters, providing an analog signal proportional to the gas concentration. MQ3 sensors are widely used in projects related to safety and monitoring systems, including those designed to prevent drunk driving by integrating alcohol sensing capabilities into vehicular systems like Vehiguard.



Fig10 Alcohol Sensor

iii) **GPS Antenna**

The GPS antenna utilized in the Vehiguard project enhances the system's location tracking capabilities. Designed to receive signals from Global Positioning System (GPS) satellites, the antenna provides accurate geographical coordinates in real-time. This information is pivotal for features such as continuous location tracking and precise emergency response in the event of an accident. The GPS antenna contributes to the project's overall safety objectives by ensuring reliable and up-to-date positioning data for the Vehiguard system.

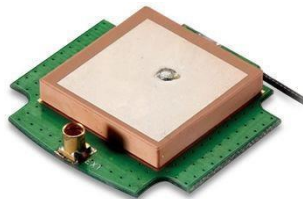


Fig11 GPS Antenna

iv) **GPS Module**

The GPS module integrated into the Vehiguard system serves as a crucial component for precise location tracking. By receiving signals from Global Positioning System (GPS) satellites, the module provides accurate geographical coordinates in real-time. This information is utilized to enhance safety features, such as continuous monitoring

of the vehicle's location and facilitating swift emergency responses in the event of an accident. The GPS module contributes to the overall effectiveness of the Vehiguard system by ensuring reliable and up-to-date positioning data.



Fig12 GPS Module

v) Motor Driver

The motor driver in the Vehiguard system is a critical electronic component responsible for controlling and managing the movements of motors, such as those used in alert systems or automatic responses. Acting as an interface between the Raspberry Pi and the motors, the motor driver interprets digital signals to regulate motor speed and direction, enabling the implementation of safety features, emergency responses, or other automated actions within the vehicle. Its role is pivotal in converting digital commands into precise motor control, contributing to the overall functionality and responsiveness of the Vehiguard system.

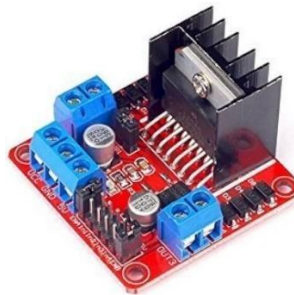


Fig13 Motor Driver

vi) Accelerometer Sensor

The accelerometer sensor employed in the Vehiguard system is designed to measure and detect changes in the vehicle's acceleration. This compact sensor provides real-time data on sudden speed variations and collision-like events, enabling the system to trigger accident detection algorithms promptly. Integrated into the Raspberry Pi-based setup, the accelerometer enhances the system's ability to respond to dynamic changes in vehicle dynamics, contributing to the project's core objective of enhancing vehicular safety.



Fig14 Accelerometer Sensor

vii) Switch

The switch used in the Vehiguard system is a fundamental electronic component that functions as an input device to control various functionalities. Typically connected to the Raspberry Pi, the switch allows the driver to manually engage or disengage specific safety features, such as enabling or disabling the system's alert mechanisms. This user-friendly interface enhances the system's adaptability to different driving scenarios and contributes to a more user-controlled safety experience within the vehicle.



Fig15 Switch

viii) Buzzer

The buzzer in the Vehiguard system serves as an audible alert mechanism. Connected to the Raspberry Pi, it emits sound signals in response to specific events, such as detecting driver drowsiness or non-compliance with seat belt usage. This audio feedback enhances driver awareness and contributes to the prevention of potential safety hazards. The buzzer's integration is a key element in providing real-time alerts and ensuring timely responses for a safer driving experience.



Fig16 Buzzer

VNC viewer and VNC server

VNC is a graphical desktop sharing system that allows one to remotely control the desktop interface of one computer (running VNC Server) from another computer or mobile device (running VNC Viewer). VNC Viewer transmits the keyboard and either mouse or touch events to VNC Server and receives updates to the screen in return.

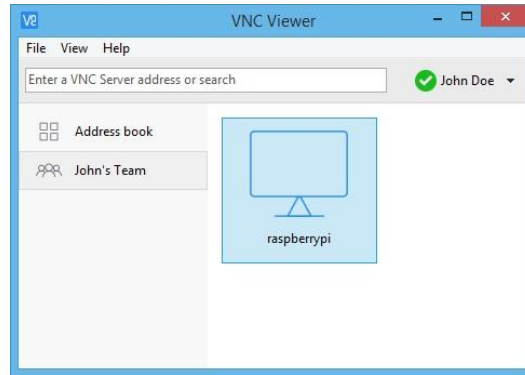


Fig17 VNC Viewer

VNC Connect from Real-VNC is included with Raspbian. It consists of both VNC Server, which allows us to control your Raspberry Pi remotely, and VNC Viewer, which allows us to control desktop computers remotely from our Raspberry Pi.

1. Enabling VNC Server

On the Raspberry Pi, run the following commands to make sure the latest version of VNC is installed. Follow the following steps: In the Terminal enter these commands

- `sudo apt-get update`
- `sudo apt-get install realvnc-vnc-server realvnc-vnc-viewer`

Now enable VNC Server. You can do this graphically or at the command line.

- Enabling VNC Server graphically ○ On your Raspberry Pi, boot into the graphical desktop.
 - Select **Menu > Preferences > Raspberry Pi Configuration > Interfaces**.
 - Ensure **VNC** is **Enabled**.
- Enabling VNC Server at the command line ○ You can enable VNC Server at the command line
 - `sudo raspi-config`
 - Now, enable VNC Server by doing the following:
 - Navigate to **Interfacing Options**.
 - Scroll down and select **VNC > Yes**.

2. Connecting to your Raspberry Pi with VNC Viewer There are two ways to connect to your Raspberry Pi.

- Establishing a direct connection-Direct connection is quick and simple providing you're joined to the same private local network as your Raspberry Pi. For example, this might be a wired or wireless network at home, at school, or in the office).
- On your Raspberry Pi (using a terminal window or via SSH) use these instructions or run `ifconfig` to discover your private IP address.
- On the device you'll use to take control, download VNC Viewer. For best results, use the compatible app from RealVNC.
- Enter your Raspberry Pi's private IP address into VNC Viewer:

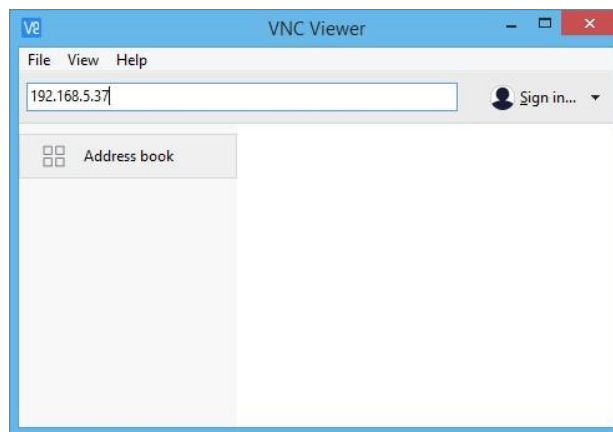


Fig18 VNC Viewer Address

- Establishing a cloud connection- Cloud connections are convenient and encrypted end-to-end. They are highly recommended for connecting to your Raspberry Pi over the internet.
- On your Raspberry Pi, sign in to VNC Server using your new RealVNC account credentials:

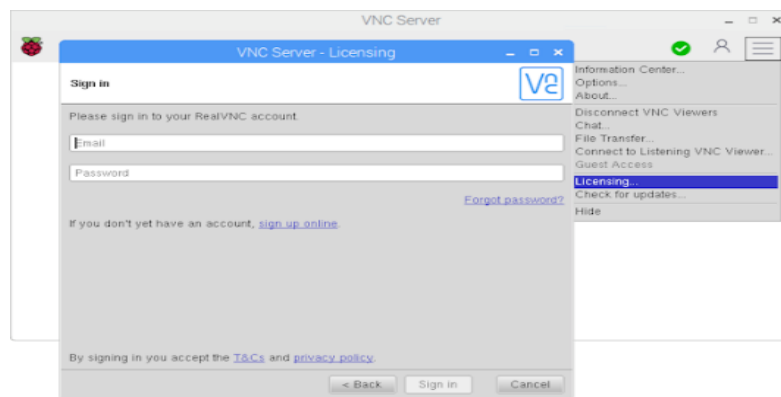


Fig19 VNC Server

- On the device you'll use to take control, download VNC Viewer.
- Sign in to VNC Viewer using the same RealVNC account credentials, and then connect to your Raspberry Pi or set up a new connection:

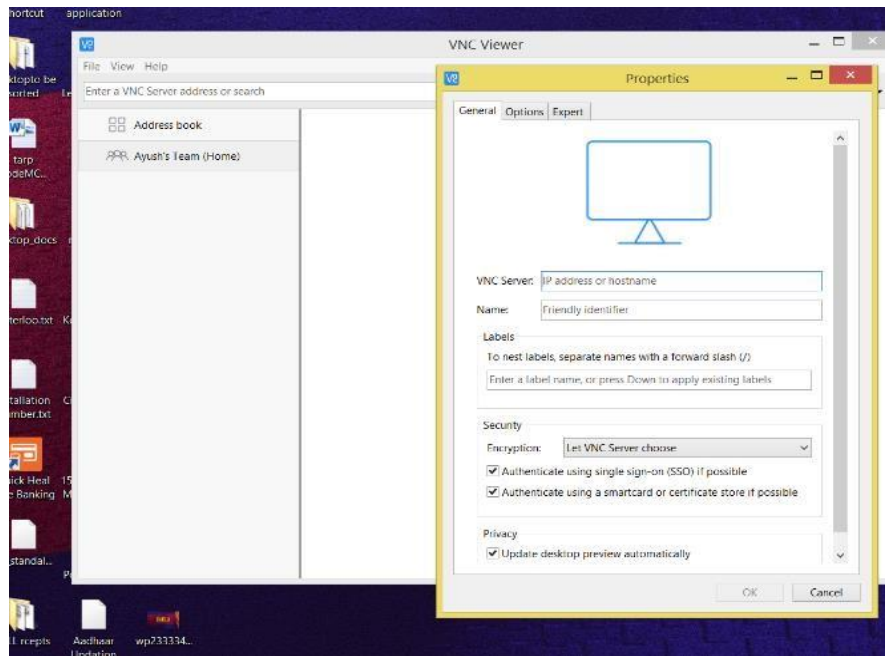


Fig20 VNC Properties Window

- NOTE: IP address here is Internet IP address which is different from Ethernet IP (LAN) address as both Ethernet port and built in WIFI modules have different MAC addresses

Before initiating the VNC Server setup, the installation process for Raspberry Pi must be completed. Upon the initial startup of the Raspberry Pi, users are greeted with the Welcome to Raspberry Pi application, which provides step-by-step guidance for the initial setup. This essential setup phase ensures that the Raspberry Pi is ready for subsequent configurations, including the establishment of the VNC Server. Sometimes it is not convenient to work directly on the Raspberry Pi as sometimes we need to work on it from another device by remote control .VNC is a graphical desktop sharing system that allows us to remotely control the desktop interface of one computer (running VNC Server) from another computer or mobile device (running VNC Viewer). VNC Viewer transmits the keyboard and either mouse or touch events to VNC Server and receives updates to the screen in return.

CHAPTER 3

3. COST ANALYSIS

3.1 List of components and their cost

The costs of the various components used in this project are given below in Table 3.1.

Table 3.1 List of components and their costs

COMPONENT	COST
Raspberry Pi	₹ 8000
Accelerometer Sensor	₹ 400
Alcohol Sensor	₹ 360
GPS Antenna	₹ 274
GPS Module	₹ 450
Motor Driver	₹ 261
Switch	₹ 60
Buzzer	₹ 30
TOTAL	₹ 9835

CHAPTER 4

RESULTS AND DISCUSSIONS

1. DROWSINESS ALERT:

The EAR (Eye Aspect Ratio) algorithm starts working when it catches an eye close in the video stream. If the ratio falls under a threshold value for the continuous frames, then it is registered as the blink and then the driver is alerted using the sound system kept in the driver's place.

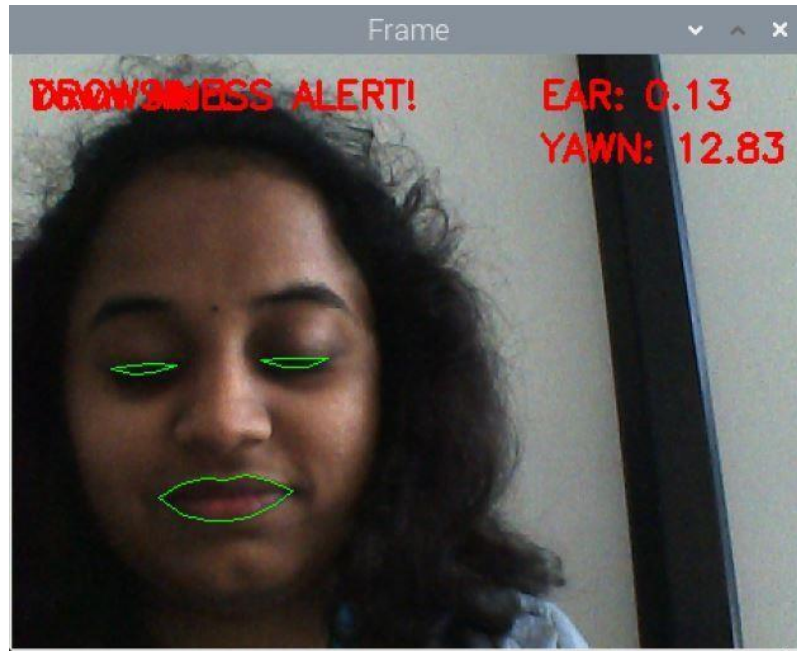


Fig21 Drowsiness alert 1

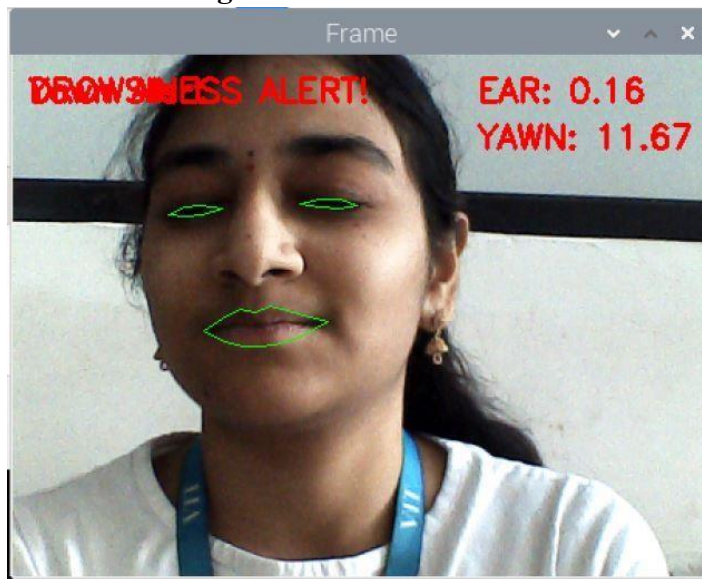


Fig22 Drowsiness alert 2

2. YAWN ALERT

The working of EAR and MAR is inversely proportional to each other. The alarm works on finding the eyes close and mouth open. Yawning is identified as the inducing factor of drowsiness. If the driver keeps on yawning beyond the fixed threshold such as 20 times it is detected as drowsy. For a person, 5–10 is negligible; when it counts to 11–20 it is termed as a little yawn. The alert system starts functioning for yawn when it crosses the threshold.

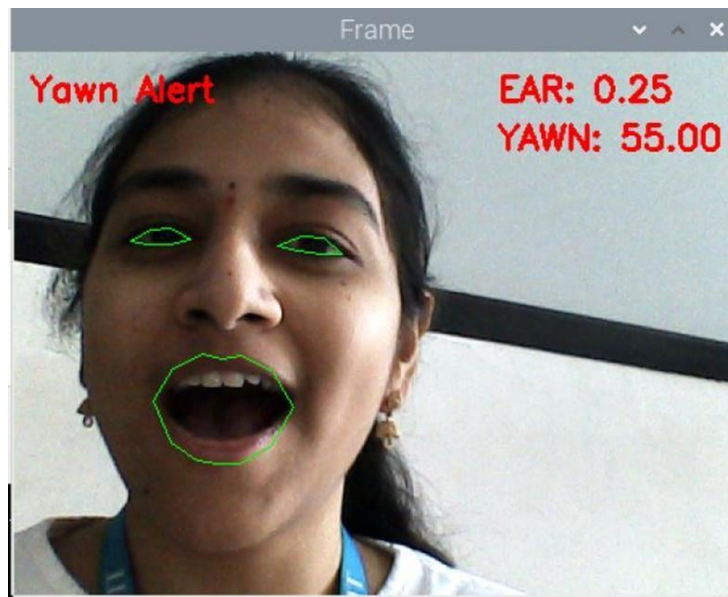


Fig23 Yawn alert 1

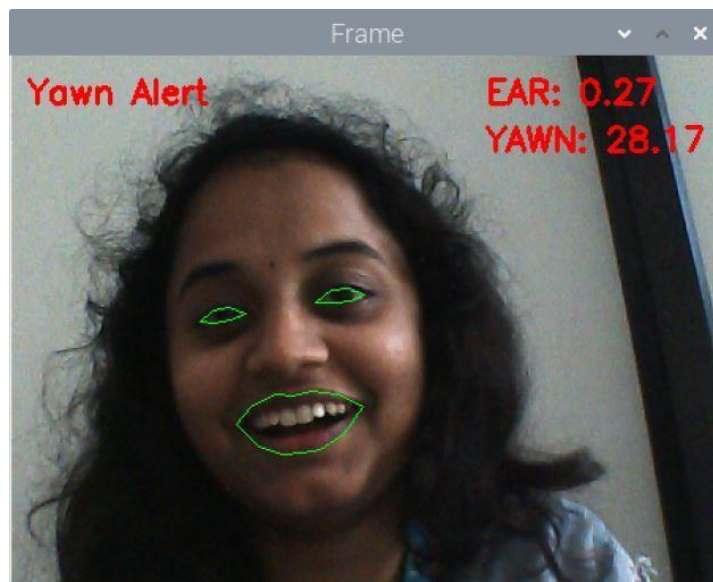


Fig24 Yawn alert 2

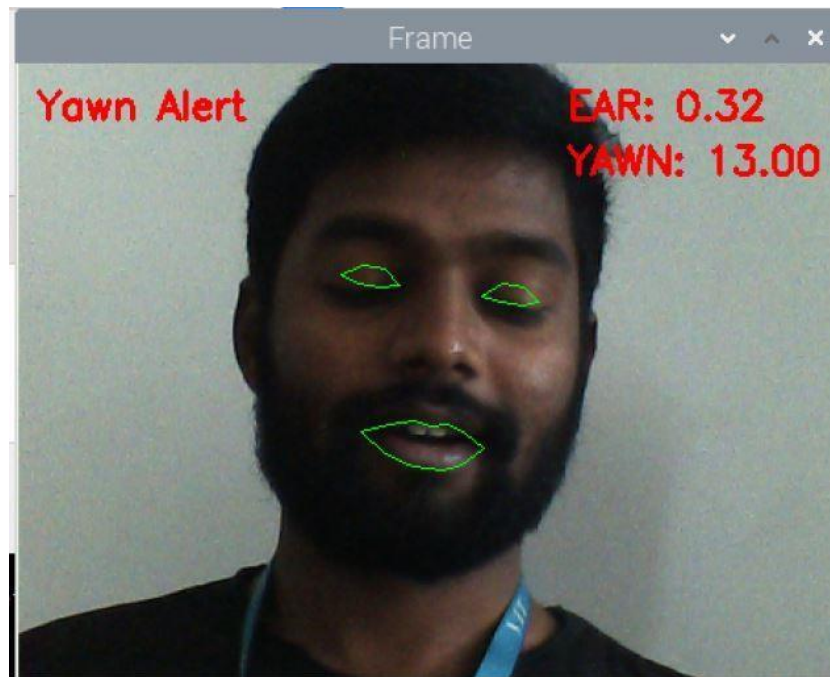


Fig25 Yawn alert 3

From the above pics, The Yawn Alert and Drowsiness Alert images serve as compelling evidence of Vehiguard's practical effectiveness in real-world scenarios. Capturing moments of driver fatigue, these images highlight the system's precise detection capabilities, showcasing its ability to identify yawning and drowsiness accurately. The integration of facial landmark analysis into the safety framework is visibly successful, as depicted in these snapshots. These visuals reinforce Vehiguard's commitment to preventing potential distractions and ensuring the utmost alertness of drivers, contributing significantly to the overall goal of enhancing road safety. The demonstrated accuracy and responsiveness of the system underscore its practical applicability in mitigating the risks associated with drowsy driving.

3.SMS ALERT

If a vehicle is involved in an accident, emergency assistance in the car gives message-alert services through SMS about the driver. Figs. 26 and 27 show that they provide precise and proximate information like the latitude and longitude of the accident zone.

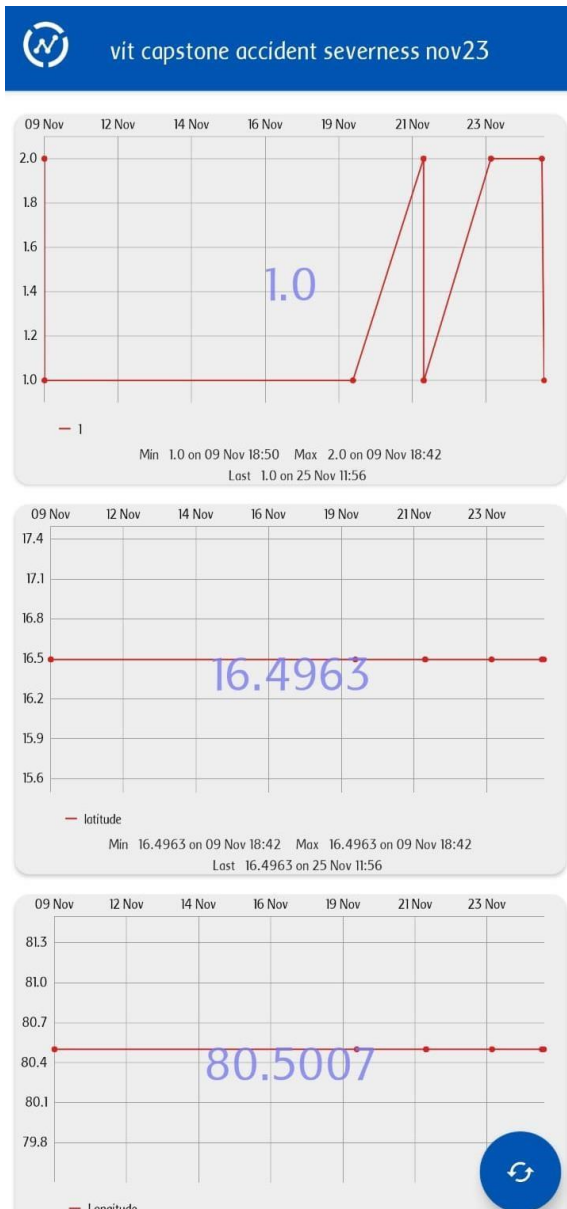


Fig26 GPS location1

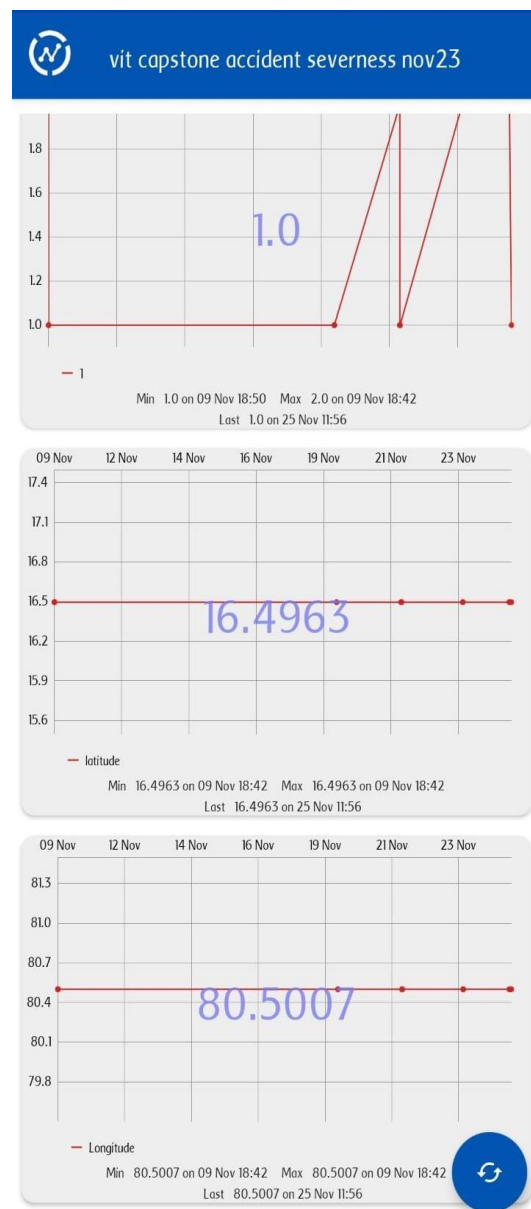


Fig27 GPS location2

The KNN model with optimal hyperparameters (n_neighbors = 5, weights =uniform') achieved an accuracy of 100%. Fig. 28 shows the confusion matrix of the KNN model.

```
Best Parameters: {'n_neighbors': 5, 'weights': 'uniform'}
Accuracy: 1.0
Confusion Matrix:
[[ 6  0  0]
 [ 0  4  0]
 [ 0  0 23]]
Classification Report:
              precision    recall  f1-score   support

     1         1.00      1.00      1.00         6
     2         1.00      1.00      1.00         4
     3         1.00      1.00      1.00        23

 accuracy          1.00
macro avg          1.00      1.00      1.00        33
weighted avg          1.00      1.00      1.00        33
```

Fig28 KNN Confusion Matrix

The SVC model with tuned hyperparameters (C = 1, gamma =scale') exhibited an accuracy of 100%. Fig. 29 shows the confusion matrix of the SVC model.

```
Best Parameters: {'C': 1, 'gamma': 'scale'}
Accuracy: 1.0
Confusion Matrix:
[[ 6  0  0]
 [ 0  4  0]
 [ 0  0 23]]
Classification Report:
              precision    recall  f1-score   support

     1         1.00      1.00      1.00         6
     2         1.00      1.00      1.00         4
     3         1.00      1.00      1.00        23

 accuracy          1.00
macro avg          1.00      1.00      1.00        33
weighted avg          1.00      1.00      1.00        33
```

Fig29 SVC Confusion Matrix

Logistic regression, without hyperparameter tuning, achieved an accuracy of 88%. Fig. 30 shows the confusion matrix of the logistic regression model.

```

Logistic Regression:
Accuracy: 0.8787878787878788
Confusion Matrix:
[[ 6  0  0]
 [ 3  0  1]
 [ 0  0 23]]
Classification Report:

```

	precision	recall	f1-score	support
1	0.67	1.00	0.80	6
2	0.00	0.00	0.00	4
3	0.96	1.00	0.98	23
accuracy			0.88	33
macro avg	0.54	0.67	0.59	33
weighted avg	0.79	0.88	0.83	33

Fig30 Logistic Regression Confusion Matrix

Hardware Setup

The hardware setup, depicted in the accompanying image, seamlessly integrated Raspberry Pi, Pi Camera, and sensors for robust real-time monitoring in Vehiguard. This configuration underpinned the project's success in implementing advanced safety features like drowsiness detection, seat belt detection and alcohol sensing.

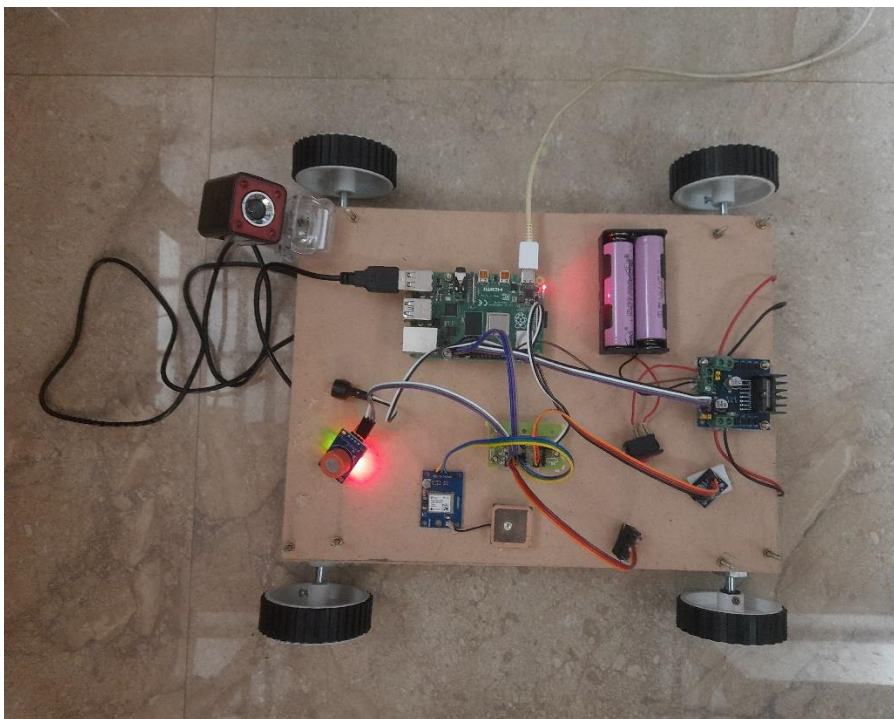


Fig31 Hardware setup

CHAPTER 5

CONCLUSION AND FUTURE WORK

In conclusion, Vehiguard represents a groundbreaking application of technology to enhance road safety. The project showcases the capability of innovative solutions to mitigate common driving risks, offering a comprehensive approach to driver assistance and accident prevention. As Vehiguard continues to evolve, its impact on road safety has the potential to create a paradigm shift, fostering safer road environments for drivers, passengers, and pedestrians alike.

Future Scope:

1. Expand Safety Features:

Vehiguard can further broaden its safety features by incorporating advanced technologies and sensors. Integration with more sophisticated algorithms for driver behavior analysis, collision prediction, and adaptive cruise control can elevate the system's ability to prevent accidents and enhance overall driving safety.

2. Integration with Smart Infrastructure:

Future iterations of Vehiguard could explore integration with smart city infrastructure. Collaborating with traffic management systems, traffic lights, and other smart devices can enhance real-time communication, providing a holistic approach to traffic safety and efficiency.

3. Ambulance Coordination:

Extend the project's scope by integrating with emergency services. Vehiguard could automatically alert and coordinate with ambulance services in the event of a severe accident, providing crucial information such as location, impact severity, and potential injuries. This feature aims to expedite emergency responses and improve outcomes for accident victims.

4. Vehicle Health Monitoring:

Enhance Vehiguard's capabilities by incorporating real-time vehicle health monitoring. Integrating sensors to assess engine performance, tire pressure, and other critical parameters can contribute to proactive maintenance alerts, reducing the likelihood of unexpected breakdowns and ensuring optimal vehicle performance.

5. Cybersecurity Measures:

With the increasing integration of IoT and cloud-based systems, future developments should prioritize robust cybersecurity measures. Ensuring the security of data transmission and protecting the system from potential cyber threats is crucial for maintaining the integrity and reliability of Vehiguard.

6. User Interface Improvements:

Consider enhancing the user interface of the system, possibly through a dedicated mobile application or an intuitive dashboard. A user-friendly interface can provide drivers with valuable insights into their driving behavior, encourage adherence to safety measures, and offer a more interactive experience with the Vehiguard system.

7. Legislation and Standardization:

Work towards aligning Vehiguard with emerging safety standards and regulations. Collaboration with transportation authorities and policymakers can facilitate the integration of the system into broader road safety initiatives, ensuring its widespread adoption and impact.

8. Driver Profiling and Personalization:

Explore the incorporation of machine learning algorithms to create individualized driver profiles. By understanding unique driving behaviors and patterns, Vehiguard can provide personalized safety recommendations and alerts tailored to each driver's habits.

As Vehiguard continues to evolve and embrace these future prospects, it stands poised to revolutionize the landscape of road safety, advancing towards a future where accidents are not only mitigated but prevented through intelligent, technology-driven solutions.

CHAPTER 6

APPENDIX

DROWSINESS YAWN CODE

```
from scipy.spatial import distance as dist
from imutils.video import VideoStream
from imutils import face_utils
from threading import Thread
import numpy
as np
import argparse
import imutils
import time
import dlib
import cv2
import os
import serial
import RPi.GPIO as GPIO

import smbus
import time
from time import
sleep
import sys
import sys
import urllib.request

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

als = 20
m1 = 26
m2 = 19
m3 = 13
m4 = 6
sb=4
buz=17
GPIO.setup(buz, GPIO.OUT)
GPIO.setup(m1, GPIO.OUT)
GPIO.setup(m2, GPIO.OUT)
GPIO.setup(m3, GPIO.OUT)
GPIO.setup(m4, GPIO.OUT)
GPIO.setup(als, GPIO.IN)
GPIO.setup(sb, GPIO.IN)
GPIO.output(buz,0)

def alarm(msg):
```

```

    global alarm_status
global alarm_status2
    global saying

    while alarm_status:
print('call')        s =
'espeak "'+msg+'"'
os.system(s)

    if alarm_status2:
print('call')        saying =
True        s = 'espeak "' +
msg + '"'      os.system(s)
        saying = False

def eye_aspect_ratio(eye):    A =
dist.euclidean(eye[1], eye[5])
B = dist.euclidean(eye[2], eye[4])

C = dist.euclidean(eye[0], eye[3])

    ear = (A + B) / (2.0 * C)

    return ear

def final_ear(shape):
(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]

    leftEye = shape[lStart:lEnd]
    rightEye = shape[rStart:rEnd]

    leftEAR = eye_aspect_ratio(leftEye)
    rightEAR = eye_aspect_ratio(rightEye)

    ear = (leftEAR + rightEAR) / 2.0
    return (ear, leftEye, rightEye)

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

    low_lip = shape[56:59]
    low_lip = np.concatenate((low_lip, shape[65:68]))

    top_mean = np.mean(top_lip, axis=0)

```

```

low_mean = np.mean(low_lip, axis=0)

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

ap = argparse.ArgumentParser()
ap.add_argument("-w", "--webcam", type=int, default=0,
                help="index of webcam on system") args
= vars(ap.parse_args())

EYE_AR_THRESH = 0.2
EYE_AR_CONSEC_FRAMES = 1 YAWN_THRESH
= 10
alarm_status = False alarm_status2
= False
saying = False
COUNTER = 0

print("-> Loading the predictor and detector...") #detector
= dlib.get_frontal_face_detector()
detector = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")      #Faster but less
accurate
predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')

print("-> Starting Video Stream")
vs = VideoStream(src=args["webcam"]).start()
#vs= VideoStream(usePiCamera=True).start()      #For Raspberry Pi time.sleep(1.0)
GPIO.output(m1,1)
GPIO.output(m2,0)
GPIO.output(m3,1) GPIO.output(m4,0)
ii=0

while True:

    frame = vs.read()
    frame = imutils.resize(frame, width=450)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    #rects = detector(gray, 0)      rects =
detector.detectMultiScale(gray, scaleFactor=1.1,
                        minNeighbors=5, minSize=(30, 30),
                        flags=cv2.CASCADE_SCALE_IMAGE)

    #for rect in rects:

```

```

for (x, y, w, h) in rects:      rect = dlib.rectangle(int(x),
int(y), int(x + w),int(y + h))

    shape = predictor(gray, rect)
    shape = face_utils.shape_to_np(shape)

    eye = final_eye(shape)
    ear = eye[0]
leftEye = eye [1]
rightEye = eye[2]

    distance = lip_distance(shape)

    leftEyeHull = cv2.convexHull(leftEye)    rightEyeHull
= cv2.convexHull(rightEye)    cv2.drawContours(frame,
[leftEyeHull], -1, (0, 255, 0), 1)    cv2.drawContours(frame,
[rightEyeHull], -1, (0, 255, 0), 1)

    lip = shape[48:60]
    cv2.drawContours(frame, [lip], -1, (0, 255, 0), 1)

    if ear < EYE_AR_THRESH:
        COUNTER += 1

        if COUNTER >= EYE_AR_CONSEC_FRAMES:
            if alarm_status == False:                alarm_status = True
                t = Thread(target=alarm, args=('Alert and wake up sir',))
GPIO.output(buz,1)                t.daemon = True
                t.start()
            ii=ii+1
            if(ii>2):
                GPIO.output(buz,1)
            time.sleep(0.5)
            GPIO.output(buz,0)                if(ii>4):
                GPIO.output(buz,1)
            time.sleep(0.5)                GPIO.output(buz,0)
                GPIO.output(m1,0)
                GPIO.output(m2,0)
                GPIO.output(m3,0)
            GPIO.output(m4,0)                wp
            =
            urllib.request.urlopen("https://api.thingspeak.com/update?api_key=5JXYCST9SUCAJQS4&field
1=2" + "&field2=" +str(16.4963) + "&field3=" +str(80.5007))

        while(1):
            time.sleep(1)

```

```

        cv2.putText(frame, "DROWSINESS ALERT!", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

else:
    COUNTER = 0
    alarm_status = False

    if (distance > YAWN_THRESH):
        cv2.putText(frame, "Yawn Alert", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)        if
        alarm_status2 == False and saying == False:
            alarm_status2 = True
            t = Thread(target=alarm, args=('Alert and take some fresh air sir',))
#GPIO.output(buz,1)            t.daemon = True
            t.start()
        ii=ii+1
        if(ii>2):
            GPIO.output(buz,1)
            time.sleep(0.5)            GPIO.output(buz,0)
            #wp =
            urllib.request.urlopen("https://api.thingspeak.com/update?api_key=VQTNCH02VBC0YVJW&fie
ld1=2" + "&field2=" +str(long_in_degrees) + "&field3=" +str(lat_in_degrees))

else:
    alarm_status2 = False

    cv2.putText(frame, "EAR: {:.2f}".format(ear), (300, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)        cv2.putText(frame,
"YAWN: {:.2f}".format(distance), (300, 60),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    if key == ord("q"):
        break

cv2.destroyAllWindows()
vs.stop()

```

ALCOHOL ACCELEROMETER SEATBELT SENSOR

```
#python drowniness_yawn.py --webcam webcam_index

from scipy.spatial import distance as dist
from imutils.video import VideoStream
from imutils import face_utils
from threading import Thread
import numpy
as np
import argparse
import imutils
import time
import dlib
import cv2
import os
import serial
import RPi.GPIO as GPIO

import smbus
import time
from time import sleep
import sys
import sys
import urllib.request

bus = smbus.SMBus(1)

bus.write_byte_data(0x53, 0x2C, 0x0B)
value = bus.read_byte_data(0x53, 0x31)
value &= ~0x0F; value |= 0x0B; value
|= 0x08; bus.write_byte_data(0x53,
0x31, value) bus.write_byte_data(0x53,
0x2D, 0x08)
#wp
urllib.request.urlopen("https://api.thingspeak.com/update?api_key=MW6ESOWJ00JZVEC3&fiel
d1=0" + "&field2=" +str(0) + "&field3=" +str(0))

def getAxes():
    bytes = bus.read_i2c_block_data(0x53, 0x32, 6)

    x = bytes[0] | (bytes[1] <<
8) if(x & (1 << 16 - 1)):
x = x - (1<<16)

    y = bytes[2] | (bytes[3] <<
8) if(y & (1 << 16 - 1)):
y = y - (1<<16)

    z = bytes[4] | (bytes[5] <<
8) if(z & (1 << 16 - 1)):
z = z - (1<<16)
```

```

    x = x * 0.004
    y = y * 0.004    z
    = z * 0.004

    x = x * 9.80665
    y = y * 9.80665
    z = z * 9.80665

    x = round(x, 1)
    y = round(y, 1)
    z = round(z, 1)

    ## print(" x = %.3f ms2" %x)
    ## print(" y = %.3f ms2" %y)
    ## print(" z = %.3f ms2" %z)
    ## print("\n\n")

    return {x, y}

```

```

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

als = 20 m1 = 26 m2 = 19 m3
= 13 m4 = 6 sb=21 buz=17
GPIO.setup(buz, GPIO.OUT)
GPIO.setup(m1, GPIO.OUT)
GPIO.setup(m2, GPIO.OUT)
GPIO.setup(m3, GPIO.OUT)
GPIO.setup(m4, GPIO.OUT)
GPIO.setup(als, GPIO.IN) GPIO.setup(sb,
GPIO.IN)
GPIO.output(buz,0)

```

```

kk=0 def GPS_Info():
    global NMEA_buff
    global lat_in_degrees
    global
    long_in_degrees
    nmea_time = []
    nmea_latitude = []
    nmea_longitude = []
    nmea_time =

```



```

NMEA_buff[0]
#extract time from
GPGGA string
nmea_latitude =
NMEA_buff[1]
#extract latitude from
GPGGA string
nmea_longitude =
NMEA_buff[3]
#extract longitude
from GPGGA string

#print("NMEA Time: ", nmea_time,'\n')
#print ("NMEA Latitude:", nmea_latitude,"NMEA Longitude:", nmea_longitude,'\n')
try:
    lat = float(nmea_latitude)          #convert string into float for calculation
    longi = float(nmea_longitude)       #convert string into float for calculation
except:
    lat=0    longi=0
    lat_in_degrees = convert_to_degrees(lat)    #get latitude in degree decimal format
    long_in_degrees = convert_to_degrees(longi) #get longitude in degree decimal format

def convert_to_degrees(raw_value):
    decimal_value = raw_value/100.00    degrees
    = int(decimal_value)
    mm_mmmm = (decimal_value - int(decimal_value))/0.6
    position = degrees + mm_mmmm    position = "%.4f"
    %(position)
    return position

gpgga_info = "$GPGGA,"
ser = serial.Serial ("/dev/ttyS0")    #Open port with baud rate
GPGGA_buffer = 0 NMEA_buff
= 0
lat_in_degrees = 0
long_in_degrees = 0

time.sleep(1.0)
GPIO.output(m1,1)
GPIO.output(m2,0)
GPIO.output(m3,1) GPIO.output(m4,0)
ii=0

while True:

```

```

received_data = (str)(ser.readline())          #read NMEA string received
GPGGA_data_available = received_data.find(gpgga_info)  if(kk==0):    lat_in_degrees=0
lat_in_degrees=0  if (GPGGA_data_available>0):
    kk=1
    GPGGA_buffer = received_data.split("$GPGGA,",1)[1] #store data coming after "$GPGGA,"
string
    NMEA_buff = (GPGGA_buffer.split(','))          #store comma separated data in buffer
    GPS_Info()                                     #get time, latitude, longitude
    map_link = 'http://maps.google.com/?q=' + str(lat_in_degrees) + ',' + str(long_in_degrees)
#create link to plot location on Google map

    map_link = 'http://maps.google.com/?q=' + str(16.4963) + ',' + str(80.5007) #create link to
plot location on Google map  print(map_link)
print()

    x,y=getAxes()
print("X:" + str(x))
print("Y:" + str(y))
    #time.sleep(3)

    aval=1-GPIO.input(als)
print("ALCOHOLIC:" + str(aval))

    sval=1-GPIO.input(sb)
print("SB:" + str(sval))
if(sval==1):
    GPIO.output(buz,1)
time.sleep(0.5)
GPIO.output(buz,0)
time.sleep(0.5)

if(aval==1):
    GPIO.output(buz,1)
    GPIO.output(m1,0)
    GPIO.output(m2,0)
    GPIO.output(m3,0)
    GPIO.output(m4,0)

if(x<-6.5 or y<-6.5 or x>6.5 or y>6.5):
    print('Accident...')
GPIO.output(buz,1)
    GPIO.output(m1,0)
    GPIO.output(m2,0)
    GPIO.output(m3,0)
    GPIO.output(m4,0)
wp

```

=

```
urllib.request.urlopen("https://api.thingspeak.com/update?api_key=5JXYCST9SUCAJQS4&field1=1" + "&field2=" + str(16.4963) + "&field3=" + str(80.5007))
```

```
while(1):
    time.sleep(1)
```

ACCELEROMETER CODE

```
import smbus import
time from time import
sleep import sys
```

```
bus = smbus.SMBus(1)
```

```
bus.write_byte_data(0x53, 0x2C, 0x0B)
value = bus.read_byte_data(0x53, 0x31)
value &= ~0x0F; value |= 0x0B; value
|= 0x08; bus.write_byte_data(0x53,
0x31, value)
bus.write_byte_data(0x53, 0x2D, 0x08)
```

```
def getAxes():
    bytes =
    bus.read_i2c_block_data(0x53, 0x32, 6)
```

```
x = bytes[0] | (bytes[1] <<
8) if(x & (1 << 16 - 1)):
    x = x - (1<<16)
```

```
y = bytes[2] | (bytes[3] <<
8) if(y & (1 << 16 - 1)):
    y = y - (1<<16)
```

```
z = bytes[4] | (bytes[5] <<
8) if(z & (1 << 16 - 1)):
    z = z - (1<<16)
```

```
x = x * 0.004
y = y * 0.004
z = z * 0.004
```

```
x = x * 9.80665
y = y * 9.80665
z = z * 9.80665
```

```
x = round(x, 4)
y = round(y, 4) z
= round(z, 4)
```

```

print("  x = %.3f
ms2" %x)
print("  y = %.3f
ms2" %y)
print("  z = %.3f
ms2" %z)
    print("\n\n")

    return {"x": x, "y": y, "z": z}

try:    while True:
getAxes()
time.sleep(2) except
KeyboardInterrupt:
    sys.exit()

```

SEVERITY CODE

```

import numpy as np import pandas as pd from sklearn.model_selection import
train_test_split, GridSearchCV from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression from sklearn.neighbors
import KNeighborsClassifier from sklearn.svm import SVC from sklearn.metrics
import accuracy_score, confusion_matrix, classification_report import seaborn as
sns

def perform_grid_search(model, param_grid, X_train, y_train, X_test, y_test):
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

    best_model        =        grid_search.best_estimator_
best_params = grid_search.best_params_

    print("Best Parameters:", best_params)

y_pred = best_model.predict(X_test)

acc = accuracy_score(y_test, y_pred)

```

```

cm = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy: ", acc)
print("Confusion Matrix:")
print(cm) print("Classification
Report:") print(report)

# Load the dataset accdt =
pd.read_csv("jai.csv") y =
accdt['RES']
X = accdt.drop(['RES'], axis=1)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=0)

# Standardize features scaler
= StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Hyperparameter tuning for KNN
knn_param_grid = {'n_neighbors': [3, 5, 7, 9], 'weights': ['uniform', 'distance']}
knn_model = KNeighborsClassifier() perform_grid_search(knn_model,
knn_param_grid, X_train, y_train, X_test, y_test)

# Hyperparameter tuning for SVC
svc_param_grid = {'C': [0.1, 1, 10], 'gamma': ['scale', 'auto']} svc_model = SVC()
perform_grid_search(svc_model, svc_param_grid, X_train, y_train, X_test, y_test)

# Logistic Regression (No hyperparameter tuning in this example)
regr = LogisticRegression(solver="liblinear").fit(X_train, y_train) regr_preds
= regr.predict(X_test)

```

```
regr_acc = accuracy_score(y_test, regr_preds) regr_cm  
= confusion_matrix(y_test, regr_preds) regr_report =  
classification_report(y_test, regr_preds)
```

```
print("\nLogistic Regression:")  
print("Accuracy: ", regr_acc)  
print("Confusion Matrix:")  
print(regr_cm)  
print("Classification Report:")  
print(regr_report)
```

```
import numpy as np import pandas  
as pd import matplotlib.pyplot as  
plt from matplotlib import  
rcParams from matplotlib.cm  
import rainbow
```

```
from sklearn.model_selection import train_test_split from  
sklearn.preprocessing import StandardScaler from  
sklearn.preprocessing import MinMaxScaler from  
sklearn.linear_model import LogisticRegression from  
sklearn.neighbors import KNeighborsClassifier from  
sklearn.svm import SVC from sklearn.tree import  
DecisionTreeClassifier from sklearn.ensemble import  
RandomForestClassifier from sklearn import datasets,  
linear_model from sklearn.metrics import  
accuracy_score, make_scorer
```

```
def takeInput():
```

```

inputValues = ['2.0','2.0','-2.2','-2.2']
print("\n")    final_Result =
knn_classifier.predict([inputValues])    final_Result =
svc_clf.predict([inputValues])    print(final_Result)

```

```

if    final_Result[0]    ==    1:
print('NORMAL CONDITION')    elif
final_Result[0]    ==    2:
print('AVERAGE    CONDITION')
elif final_Result[0] == 3:
    print('SEVERE    CONDITION')
else:
    print('GOOD condition')

```

```

accdt = pd.read_csv("jai.csv") y
= accdt['RES']
X = accdt.drop(['RES'], axis = 1)

```

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state = 0)
knn_classifier = KNeighborsClassifier(n_neighbors = 4) knn_classifier.fit(X_train,
y_train)

```

```

knn_preds = knn_classifier.predict(X_test) knn_acc =
accuracy_score(y_test, knn_preds) print("Accuracy with KNN: ",
accuracy_score(y_test, knn_preds))

```

```

svc_clf = SVC(gamma='scale') svc_clf.fit(X_train,y_train)
svc_preds = svc_clf.predict(X_test) svc_acc =
accuracy_score(y_test, svc_preds) print("Accuracy with SVC: ",
accuracy_score(y_test, svc_preds)) regr =
LogisticRegression(solver="liblinear").fit(X_train,y_train)

```

```
regr_preds = regr.predict(X_test) regr_acc =  
accuracy_score(y_test, regr_preds) print("Accuracy with LR: ",  
accuracy_score(y_test, svc_preds))
```

```
takeInput()
```


CHAPTER - 7

REFERENCES

- [1] Bodare, D., Ranmode, M., & Ghogare, P. (2015). Car accident prevention system. International journal of engineering research and general science, 3(2 Part 2), 2-10.
- [2] Kumar, J. M., Mahajan, R., Prabhu, D., & Ghose, D. (2016, December). Cost effective road accident prevention system. In 2016 2nd International Conference on Contemporary Computing and Informatics (IC3I) (pp. 353-357). IEEE.
- [3] Ali, H. M., & Alwan, Z. S. (2017). Car accident detection and notification system using smartphone. Saarbrücken: Lap Lambert Academic Publishing.
- [4] U. Alvi, M. A. K. Khattak, B. Shabir, A. W. Malik and S. R. Muhammad, "A Comprehensive Study on IoT Based Accident Detection Systems for Smart Vehicles," in IEEE Access, vol. 8, pp. 122480-122497, 2020, doi: 10.1109/ACCESS.2020.3006887.
- [5] Putra, I. P. E. S., Brusey, J., Gaura, E., & Vesilo, R. (2017). An event-triggered machine learning approach for accelerometer-based fall detection. Sensors, 18(1), 20.
- [6] Goyal, S. B., Bedi, P., Kumar, J., & Ankita. (2022). Realtime accident detection and alarm generation system over IoT. Multimedia Technologies in the Internet of Things Environment, Volume 2, 105-126.
- [7] Lakshmy, S., Gopan, R., Meenakshi, M. L., Adithya, V., & Elizabeth, M. R. (2022, March). Vehicle accident detection and prevention using iot and deep learning. In 2022 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES) (Vol. 1, pp. 22-27). IEEE.
- [8] Bedane, T. T., Assefa, B. G., & Mohapatra, S. K. (2021, November). Preventing Traffic Accidents through Machine Learning Predictive Models. In 2021 International Conference on Information and Communication Technology for Development for Africa (ICT4DA) (pp. 36-41). IEEE.
- [9] Lee, B. G., Lee, B. L., & Chung, W. Y. (2015). Wristband-type driver vigilance monitoring system using smartwatch. IEEE Sensors Journal, 15(10), 5624-5633.
- [10] Gbenga, D. E., Hamed, H. I., Lateef, A. A., & Opeyemi, A. E. (2017). Alcohol detection of drunk drivers with automatic car engine locking system. Nova Journal of Engineering and Applied Sciences, 6(1).
- [11] Mammadov, A. (2021). DERIVATION OF PRESCRIPTIVE ACCIDENT PREVENTION MODEL FROM PREDICTIVE MODELS USING ML ALGORITHMS (Master's thesis, Middle East Technical University).

- [12] Shayboun, M. (2022). Toward Accident Prevention Through Machine Learning Analysis of Accident Reports (Doctoral dissertation, Chalmers Tekniska Hogskola (Sweden)).
- [13] Bhakat, A., Chahar, N., & Vijayasherly, V. (2021, August). Vehicle Accident Detection & Alert System using IoT and Artificial Intelligence. In 2021 Asian Conference on Innovation in Technology (ASIANCON) (pp. 1-7). IEEE.
- [14] Ahmed, S., Hossain, M. A., Ray, S. K., Bhuiyan, M. M. I., & Sabuj, S. R. (2023). A study on road accident prediction and contributing factors using explainable machine learning models: analysis and performance. *Transportation research interdisciplinary perspectives*, 19, 100814.
- [15] Babu, C. G., Chandrashekhara, K. T., Verma, J., & Thungamani, M. (2021, December). Real time alert system to prevent Car Accident. In 2021 International Conference on Forensics, Analytics, Big Data, Security (FABS) (Vol. 1, pp. 1-4). IEEE.