Faculty of Engineering

Department of Electronics and Communication

"Fatigue Detection & Automatic Emergency Response Based on ADAS"

Designed & Implemented By:

Name	ID
Habiba Hassan Soliman	2018-11594
Youssef Ahmed Mohamed	2018-01616
Youssef Mamdouh Abdelhamid	2017-01124

Under The Supervision Of:

Associate Professor Dr. Mohamed Ali Mohamed Zrokany

Associate Professor in the Electronics department at the National Telecommunications Institute NTI

Acknowledgement

We would like to express our sincere gratitude to Dr. Mohamed El Zrokany for his guidance and encouragement throughout the whole journey of this project. We would not have had a better and more supporting instructor.

We also wish to thank the faculty members of our department, the department of Electronics and Communications who rendered their help during the period of our project work.

Finally, we would like to thank ITIDA for accepting our project as one of their sponsored projects.

Abstract

It has become significantly clear that the world is rapidly advancing in a way like none before. New technologies that, once, were just a dream, are coming to reality in a dramatic pace. From digitization to artificial intelligence, fast forward to the Internet of things concepts. This world is evolving like crazy. And what is better than bringing such technologies to the benefit of human beings, and the moral idea of shielding them and saving their vulnerable, and much valued lives?

This project is an approach to bring this ideology to reality.

Drivers are subject to tiredness while they're on the road, a single mistake can have a fatal effect upon them. ADAS are passive and active safety systems designed to catch up just where the human attention falters. Sensors, and Microcontrollers are used to form ADAS units, and with the help of these units, many road accidents can be significantly avoided.

Driver assistance systems represent a major field of innovation in the automotive industry domain. They automate, facilitate, and enhance vehicle structures by using long and medium range radar and vision systems.

This document dives into the complex world of artificial intelligence and embedded systems together, and how they can be united into one unit in order to form a sturdy system that is meant to combat the ugly reality of fatal road accidents.

In such a fast paced world, it is important to study the intersection of powerful, evolving systems such as Ai and embedded systems, and to delve into the technological details of how they work, in addition to the benefits and risks they impose.

Today, Ai is driving innovation, and shaping the future of transportation. This is a comprehensive overview of the cutting-edge technologies that are transforming the automotive industry, and paving the way for a safer, more efficient, and more connected driving experience.

This work provides a description of the project's problem statement, its objective, the methodology, the technical approach, the tools used to build the unit, hardware and software implementation, and the outcome of the project.

Table of Contents

List of Figures

List of Acronyms

ADAS	*Advanced Driver Assistance Systems*
ADC	Analog to Digital Converter
ABS	Anti Lock Braking System
ACC	Adaptive Cruise Control
AEB	Automatic Emergency Braking
AVR	Advanced Virtual Risc
CNN	Convolutional Neural Network
DIO	Digital Input Output
DC	Direct Current
GPS	Global Positioning System
GSM	Global System For Mobile Communication
ISR	Interrupt Service Routine
IC	Integrated Circuit
I2C	Inter Integrated Circuit
V2V	Vehicle to Vehicle
V2I	Vehicle to Infrastructure

Chapter 1: Introduction

1.1 History of Automotive Systems and Technology

Throughout the years, cars have played a crucial role in our everyday lives. From horse carriages in the Renaissance era, 17th century, to steam and combustion vehicles in the Victorian era. Karl Benz was known to be the first man to ever invent a car. Today we see gasoline vehicles everywhere, but with the growing global warming crisis, Electric and Hydrogen vehicles are much needed, as not only does a fuel cell produce no emissions at all, but it also turns waste into energy.

Figure 1 The Evolution Of Automotive Technology

Automotive systems have also evolved technologically. Earlier in the day, cars have only been a means of transportation with no further use, thus, over the years, this has been changing alongside every other technological change in the fast pace of the modern world. Cars have been adapting to every driver need while he is on the road. The scope of advancement has moved from mechanics to telematics,

electronics, and software systems. Early vehicles did not have doors, wind shields or indicators. During the early 20th century, new features were introduced in vehicles, which included seatbelts, windshields, speedometers, and rear-view mirrors. With improvement in features came the security aimed at preventing people from mishaps. The first passenger airbag was installed in 1973, and it was not until 1998 that airbags were mandated for all passenger vehicles. Ever since, manufacturers have adopted technological advancements (bringing in new features, smoother experience) to suppress consumer fatigue. These technological advancements include: Air conditioning, Cruise control, Bluetooth integration, Stability control, Digital dashboard displays, GPS satellite navigation, Connected cars, Autopilot & Automatic parking, and more and more including Driver assistance systems.

1.2Problem Statement & Statistics

Road accidents (Unintentional injuries) are always ranking as one of the leading death causes in every country in every continent. Moreover, they are the eighth prime death cause globally. According to the Global Road Safety Facility (GRSO), and the World Health Organization paper issued in 20th June 2022, 1.35 million people are the victims of crashes on the world's roads, most of whom are aged between 5 to 29 years. They are responsible for more fatalities than HIV/AIDS. In Egypt, 76% of road crash fatalities are in the economically productive age groups (15 to 64 years). This disaster is expected to cost the world about 1.8 trillion dollars by 2030. By taking a look at the map down below, it's obvious that the developing countries are the most affected. 93% of the world's casualties on the road occur in low and middle income countries. And not only are drivers the victims of crashes, but also the passengers, and the pedestrians in the street are. Consequently, the United Nations General Assembly has set a goal to reduce the number of deaths and injuries from road crashes to the half by 2023.

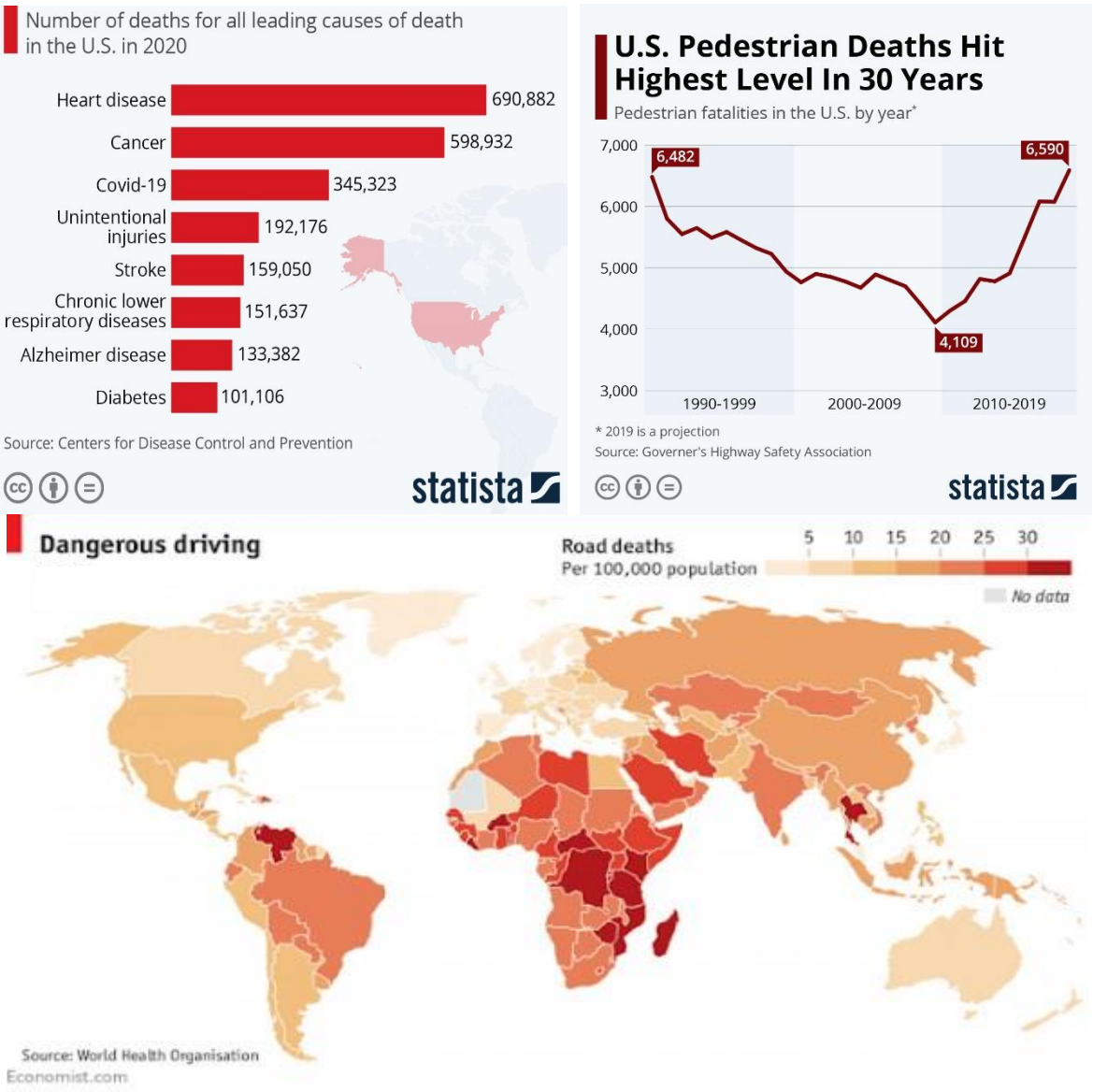


Figure 2 Global Road Accidents Statistics

Speaking of Egypt, in March 2022, Kamel Al Wazir, the Minister of Transport announced that the cost of road projects around Cairo had reached 300 billion Egyptian pounds. And although broken and unattended roads actually play a major role in car crashes, still specialists in the field of roads and bridges have emphasized that there is more to road accidents than just fixing the roads. According to the Central Agency for Public Mobilization and Statistics (CAPMAS) data issued in November 2021, there is a significant decline in the number of road accidents by the end of 2020. Around 56,789 injuries from road accidents were recorded compared to about 79,904 in 2019. The number of deaths due to road accidents also decreased to 6,164 in 2021, compared to 6,722 in 2020. Yet still despite this promising news. According to the NADA Foundation for

Safer Egyptian Roads, an Egyptian nongovernmental organization, the death rate due to road accidents in Egypt is among the highest in the world, with 42 deaths per 100,000 individuals.
The foundation said that traffic accidents are the No. 1 cause of death for Egyptian youth between the ages of 15 and 35, and that an average of four children die every day during or as a result of car accidents.

The foundation states that 50,000 people are injured or disabled every year due to traffic accidents.

1.3 Proposed Solution

In the wake of this news manufacturers have started thinking, what if cars can talk to each other (Vehicle to Vehicle Communication) ? What if cars can talk to the driver himself (Driver Assistance Systems) ? Or maybe they can just drive alone (Autonomous Driving Vehicles) ?

This project has conducted the driver assistance systems idea. The project involves more than one scope, these are: Embedded Systems, Embedded Linux, Deep learning, and ADAS (Advanced Driver Assistance Systems).

But what are ADAS systems in the first place ? ADAS are passive and active safety systems designed to catch up to human errors in real time and critical conditions such as driving a car. Their major role is analysing driving behaviours and taking real-time actions such as pedestrian detection & avoidance, traffic sign recognition , lane departure warning, speed limit indication, forward collision warning, blind spot detection & adaptive cruise control. These are examples of active safety systems.

Passive safety systems involve: Shatter resistant glass, three-point seat belts, airbags...etc.

ADAS uses Radar & Ultrasound sensors and cameras for such applications. These lifesaving systems incorporate the latest interface standards and run multiple vision-based algorithms to support real-time multimedia, vision co-processing, and sensor fusion subsystems.

There are six levels of vehicle autonomy. These systems occupy the 1st & 2nd levels of vehicle autonomy in which the car still needs the driver yet assists him.

The proposed solution in this project involves building a deep learning model (Based on ResNet50 version 2) to keep track of the driver's actions through a Raspberry Pi camera module. The images obtained by the camera are processed using a Raspberry Pi microcontroller, and this is where embedded Linux is involved. Furthermore, a photoplethysmography pulse sensor is embedded into the system for early heart attack detection. If the system recognizes any sign of drowsiness or unusual behaviour e.g. yawning, closed eyes for more than a couple of seconds, abnormal heart rate…etc. The car automatically shifts into the Adaptive Cruise Control mode, this is when the car slows down or speeds up automatically to maintain a safe distance from the vehicles ahead. The control is based on sensor information (Ultrasonic sensor). Concurrently, a buzzer goes off in an attempt to wake the driver. If the driver does not respond to the stimulation, a GSM module intervenes to contact the Emergency immediately through an SMS that holds the car's location obtained using a GPS module. That is all for the simple fatigue detection ADAS unit. It is integrated and implemented using an AVR (ATmega32) microcontroller that shall use a serial communication protocol (SPI) to communicate with the Raspberry Pi microcontroller. The hardware connection between both controllers is obtained using a level shifter, and that is a voltage level translator. It connects different integrated circuits that operate on different voltage levels e.g. 3V & 5V together.

On another scope, the system also provides blind spot assistance using an IMU (MPU6050) module and a servo motor. They adjust the side mirrors for a better sight of the road especially when entering tunnels or mounting bridges.

The following are the use cases of the provided system:

1) Fatigue Detection Using Deep Learning
2) Drowsy Driver Alert Signal
3) Driver Asleep: Emergency Notification
4) Adaptive Cruise Control. (Obstacle Detection).
5) Blind Spot Assistance.
6) Location Identification & Emergency Contact

The General Block Diagram of the proposed unit is shown below:

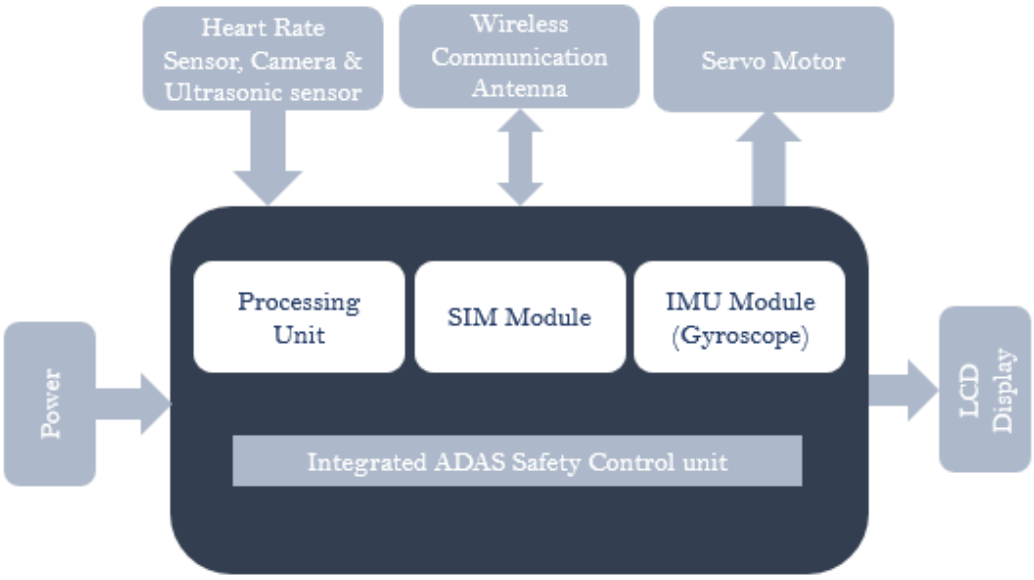


Figure 3 General Block Diagram Of the ADAS Unit

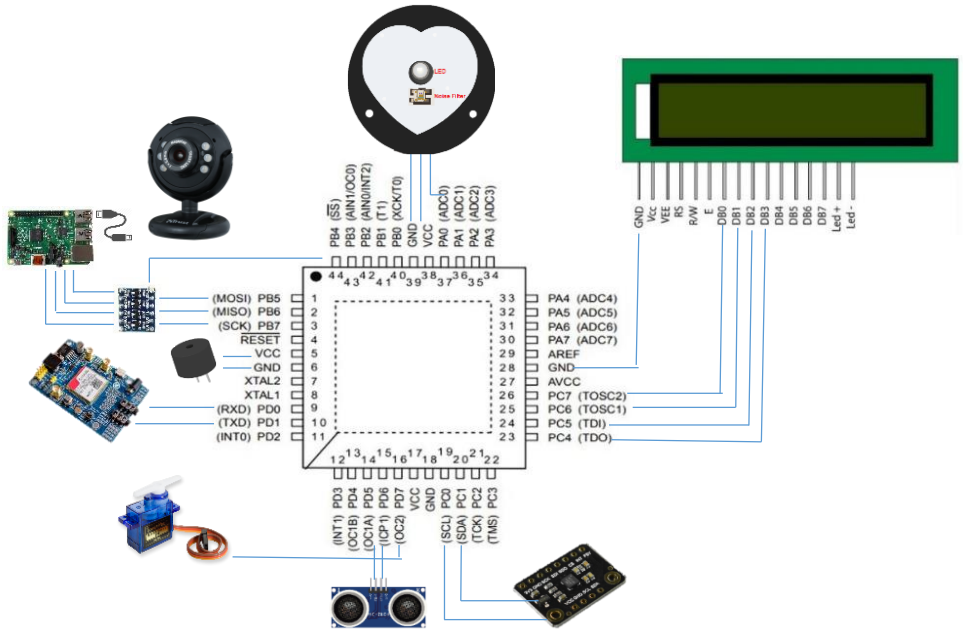


Figure 4 Pin Configuration & AVR interface

1.4Objective

Developing an accurate sensing apparatus, and an automatic emergency response system to be of use in the automotive industry using ADAS so as to reduce the amount of car accidents as much as possible thus reducing the amount of casualties amongst us and ensuring occupant and pedestrian safety. Furthermore, using existing components to build a simple structure that does not damage the environment and does not force the driver to carry the burden of wearing anything to ensure his safety and measure his vitals.

1.5Reassuring Effects of ADAS

According to the National Safety Council, USA, ADAS is showing promising results in limiting the numbers of road casualties. Some of these are shown below:

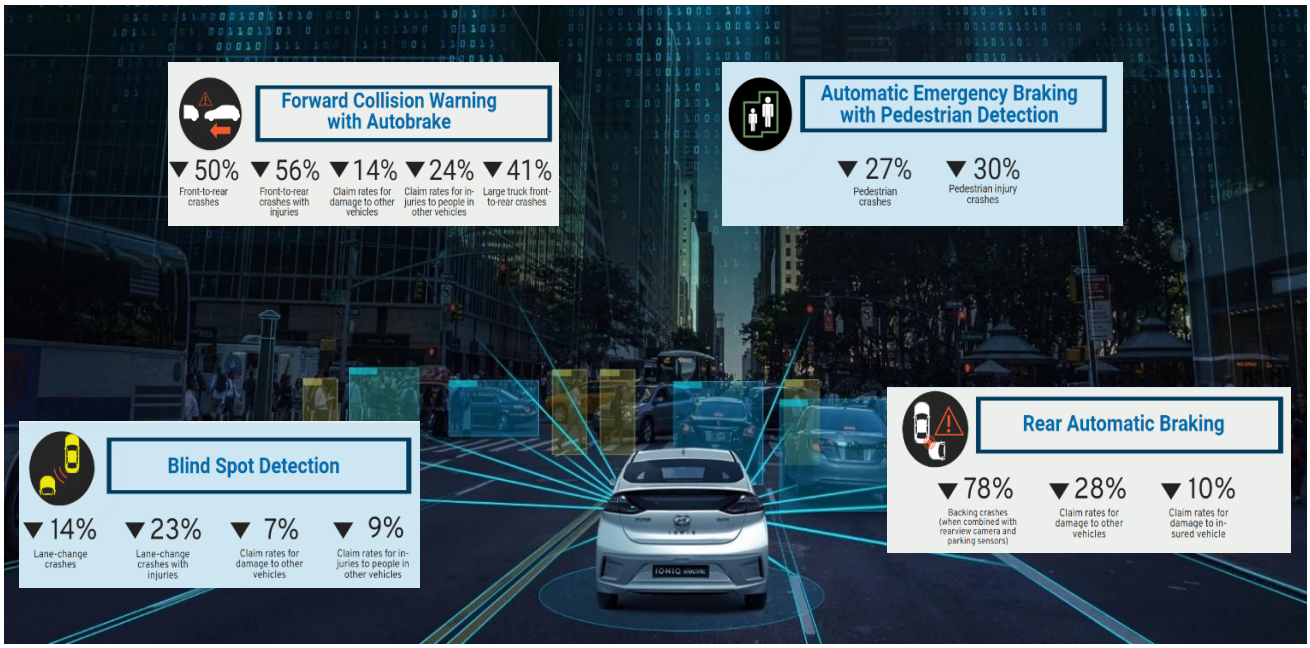


Figure 5 ADAS Results On Road

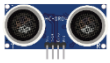
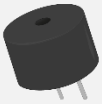
1.6 Target Audience

All auto components brands are target audience for such ADAS units. Some are

1) Valeo (France)
2) APTIV (Ireland)
3) DENSO (Japan)
4) SCHAEFLER (Germany)
5) NAPA (U.S.A)
6) FORVIA (France)
7) BorgWarner (U.S.A)
8) HYUNDAI MOBIS (S. Korea)
9) Toyota (Japan)
10) General Motors (U.S.A)
11) Volkswagen (Germany)

And many more advanced and well known firms

1.7 Components

Component	Specification	Price EGP	PNG
Raspberry Pi Camera Module	5 MP V2	525	
Raspberry Pi	4B-4GB	4600	
Pulse Sensor	PPG Sensor	190	
Sim808 GPS & GSM module	Location & SMS module	950	
Atmel AVR USB Development System	Atmega32	1250	
Ultrasonic Sensor	Distance Sensor	70	
MPU6050	IMU Module	280	
Servo motor	Actuator	75	
Buzzer	Actuator	5	

Chapter Two: Modern Advancements in Automotive Technology

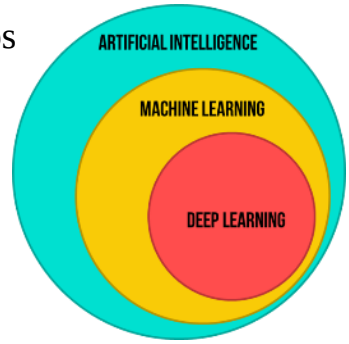
Today, it is difficult to classify and differentiate between different driver safety systems and approaches for their huge diversity. In accordance, this chapter discusses the different approaches for driver safety systems including both the artificial intelligence, and embedded systems fields.

2.1 Artificial Intelligence Based Technologies

The section that follows explores the fascinating field of artificial intelligence (AI) and its useful applications, with a focus on detecting driver fatigue. It examines the ideas behind AI and how it might transform the auto industry. The section also explores Convolutional Neural Networks' (CNNs') potent capabilities and how they can be used to identify driver fatigue. CNNs can analyse visual cues in real-time and identify important signs of drowsiness or fatigue by utilising advances in deep learning and computer vision. The section provides a thorough investigation of how AI, in particular CNNs, may improve traffic safety by reducing the dangers posed by driver fatigue.

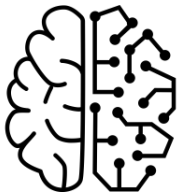
2.1.1Artificial Intelligence (AI)

Is a branch of technical science that researches and develops theories, procedures, technologies, and software for emulating and enhancing human intelligence. Artificial intelligence can be broadly defined as technology that can learn and produce intelligent behaviour.

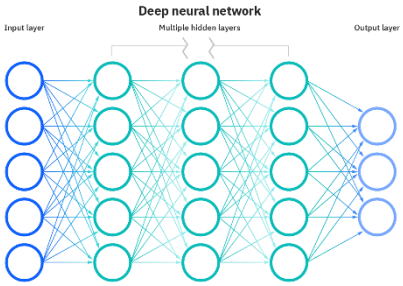


2.1.2Machine learning (ML)

Machine learning is a subfield of artificial intelligence and computer science that focuses on using data and algorithms to mimic how people learn while gradually improving its accuracy.



2.1.3Deep learning (DL)



Is a subfield of deep learning that deals with algorithms inspired by the structure and operation of the brain called artificial neural networks (ANNs).

2.1.4 Computer vision

Artificial intelligence (AI)'s field of computer vision aims to make it possible for computers to comprehend and interpret visual data from pictures and videos. It entails creating models and algorithms capable of identifying objects, recognising faces, detecting motion, and comprehending spatial relationships from visual data. Convolutional Neural Networks (CNN) are among the best tools for computer vision applications.

2.1.5 CNN for computer vision

Due to their extraordinary capacity to learn and extract features directly from raw pixel data, CNNs have become the preferred choice for computer vision tasks. They are extremely effective at analysing visual patterns and structures because they were created specifically to mimic the visual processing of the human brain. The hierarchical architecture of CNNs, which consists of several layers of interconnected neurons, is their primary strength.

2.1.6 Driver's Fatigue Detection using CNN:

In the field of automotive safety, the detection of driver fatigue is a crucial application that aims to stop accidents brought on by sleepy or fatigued drivers. Convolutional Neural Networks (CNNs) are essential in this field because they provide a practical method for identifying visual cues that indicate driver fatigue. CNNs can identify key signs of fatigue, such as drowsy eye movements, drooping head posture, or erratic driving behaviour, by analysing real-time video feeds

from in-vehicle cameras or other monitoring systems and utilising the power of deep learning and computer vision.

Due to their capacity to recognise and extract complex patterns and features from unprocessed image data, CNNs are highly effective at detecting driver fatigue.

CNNs can recognise subtle visual cues that indicate fatigue after extensive training on large datasets with diverse examples of alert and fatigued drivers. As a result of CNNs' hierarchical architecture, which enables them to capture both low-level and high-level features, they can recognise complex facial expressions, eye movements, and head poses that are indicative of drowsiness.

2.1.7CNN: convolutional neural network (CNN)

Is a feed forward neural network. Its artificial neurons might react to nearby units that are in the coverage range. CNN is excellent at processing images because it uses convolutional, pooling, and fully connected layers. In several scientific domains, particularly in the field of pattern categorization, CNN has now emerged as a research hotspot. The network is commonly utilised since it can directly input original photos instead of requiring complex image pre-processing.

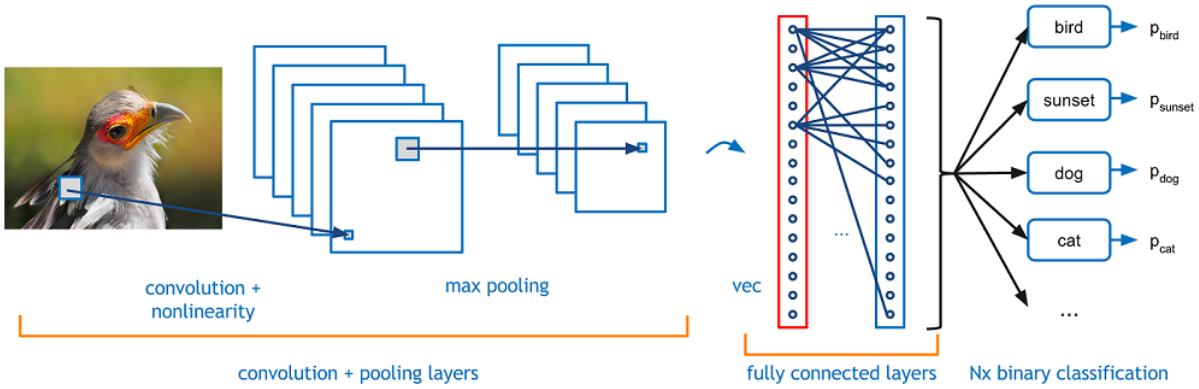


Figure 6 Convolutional Neural Network

Google's TensorFlow is an open-source machine learning framework. It includes a comprehensive set of tools and libraries for developing and deploying various machine learning models, including deep learning models. TensorFlow is intended to make it easier to build efficient and scalable neural networks, making

it a popular choice among artificial intelligence researchers and practitioners. TensorFlow allows you to create and customise CNN architectures by defining the network's layers, connections, and parameters. TensorFlow includes a variety of pre-implemented layers, such as convolutional layers, pooling layers, and fully connected layers, which are fundamental components of CNNs. TensorFlow also provides optimisation techniques, loss functions, and evaluation metrics that are extensively used in CNN training.

A CNN's multi-channel convolution architecture is composed of numerous single convolutions. The current layer's input is the result of the preceding layer, or the first layer's original image. It becomes the layer's output after being convolved with the filter in the layer. Each layer's convolution kernel serves as the learning weight. Convolution results should be biased and activated using activation functions, similar to FCN, before being input to the following layer.

In order to lower the amount of the input on the following layer through pooling, dimensions are reduced. Max pooling and average pooling are examples of common pooling. When max pooling is used, the largest value in a tiny square area is chosen as the representative of this region, however when average pooling is used, the mean value is chosen as the representative. This little space's side is the size of a pool window.

The fully connected layer is essentially a classifier. To output and categorise results, the features extracted from the convolutional and pooling layers are straightened and inserted into the fully connected layer.

2.1.8 Modern CNN:

For computer vision tasks, contemporary convolutional neural networks (CNNs) are the cutting edge of deep learning. They are sophisticated architectures that were created specifically to address the difficulties and complexities of visual data analysis. These CNNs have advanced from their forerunners by

incorporating a number of crucial innovations that have greatly enhanced their functionality and performance.

The creation of deeper architectures with numerous layers is one of the most significant developments in contemporary CNNs. Exceptional success has been shown by models like ResNet, Inception, and DenseNet in capturing complex patterns and representations. These architectures enable the effective flow of information through the network, mitigating the issue of vanishing gradients and promoting better learning and feature extraction. They do this by utilising skip connections, residual blocks, and dense connections.

Modern CNNs benefit from transfer learning and pre-training on massive datasets like ImageNet in addition to architectural improvements. CNNs can learn general knowledge and feature representations that can be honed for use on particular tasks or datasets by utilising pre-trained models. By drastically reducing the amount of labelled data required and speeding up the training process, this transfer learning approach makes CNNs more useful and effective in practical applications.

Additionally, contemporary CNNs use methods like regularisation, batch normalisation, and data augmentation. By applying transformations like rotations, translations, and flips to the current data, data augmentation involves creating extra training samples. Batch normalisation makes each layer's inputs normal, which enhances network stability and speeds convergence. CNNs can perform well on untried data thanks to regularisation techniques like dropout and weight decay, which reduce over fitting and improve generalisation abilities.

2.2 Embedded Systems Based Technologies

On the race to make cars perfectly safe, and most comfortable for humans to ride, there have been three substantial technical approaches. They all use sensors to sense the car's surroundings, and take action accordingly. This section aims to discuss each one of them separately before diving further into ADAS technology and the technical approach of this project.

2.2.1 Vehicle to Vehicle Communication (V2V)

V2V communication is the ability of cars to wirelessly exchange information about the speed, and position of surrounding vehicles. This technology allows vehicles to broadcast and receive omni-directional messages (up to 10 times per second), creating a 360-degree "awareness" of other vehicles in proximity. In other words vehicles are able to generate and receive alerting messages e.g. "Warning: Road work ahead in 500 m." or "Warning: Ambulance approaching." This means it uses sensors to feel the surrounding environment and provide accurate warning messages. Telematics units form a network around the car so that it can communicate with its surroundings i.e. Cars can talk to each other and to their surroundings. This technology has evolved & spread to include the surrounding infrastructures (V2I), and everything (Vehicle to Everything V2X).

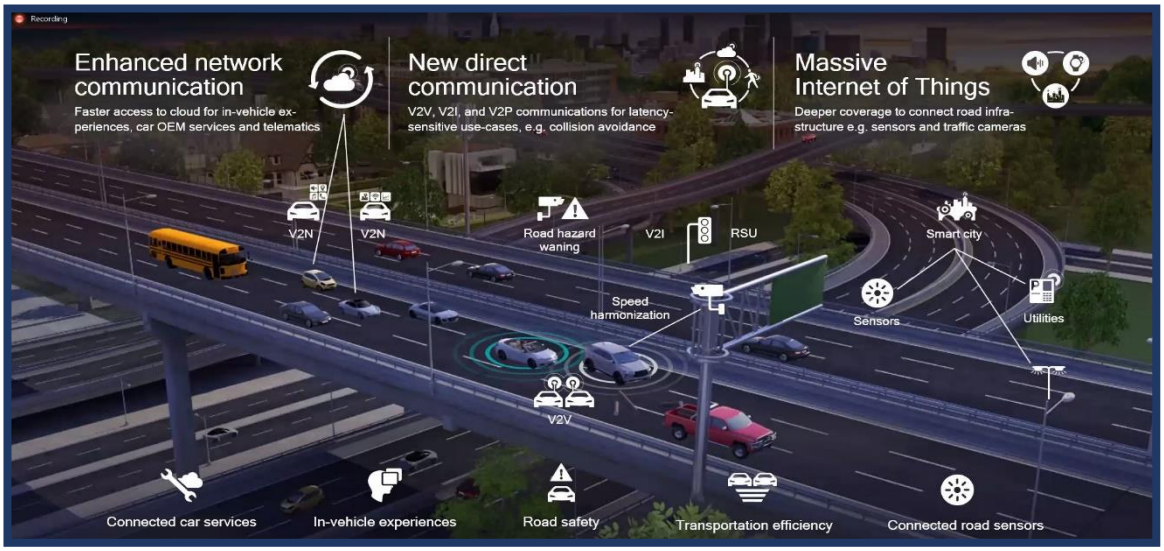


Figure 7 Vehicle to Vehicle Communication

2.2.2 Advanced Driver Assistance Systems (ADAS)

They are electronic systems integrated into cars. They utilise advanced technologies to assist the driver. They can also be referred to as “Safety Systems” ADAS-equipped vehicles can perceive the surrounding environment, and either provide information to the driver or take immediate action based on what they perceive. The technology is being applied today to cars, trucks, buses, farming, construction, and military vehicles. According to the National Highway Traffic Safety Administration in the U.S, 94% of accidents are due to human errors, or in other words, mistakes made by the driver. ADAS-equipped vehicles carry advanced sensors that augment the eyes, ears, and decision-making capabilities of a human driver. Are you capable of seeing in the dark ? Not very well, but RADAR can. Do you have a bat or a dolphin’s echolocation capability so as to detect if there’s a child behind the car before you put it in reverse ? Of course not, but SONAR sensors can. The same applies to seeing in all directions at once. You can rely on LIDAR, Ultrasonic sensors, and cameras for this one. Can you determine your Latitude & Longitude at all times ? No, either, but several constellations of global positioning satellites in space can send them to your car.

So, simply put, ADAS system architecture consists of a collection of sensors, interfaces, and a powerful processor that integrates all the data, and makes decisions in real-time conditions.

ADAS are passive and active safety systems. Passive ADAS system computers merely inform the driver of an unsafe condition, and consequently, the driver must be the one to take action in order to prevent this condition from resulting in an accident. Warning methods include: Sounds (Buzzer), flashlights (LEDs), or even physical feedback such as the vibration of the steering wheel to indicate that the lane the driver’s moving into is already occupied by another vehicle (Blind spot detection).

Familiar Passive ADAS functions include:

- **Lane Departure Warning:** Vehicle diverting from its lane.
- **Blind Spot Detection:** There is a vehicle in the driver's blind spot.
- **Forward Collision Warning:** Driver has to press the breaks immediately to avoid collision.
- **Parking Assistance:** Front or rear bumpers approaching an object while the car is parking.
- **ABS-Anti-lock Braking Systems:** Keeps the car from skidding on applying emergency breaks

Active ADAS systems are systems in which the vehicle takes direct actions on its own in critical conditions. Some of these functions are:

- **Automatic Emergency Braking:** Applying breaks automatically prior to any unwanted circumstances such as hitting pedestrians, animals or colliding with any car ahead.
- **Adaptive Cruise Control:** Adjusts car speed to match the vehicles ahead.
- **Lane keeping and centering:** Steering the car to stay centered in the lane.
- **Traffic Jam Assist:** Combines adaptive cruise control & lane keeping assist to provide semi-automated driver aid during dense traffic events like road construction for example.

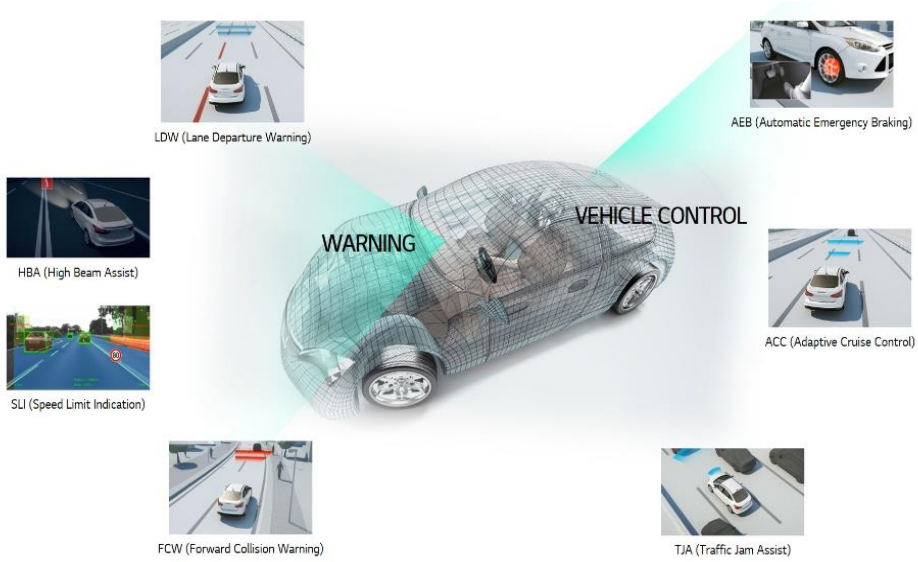


Figure 8 Some ADAS Technology Features

2.2.3 Autonomous Driving Systems

It is the ultimate extension of ADAS. On the 6-level vehicle autonomy scale, this occupies the sixth level in which the car does not need a driver, instead, it is fully capable of driving on its own. This can become real with the help of sensors, actuators, complex algorithms, machine learning, and powerful processors for software execution. Self-driving cars are under test in different countries in the world, they are still not available to the public. These complex systems will be of great use for the elderly, and the physically disabled individuals.

Figure 9 Jaguar Self-Driving Car

There are great challenges that face these systems. Some of these are:

- **Artificial VS Emotional Intelligence:** Human drivers often rely on cues and non verbal communication with pedestrians or reading the facial expressions of other drivers to make split-second judgement calls. It is hard to imagine autonomous cars that are able to replicate this connection or that have life-saving instincts such as those of human drivers.
- **Weather Conditions:** what would happen if there's a layer of snow on the road, and the lane dividers disappear ? What if there's rain ?...etc.

- **Law Regulations:** How would "Zombie Cars" driving with no passengers be tracked ? What if those cars were used in crimes ? What if they cross borders and state lines ?
- **Accident Liability:** Who is liable for accidents caused by autonomous cars ? manufacturers? or human passengers?
- **LIDAR & RADAR:** If multiple autonomous cars were to drive on the same road, would their LIDAR signals interfere with each other? If multiple radio frequencies are available, will the frequency range be enough to support mass production of autonomous cars?

Having mentioned the 6 levels of vehicle autonomy a lot, here are these levels:

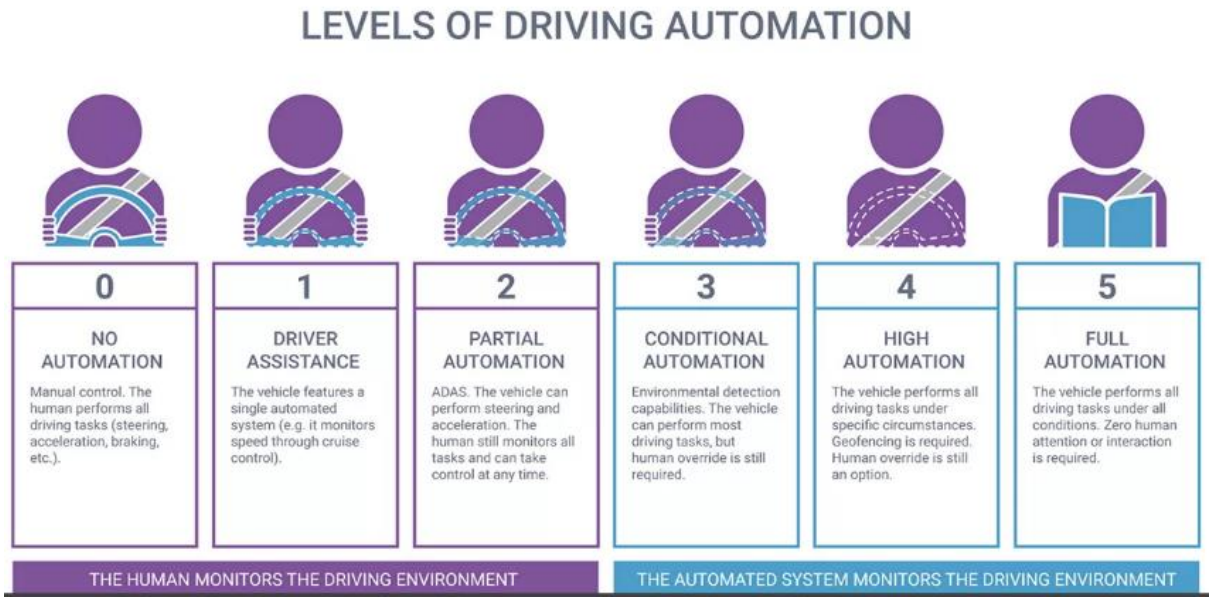
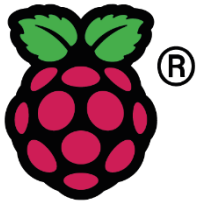


Figure 10 Vehicle Autonomy Levels

In spite of the fact that all three systems are powerful and provide revolutionary features, ADAS were preferred in this project. Because not only does the car take action, but it also alerts the driver himself, without the need to completely rely on the surrounding drivers or machines. In other words, the driver is still in control of his car whenever he is aware. The proposed system also alerts the surrounding

emergency units of extreme cases such as heart attacks or driver unresponsiveness. That is to make this system more rigid and immune towards risky situations.

2.2.4Raspberry Pi



Due to its adaptability and affordability, Raspberry Pi, a single-board computer the size of a credit card, has grown significantly in popularity in the artificial intelligence (AI) industry. The Raspberry Pi can make an excellent platform for a variety of AI applications thanks to its strong processing capabilities and GPIO (General Purpose Input/Output) pins. In addition to running deep learning algorithms and real-time AI tasks at the edge, it can be used for developing and deploying machine learning models. Because TensorFlow and PyTorch are compatible with the Raspberry Pi, programmers can use pre-trained models or create and train their own models.

Chapter 3: Fatigue detection proposed solution based on Artificial Intelligence

Artificial intelligence applications that detect signs of drowsiness and fatigue in people are required in order to prevent accidents, especially when people are driving. We will examine how OpenCV and deep learning can be used to implement a fatigue detection system in this technical overview.

The initial and crucial stage to detect driver’s fatigue level is to detect the driver’s face. The system can locate and recognise the driver's face among the input data by using computer vision techniques, such as face detection algorithms. This procedure makes it possible to analyse and extract key facial traits that indicate weariness, like eye aspect ratio and yawning.

3.1 Face Detection

Recognizing the driver's face is the first step in developing a trustworthy fatigue detection system because it serves as the foundation for effectively tracking and gauging their level of attentiveness.

"Haarcascade" is one of the widely used algorithms for facial recognition. The face detection system makes use of the Haar cascade classifier. It functions in four steps:

3.1.1 Stage one: Haar-feature selection

Dark and light sections make up a Haar-like characteristic. By comparing the difference between the sum of the intensities of the bright and dark zones, it yields a single value. is done to extract useful elements necessary for identifying an object.

3.1.2 Stage two: Integral Images creation

In an integral image, a given pixel is the total of all the pixels to its left and all the pixels to its right. The use of Integral Images drastically decreases the time required to complete this work because extracting Haar-like features requires calculating the difference between dark and light rectangular regions.

3.1.3 Stage three: AdaBoost Training:

The best features are chosen by this algorithm out of all features. It creates a "strong classifier" by combining multiple "weak classifiers" (best characteristics). The "strong classifier" that is formed is just a linear combination of all "weak classifiers."

3.1.4 Stage four: Cascade Classifier

It is a technique for combining ever-more complicated classifiers, like AdaBoost, that allows non-face input to be swiftly rejected while devoting more processing power to promising or positive face-like regions. It greatly shortens the computation time and improves the effectiveness of the procedure.

3.1.5 Code Implementation

- **Step one**: import essential libraries.

```
# Importing OpenCV package
import cv2
```

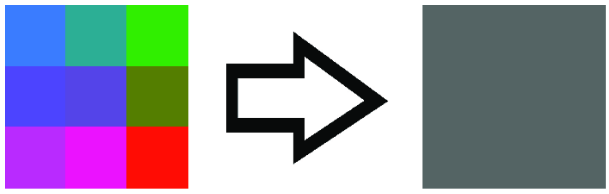
- **Step two**: Loading image:

  This step is basically loading the image path which will be passed to haar cascade classifier to detect face.

```
img = cv2.imread('E:/Jupyter_Workstation/pred/720.jpg)
```

- **Step three**: convert image to grayscale.

  The image is initially a three-layer image (RGB), so it's converted to a one-layer image (grayscale). As mentioned earlier that Haar-feature selection produces a single value by comparing the difference between the sum of the intensities of the bright and dark zones.



(a) RGB pixels. (b) Grayscale pixels.

```
convgrayimg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

- **Step four**: loading requires haar-cascade XML classifier file.

```
face_cascade= cv2.CascadeClassifier('E:/Jupyter_Workstation/haarcascade_frontalface_default.xml')
```

- **Step five**: applying face detection method on the grayscale image.

The Cascade Classifier Cv2::detectThis is done using the MultiScale method, which produces boundary rectangles for the four faces (x, y, w, and h) that were recognized. ScaleFactor and minNeighbors are two necessary parameters. The window size increases to the "maxSize" after checking all windows of that size and scaling up by the "scaleFactor." The factor used to calculate the increase in window size is called ScaleFactor. If the "scaleFactor" is large (for example, 2.0), there will be fewer steps and faster detection, but we run the risk of missing items

whose sizes lie between the two tested scales. (The default value for the scaling factor is 1.3). By employing "minNeighbors" values that are higher, false detection errors and false positives will be reduced.

- **Step six**: Iterating through rectangles of detected faces

This phase is mainly to draw a rectangle on person's face. By using the rectangle method of the cv2 module which iterates over all identified faces and draws rectangles around each one.

```
for (x,y,w,h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), thickness=2)
```

Sample input:

Sample output:

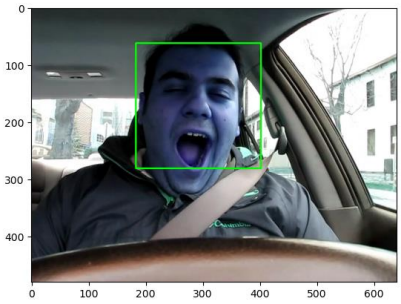


Figure 11 Focusing on the Face

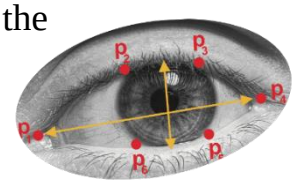
After implementing the steps mentioned above, this phase is successfully developed to take an input image and detect the faces in it by drawing a rectangle on the person's face.

3.2 Fatigue or drowsiness detection

After successfully implementing face detection phase. This following stage is to implement a fatigue detection solution based on artificial intelligence. There were two pathways. The first one is implementing the solution using OpenCV and the second one is using deep learning model.

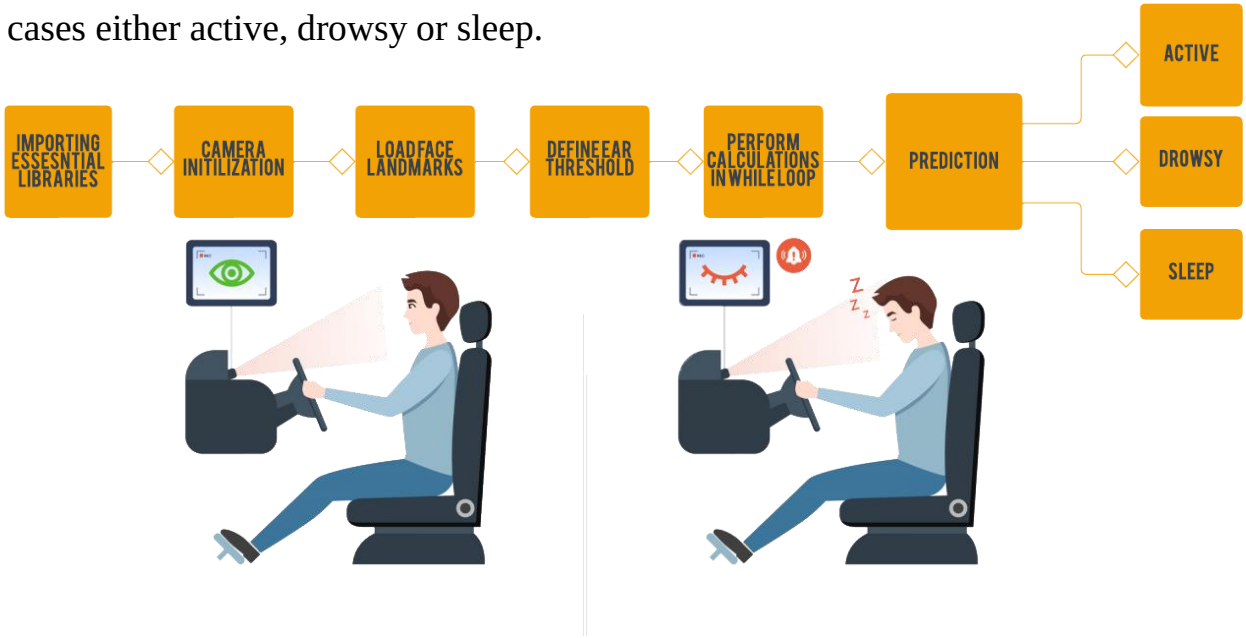
3.2.1 OpenCV

OpenCV is an open-source computer vision and machine learning software library. It uses a variety of techniques such as edge detection, segmentation, filtering, and machine learning algorithms. OpenCV is based on mathematical calculations using dlib to determine facial landmarks to detect driver's face and eye aspect ratio (EAR) computed by Euclidean distance between the eyes.



$$EAR = \frac{Sum\ of\ vertical\ distance}{2*horizontal\ distance\ of\ eye} = \frac{|p_2-p_6|+|p_3-p_5|}{2*|p_1-p_4|}$$

The workflow to utilize OpenCV to determine driver's current vigilance level is done by the following steps. First import libraries and camera initialization. Following by loading face landmarks and defining EAR threshold. Then, performing calculation in while loop to continuously check driver's current state by taking frames from camera's footage and then make calculation using EAR formula to detect driver's vigilance level. And finally making predictions of 3 cases either active, drowsy or sleep.



This method presented a lightweight model and achieved accuracy of more than 80%. Yet, OpenCV is based on computations only discarding human facial expressions.

3.2.2 Deep learning (CNN)

Another solution is suggested using Deep Learning algorithms in order to get around the aforementioned drawbacks of using OpenCV to implement the solution. The suggested remedy is based on CNN.

Artificial neural networks (ANNs), a class of algorithms inspired by the structure and function of the brain, are the focus of the deep learning (DL) subfield of deep learning.

Deep learning (CNN) Workflow

To implement the solution using deep learning algorithms. This took many trials to achieve the highest possible accuracy. The challenges that faced the way to implement the solution using deep learning algorithms is that the datasets available aren't broad enough and aren't accurate enough in every class.

As a result, models accuracy using these datasets never gave the accuracy needed. So, this problem was solved by collecting new dataset for 3 categories (active-drowsy-sleep).

Workflow of developing (CNN) model

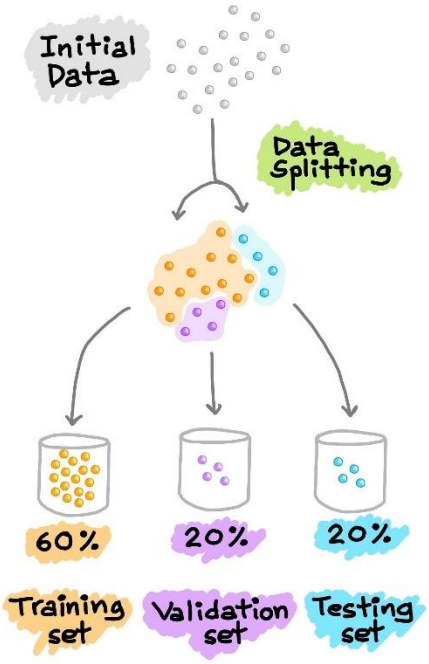
Phase 1: Importing essential libraries.

Importing crucial libraries plays a lead role at the beginning of creating a deep learning model because it lays the groundwork for utilizing a full range of potent tools and functions, including data pre-processing, model architecture design, and evaluation, enabling effective, and efficient model development.

Phase 2: Importing dataset and performing data pre-processing:

A fundamental stage in deep learning is importing a dataset and dividing it into train, test, and validation sets. Importing the dataset, which typically includes a selection of labelled or unlabelled instances, is the first step in the process. The dataset is then imported and split into three separate subsets: the training set, used

to train the model; the test set, used to assess the model's performance; and the validation set, used to adjust the model, and optimise its hyperparameters. This divide helps avoid overfitting and guarantees that the model generalises properly to new data. The train-test-validation split makes it easier to create effective models and helps in getting the best results possible when performing deep learning activities.



The set of techniques and operations performed on raw data before it is fed into a deep learning model for training or inference is referred to as data pre-processing in deep learning. It entails modifying and preparing the data so that it is ideal and appropriate for the learning process.

Enhancing the data's quality, consistency, and compatibility is the main goal of data pre-processing. Tasks like data cleaning, where missing values, outliers, or noisy data points are handled or eliminated, fall under this category. Another crucial step is data normalisation or scaling, which guarantees that the features of the data are on a comparable scale and stops some features from dominating the learning process due to their greater magnitude.

From the essential steps in data pre-processing is performing data augmentation.

Data Augmentation: The practise of artificially increasing a dataset by applying various transformations and adjustments to the current data is referred to as data augmentation. These transformations generate new training examples that are comparable to the original data but have minor differences. To improve model performance and generalisation, data augmentation is often employed in machine learning, particularly in computer vision tasks.

The purpose of data augmentation is to improve the diversity and unpredictability of the training data, hence making the model more resilient and capable of dealing with the various scenarios and variances found in real-world data. By introducing these variations, the model learns to generalise better and becomes less susceptible to changes in characteristics such as lighting, views, or other factors that may impact the presentation of the data.

Data augmentation is performed using image data generator

ImageDataGenerator is created with the following parameters:

rescale: Rescales the image's pixel values by dividing them by 255 and ensuring they are in the range [0, 1].

Shear_Range: Random shearing transformations are applied to the photos, which can assist increase the model's robustness to changes in perspective.

Zoom_Range: Randomly applies zooming adjustments to photos, which can help capture diverse scales and angles.

Horizontal_Flip: Flips photos horizontally at random, which can supplement the dataset and bring new variations.

Flow_From_Directory: builds an iterator (training_set) that loads and augments photos from the supplied directory in number of batches. In example, It also resizes the photos in size to match the input model size.

Phase 3: Building model:

The first step creates a sequential model object, which allows us to stack layers one after another.

The second step is conv2D() layer. It features number of filters, a nxn kernel, and employs the activation function. The shape of the input images is defined by the input_shape argument, which is in example is [224, 224, 3] for images with height, width, and three colour channels (RGB).

The pooling layer, a common element in convolutional neural networks (CNNs) used for down-sampling feature maps, is involved in the process' next stage. Pooling layers helps to extract crucial information while lowering computational complexity by reducing the spatial dimensions of the feature maps. The MaxPooling2D is one form of pooling layer that divides the feature maps into [nxn] non-overlapping regions and chooses the maximum value inside each zone. Other pooling layer types exist, such as AveragePooling2D, which determines the average value for each region instead.

The exact task at hand and the data's properties determines the pooling method to use. MaxPooling2D is frequently used for jobs like picture classification, where it's crucial to recognise the standout features.

The above steps are repeated to aid in the extraction of higher-level features from the previously down-sampled feature maps.

Flatten layer is added to convert the multi-dimensional feature maps into a 1D vector, preparing it as input for the subsequent fully connected layers.

Dense layer is added to add a fully connected layer with number of units and activation function to capture higher-level representations by performing calculations on the flattened feature vector.

In case of binary classification: The final layer is implemented using a single unit as the output prediction is one of the two classes and sigmoid activation function to reduce the output to a number between 0 and 1, which represents the likelihood of the positive class. On the other hand while implementing categorical classification, number of neurons in output layer equals to number of classes in the case of categorical classification using Softmax activation function.

Phase 4: Transfer learning:

Transfer learning is the process of using features discovered while solving one problem to solve another that is somewhat related.

Fine-tuning is the final, optional step. It entails unfreezing the entire model you obtained earlier (or a portion of it) and retraining it using the new data at a very slow learning rate. By gradually adjusting the pretrained features to the new data, this has the potential to produce significant improvements.

Transfer learning isn't mandatory in building a CNN model in general. On the other hand, transfer learning is utilized in the f**atigue detection proposed solution**.

Phase 5: Model compiling and Early Stopping

Due to the callback's monitoring of the validation set, the validation set is also passed. The number of epochs in this scenario is set, but the callback will terminate the training process if the loss does not improve after two epochs, as stated in the EarlyStopping callback.

Phase 6: Model training:

model training is implemented using model.fit()

The input data for the model's training are specified by the first parameter.

The validation data to assess the model's performance during training are specified by the second parameter.

The number of times the entire training dataset will be run through the model during training is determined by the epochs parameter.

Any callback functions to be run during training are specified using the callbacks parameter.

Whether or not the training data should be shuffled at the start of each epoch is determined by the shuffle parameter. It helps prevent any potential bias brought on by the order of the data by ensuring that the training samples are presented to the model in a random order during each epoch when it is set to True.

The fit() function returns a History object associated with the variable history after the model has finished training. This object includes details about the training procedure, such as the metrics' values for loss and accuracy at each epoch.

Phase 7: Model evaluation using test data

After model training, model evaluation is done using test data which was not seen by model during training process. This evaluation method ensures that the model's performance is good along with new data.

3.3 Deep Learning Model Implementation

In order to implement the solution based on deep learning, this took several trials with different model architectures and methods.

3.3.1 Binary Classification

The first trial is by making a fully customized model to learn from one of the datasets available online to make a binary classification either active or fatigue. This method achieved validation accuracy of nearly 87%. The con of this trial is that the model accuracy isn't high. Also, models learn from insignificant features as background.

The first trial is implemented using a ready dataset that consists of two classes of active and fatigue subjects.

- Model implementation

Here the model is a fully customized one with an output layer consisting of one neuron due to binary classification and activation function sigmoid is utilized.

```
cnn = tf.keras.models.Sequential()
```

```
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu' , input_shape=[224, 224, 3]))
```

```
#Pool size 2x2 strides=2 l huwa l na2la maben l selection b 2 coloumns
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

```
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

```
cnn.add(tf.keras.layers.Flatten())
```

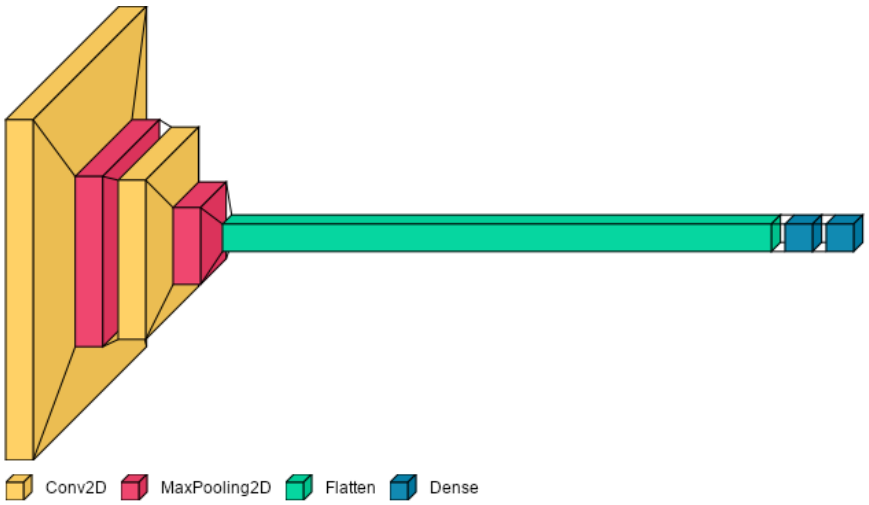
```
cnn.add(tf.keras.layers.Dense(units = 64, activation='relu'))
```

```
cnn.add(tf.keras.layers.Dense(units = 1, activation='sigmoid'))
```

```
cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy' , metrics=['accuracy'])
```

Also, here binary_crossentropy loss function is used as for the aim of model binary classification.

- Model structure:



- Model performance

Model achieved validation accuracy of 86.5 %

3.3.2Categorical Classification

In a trial to enhance model's performance, another dataset was used which is a categorical dataset of three classes and ResNet50v2 model architecture used.

- Model implementation

```
demo_resnet2_model= tf.keras.models.Sequential()
pretrained_model= tf.keras.applications.resnet_v2.ResNet50V2(
    include_top=False,
    weights='imagenet',
    input_tensor=None,
    input_shape= (224,224,3),
    pooling=max,
)
for layer in pretrained_model.layers:
    layer.trainable = False
```

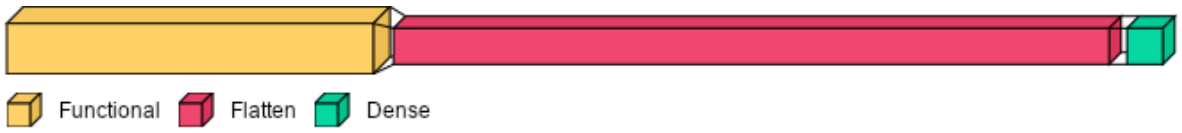
```
demo_resnet2_model.add(pretrained_model)
demo_resnet2_model.add(tf.keras.layers.Flatten())
demo_resnet2_model.add(tf.keras.layers.Dense(3, activation='softmax'))
```

- Model training.

```
history= model.fit(x = training_set, validation_data = val_set, epochs =5, callbacks=[early_stop])
```

```
Epoch 1/5
98/98 [==============================] - 250s 3s/step - loss: 0.5920 - accuracy: 0.7807 - val_loss: 0.2456 - val_accuracy: 0.92
80
Epoch 2/5
98/98 [==============================] - 235s 2s/step - loss: 0.1709 - accuracy: 0.9456 - val_loss: 0.1160 - val_accuracy: 0.96
66
Epoch 3/5
98/98 [==============================] - 237s 2s/step - loss: 0.0654 - accuracy: 0.9830 - val_loss: 0.0969 - val_accuracy: 0.97
17
Epoch 4/5
98/98 [==============================] - 189s 2s/step - loss: 0.0496 - accuracy: 0.9853 - val_loss: 0.0639 - val_accuracy: 0.98
97
Epoch 5/5
98/98 [==============================] - 155s 2s/step - loss: 0.0389 - accuracy: 0.9910 - val_loss: 0.0405 - val_accuracy: 0.98
97
```

- Model structure.



- Model results

  Model achieved validation accuracy =98.9 %. Although the model achieved an astonishingly high accuracy on the training data, it showed signs of overfitting, indicating that it had difficulty generalising well to new data.

  Another trials were conducted using other architectures (VGG-16, mobilenetv2, mobilenetv3small)

3.3.3 VGG-16

Another trial to enhance model's performance is done using VGG-16 model architecture.

- Model implementation

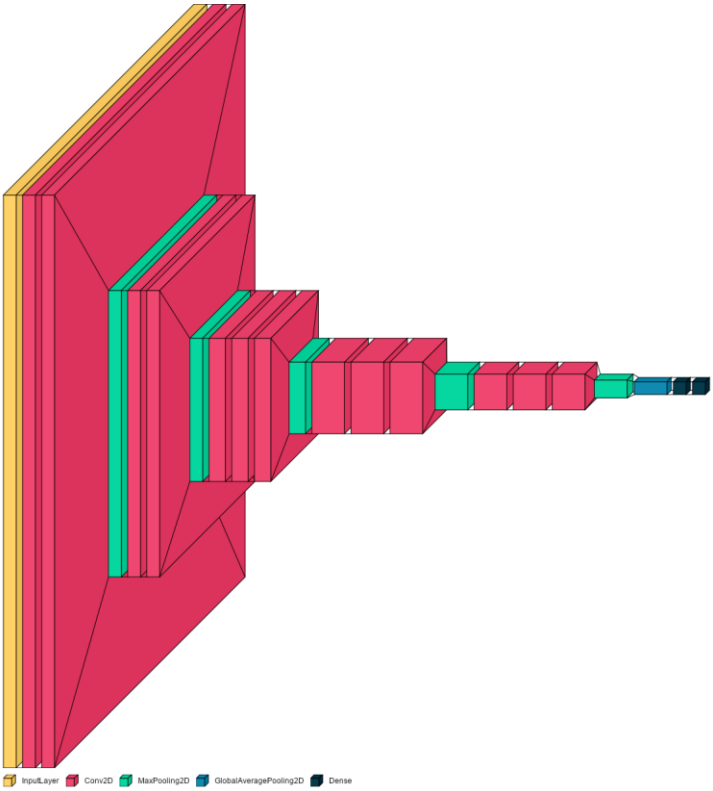
```
pretrained_model=tf.keras.applications.VGG16(
    include_top=False,
    weights="imagenet",
    input_shape=(224,224,3),
    pooling=max,
)
```

```
for layer in pretrained_model.layers:
    layer.trainable = False
```

```
x= pretrained_model.output
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dense(64, activation='relu')(x)
predictions = tf.keras.layers.Dense(3, activation='softmax')(x)
```

```
model = tf.keras.models.Model(inputs=pretrained_model.input, outputs=predictions)
```

- Model structure



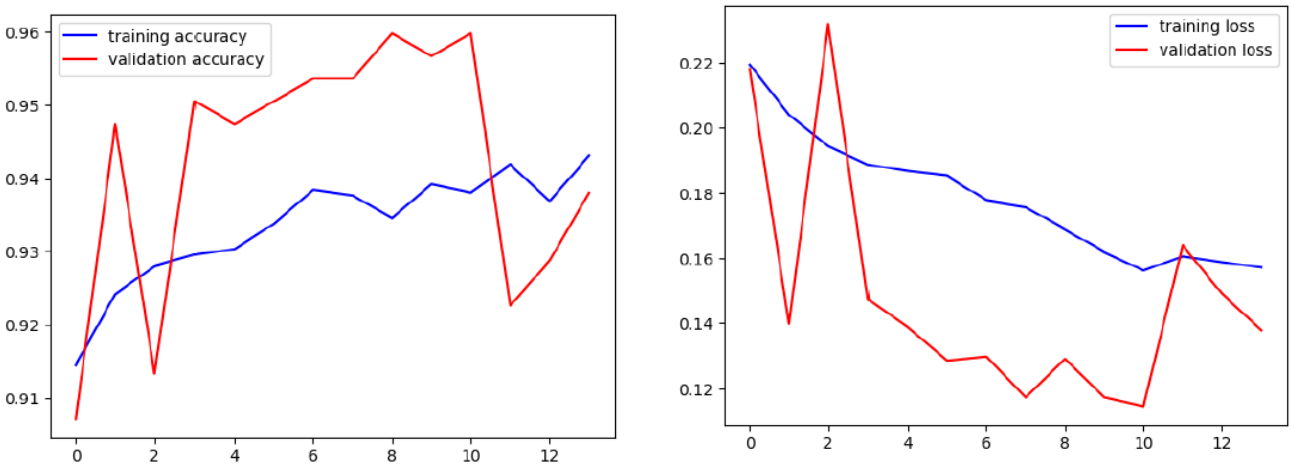
- Model results

Upon measuring the model's performance on validation data. Validation accuracy equals 93.8%. While on test data test accuracy = 94.4%

Training and validation accuracy:

```
Epoch 12/20
162/162 [==============================] - 363s 2s/step - loss: 0.1606 - accu
racy: 0.9420 - val_loss: 0.1639 - val_accuracy: 0.9226
Epoch 13/20
162/162 [==============================] - 424s 3s/step - loss: 0.1588 - accu
racy: 0.9369 - val_loss: 0.1491 - val_accuracy: 0.9288
Epoch 14/20
162/162 [==============================] - 424s 3s/step - loss: 0.1571 - accu
racy: 0.9431 - val_loss: 0.1379 - val_accuracy: 0.9381
Epoch 14: early stopping
```

Training and validation accuracy and loss plots:



Test Accuracy:

```
]: test_loss, test_acc = model.evaluate(test_set, steps=test_set.samples // 32)
   print('Test accuracy:', test_acc)

   10/10 [==============================] - 22s 2s/step - loss: 0.1791 - accurac
   y: 0.9438
   Test accuracy: 0.9437500238418579
```

3.3.4 Mobilenetv2

After the several trials to outcome a generative AI solution for fatigue detection

The outcome of previous trials wasn't generalized or practical enough for the purpose of driver fatigue detection. As a result, another trial was implemented using mobilnetv2 architecture.

- Model implementation

  Below is the implementation of model structure using MobileNetv2 architecture

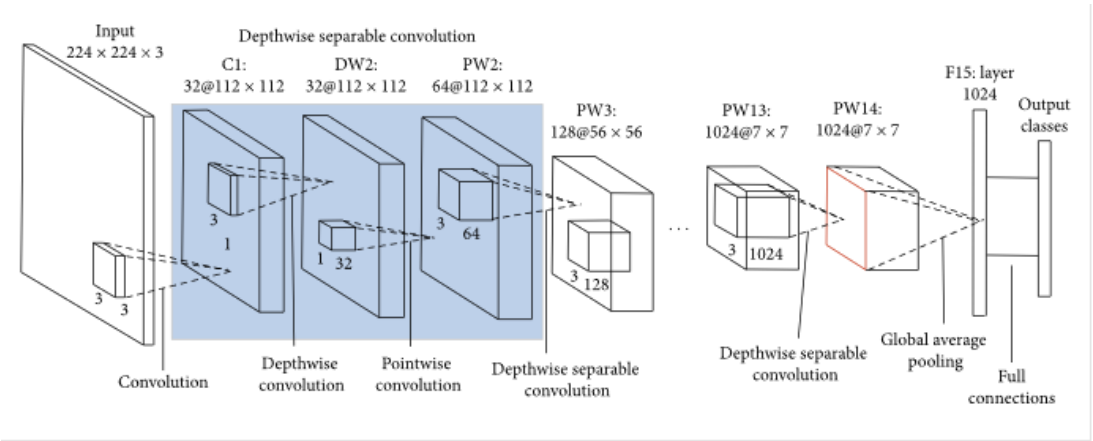
```
pretrained_model=tf.keras.applications.MobileNetV2(
    input_shape=(224,224,3),
    include_top=False,
    weights="imagenet",
    classes=3,
    pooling=max,

)
for layer in pretrained_model.layers:
    layer.trainable = False
```

```
x= pretrained_model.output
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dense(64, activation='relu')(x)
predictions = tf.keras.layers.Dense(3, activation='softmax')(x)
```

```
model = tf.keras.models.Model(inputs=pretrained_model.input, outputs=predictions)
```

- ## Model structure

  In the below figure, model layers of MobileNetv2 architecture is illustrated:



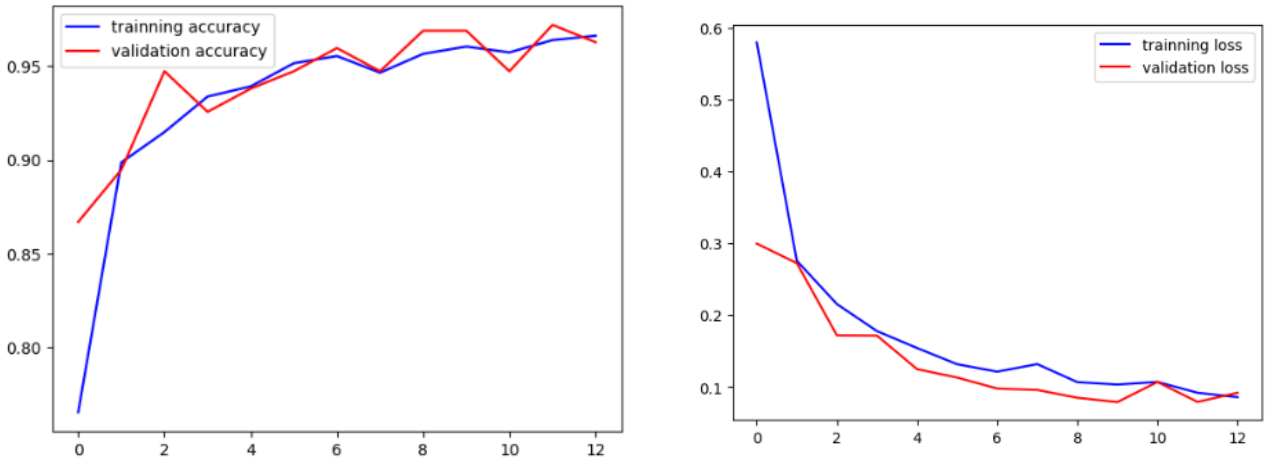
- ## Model results

Model achieved validation accuracy equals 96.3 % with early stopping operating and stopping training process at epoch 13 to prevent overfitting.

```
history= model.fit(x = training_set, validation_data = val_set, epochs =50, callbacks=[early_stop], shuffle=True)
```

```
Epoch 8/50
162/162 [==============================] - 101s 622ms/step - loss: 0.1315 - accuracy: 0.9466 - val_loss: 0.0955 - val_accurac
y: 0.9474
Epoch 9/50
162/162 [==============================] - 69s 423ms/step - loss: 0.1064 - accuracy: 0.9567 - val_loss: 0.0846 - val_accurac
y: 0.9690
Epoch 10/50
162/162 [==============================] - 70s 431ms/step - loss: 0.1030 - accuracy: 0.9605 - val_loss: 0.0786 - val_accurac
y: 0.9690
Epoch 11/50
162/162 [==============================] - 84s 515ms/step - loss: 0.1067 - accuracy: 0.9574 - val_loss: 0.1066 - val_accurac
y: 0.9474
Epoch 12/50
162/162 [==============================] - 69s 429ms/step - loss: 0.0916 - accuracy: 0.9640 - val_loss: 0.0789 - val_accurac
y: 0.9721
Epoch 13/50
162/162 [==============================] - 72s 441ms/step - loss: 0.0855 - accuracy: 0.9663 - val_loss: 0.0916 - val_accurac
y: 0.9628
Epoch 13: early stopping
```

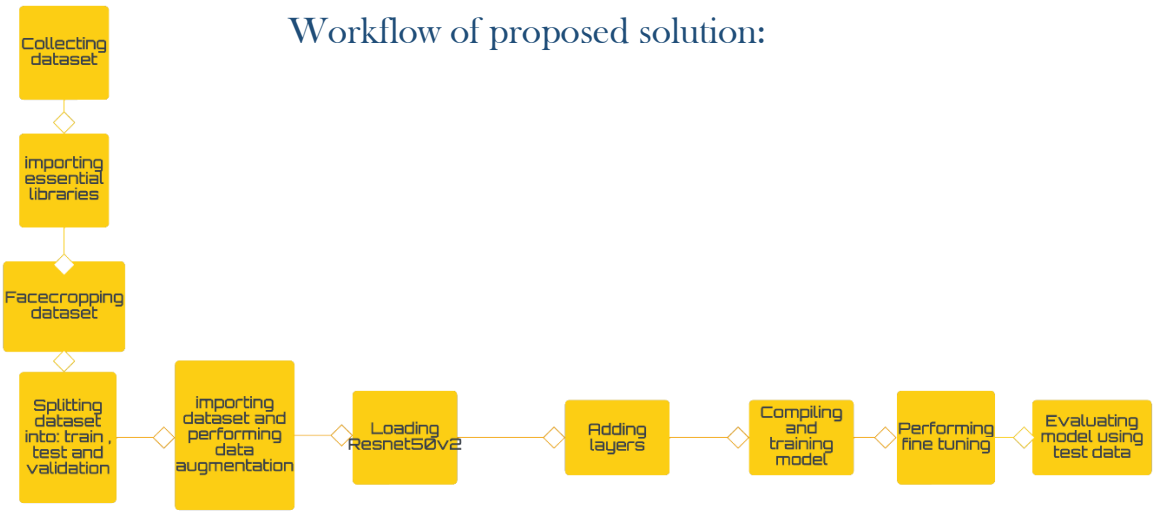
Training and validation accuracy and loss plots



As mentioned above this model had very good performance. But another last trial was made to enhance results with optimizing dataset pre-processing and performing fine-tuning.

3.3.5 Proposed solution (ResNet50v2 with face-cropping dataset and fine-tuning)

Workflow of proposed solution:



Based on the previous models' performance, ResNet50v2 architecture was chosen to be used for another trial , but with face cropping dataset to avoid the model learning from insignificant features and fine-tuning.

Fine-Tuning is done by unfreezing a few of the top layers of a frozen model base and train both the newly added classifier layers and the base model's final layers at the same time. This enables us to "fine-tune" the higher-order feature

representations in the underlying model to make them more relevant for the particular job.

- Importing essential libraries.

```
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from keras import regularizers
from keras.layers.core import Dropout
```

Here dropout is imported to be used in model's architecture to help prevent overfitting. As the Dropout layer randomly sets input units to 0 at a frequency of rate at each step throughout training period.

- Importing dataset and performing data augmentation

```
train_datagen=ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)
training_set = train_datagen.flow_from_directory(
    'E:/Jupyter_Workstation/newwds_v2/train',
    target_size=(224,224),
    batch_size=16,
    class_mode='categorical')
```

```
Found 2585 images belonging to 3 classes.
```

```
test_datagen= ImageDataGenerator(rescale=1./255)
test_set = test_datagen.flow_from_directory(
    'E:/Jupyter_Workstation/newwds_v2/test',
    target_size=(224,224),
    batch_size=16,
    class_mode='categorical')
```

```
Found 325 images belonging to 3 classes.
```

```
val_datagen = ImageDataGenerator(rescale=1./255)
val_set = val_datagen.flow_from_directory(
    'E:/Jupyter_Workstation/newwds_v2/val',
    target_size=(224,224),
    batch_size=16,
    class_mode='categorical'
)
```

```
Found 323 images belonging to 3 classes.
```

- Building model

```
model= tf.keras.models.Sequential()
pretrained_model= tf.keras.applications.resnet_v2.ResNet50V2(
    include_top=False,
    weights='imagenet',
    input_tensor=None,
    input_shape= (224,224,3),
    pooling=max,
)
for layer in pretrained_model.layers:
    layer.trainable = False
```

```
model.add(pretrained_model)
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(3, activation='softmax'))
```

- Model Summary

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50v2 (Functional)     (None, 7, 7, 2048)        23564800

 dense_2 (Dense)             (None, 7, 7, 64)          131136

 dense_3 (Dense)             (None, 7, 7, 3)           195

=================================================================
Total params: 23,696,131
Trainable params: 131,331
Non-trainable params: 23,564,800
_________________________________________________________________
```

- Model compiling

```
model.compile(optimizer= 'adam', loss= 'categorical_crossentropy' , metrics=['accuracy'])
```

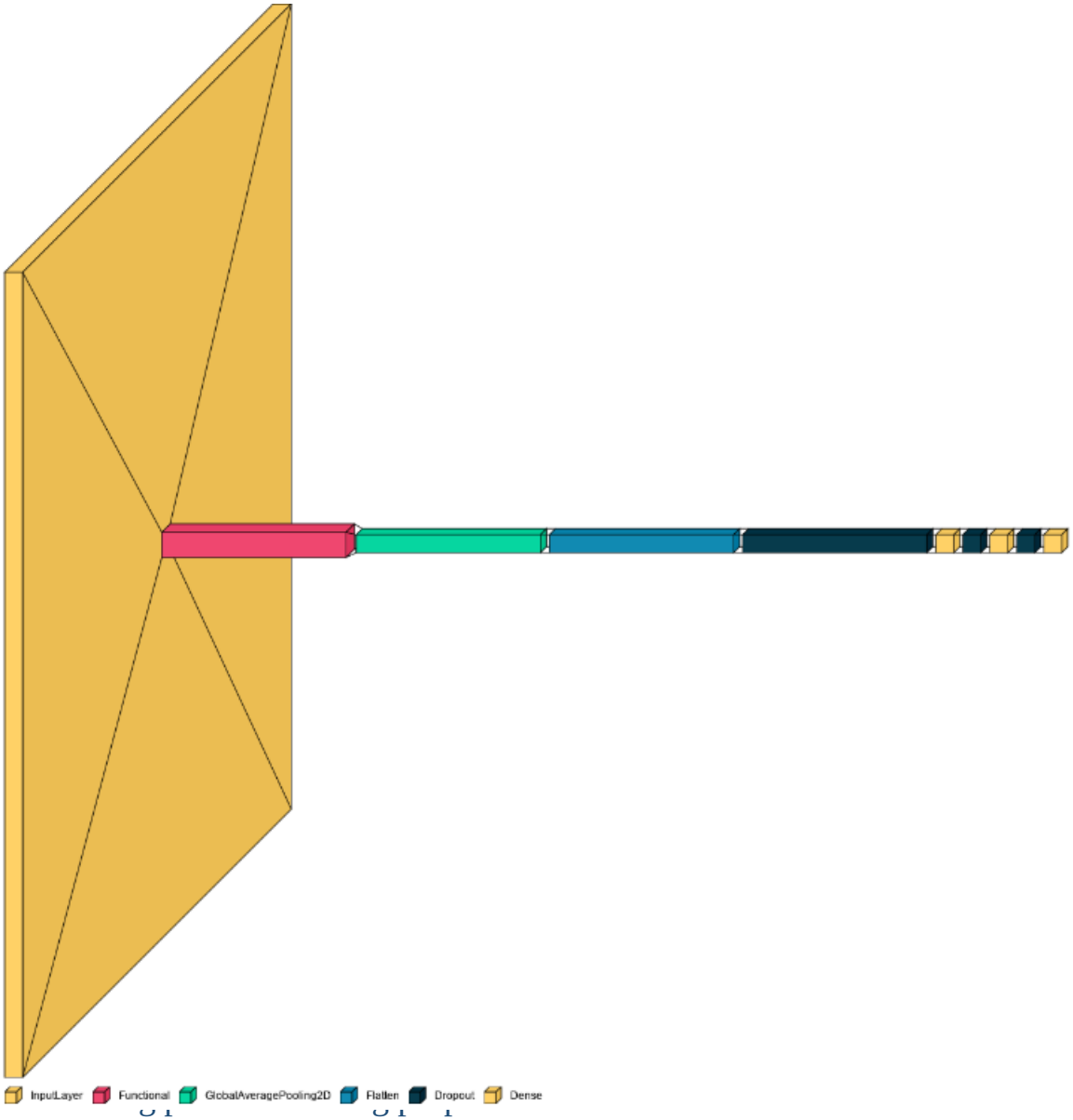
```
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, mode='min', verbose=1)
```

- Model fitting

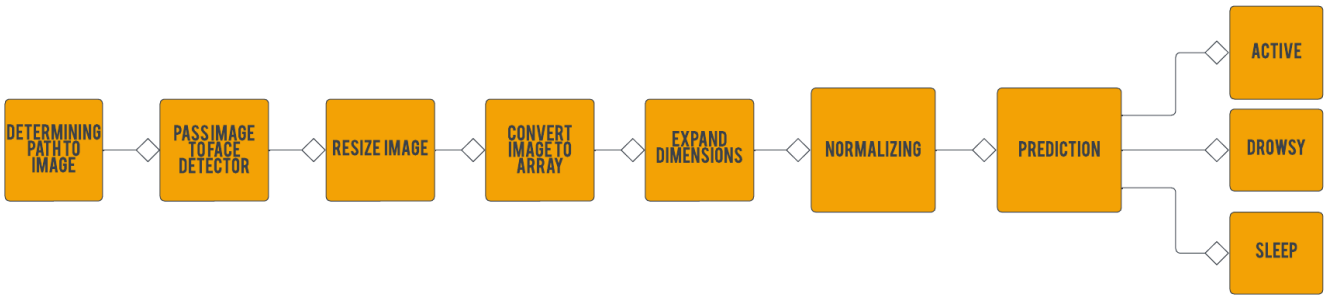
```
history= model.fit(x = training_set, validation_data = val_set, epochs =20, callbacks=[early_stop], shuffle= True)
```

```
Epoch 7/20
162/162 [==============================] - 123s 761ms/step - loss: 0.1347 - accuracy: 0.9497 - val_loss: 0.0760 - val_accura
cy: 0.9783
Epoch 8/20
162/162 [==============================] - 122s 751ms/step - loss: 0.1236 - accuracy: 0.9509 - val_loss: 0.0982 - val_accura
cy: 0.9628
Epoch 9/20
162/162 [==============================] - 120s 742ms/step - loss: 0.1073 - accuracy: 0.9605 - val_loss: 0.0659 - val_accura
cy: 0.9783
Epoch 10/20
162/162 [==============================] - 118s 728ms/step - loss: 0.0917 - accuracy: 0.9625 - val_loss: 0.1104 - val_accura
cy: 0.9474
Epoch 11/20
162/162 [==============================] - 121s 744ms/step - loss: 0.0913 - accuracy: 0.9660 - val_loss: 0.1314 - val_accura
cy: 0.9412
Epoch 12/20
162/162 [==============================] - 127s 785ms/step - loss: 0.1016 - accuracy: 0.9605 - val_loss: 0.0999 - val_accura
cy: 0.9659
Epoch 12: early stopping
```

- Model Structure



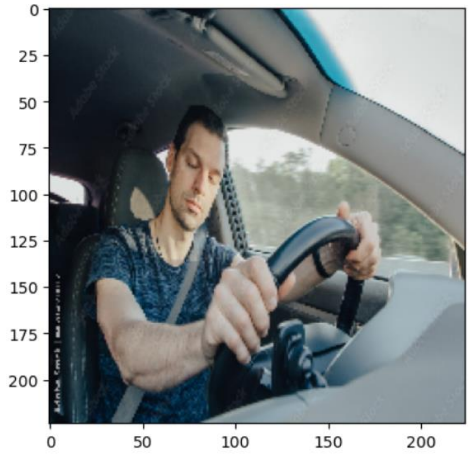
The figure below is how to make single predictions to test the proposed model.



```
test_image = image.load_img('E:/Jupyter_Workstation/pred/xs.jpg', target_size = (224,224))

# Display the image using Matplotlib
plt.imshow(test_image)

# Show the plot
plt.show()

#convert test image to array
test_image = image.img_to_array(test_image)
# expand image as the training was trained by batch containing 32 images not only one
test_image = np.expand_dims(test_image, axis = 0)
test_image= test_image/255.
result = model.predict(test_image)
# Get the predicted class
predicted_class = np.argmax(result)
if predicted_class==0:
    print("Active")
elif predicted_class==1:
    print("Drowsy")
elif predicted_class==2:
    print("Sleep")


# Print the predicted class
print(predicted_class)
```

Input:

Output:




```
1/1 [==============================] - 0s 92ms/step
Drowsy
1
```

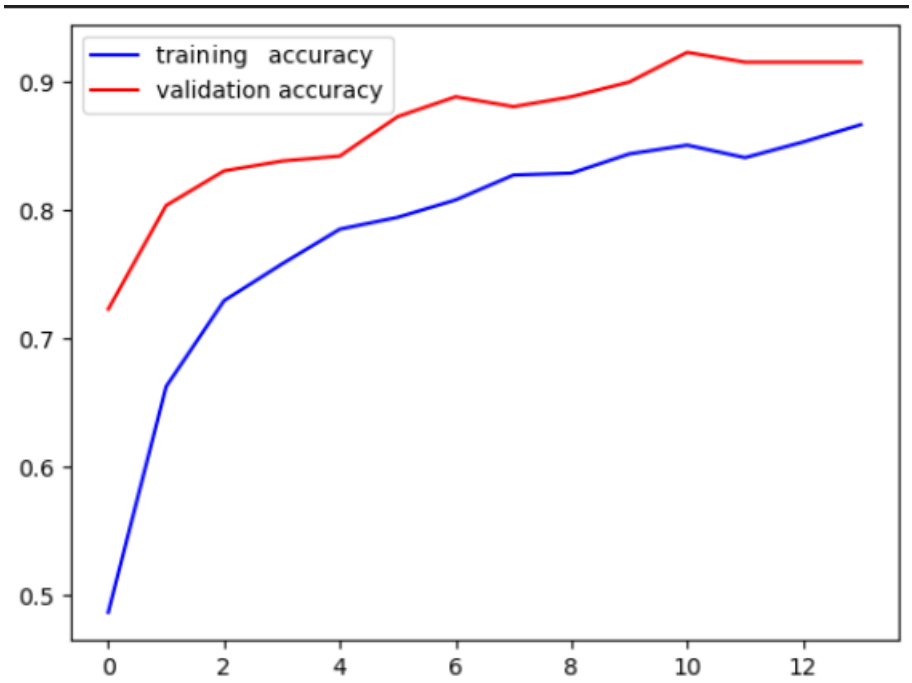
- Model performance on test dataset

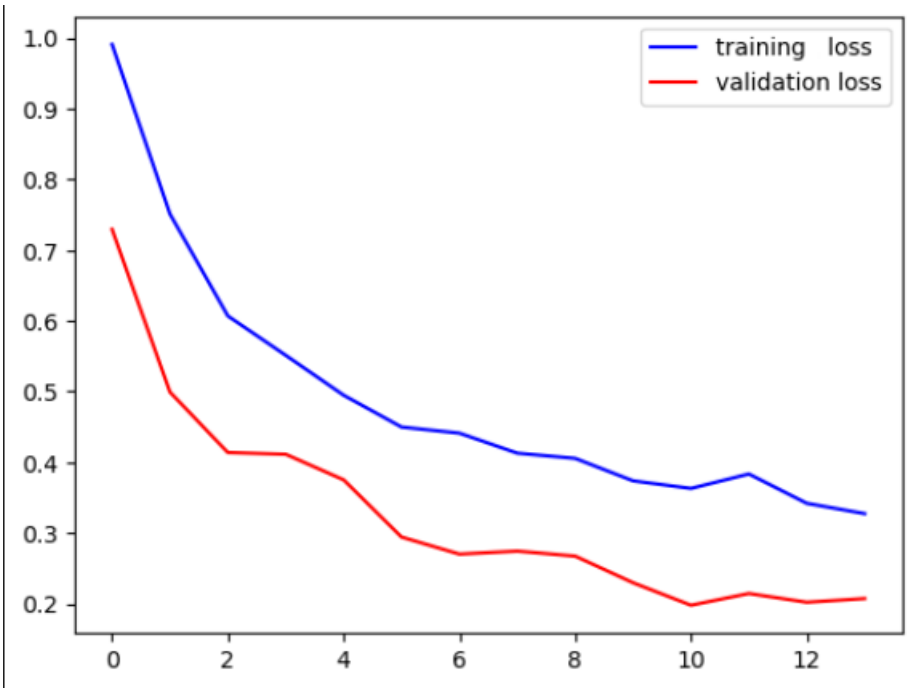
```
test_loss, test_acc = model.evaluate(test_set, steps=test_set.samples // 32)
print('Test accuracy:', test_acc)
```

```
10/10 [==============================] - 6s 577ms/step - loss: 0.1290 - accuracy: 0.9563
Test accuracy: 0.956250011920929
```

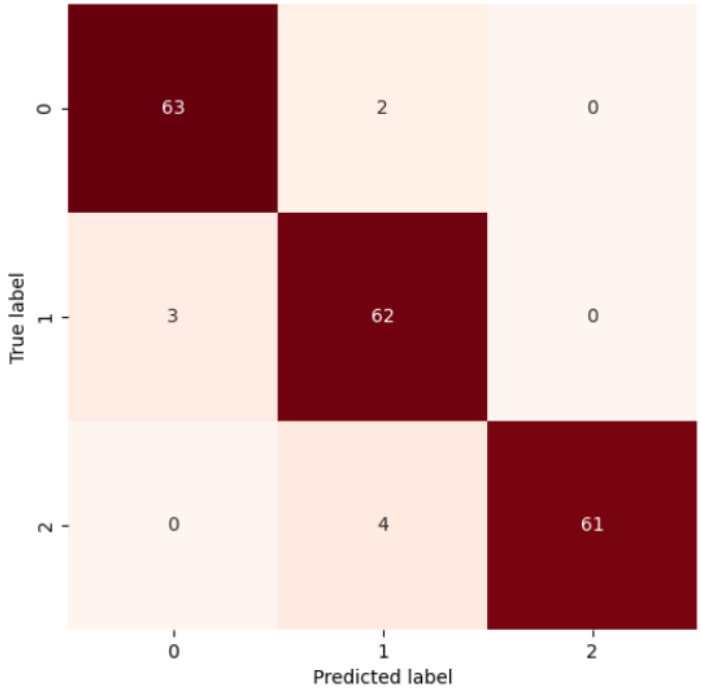
- Evaluating model performance:

After building and compiling the model, model training is performed. In order to test model's accuracy, plotting training accuracy / loss vs validation accuracy/loss is a great reflection to visualize the model's effectiveness.





To visualize testing accuracy of the model, plotting the confusion matrix is the most effective way. The below figure is the confusion matrix.



By observing the above figure, Confusion matrix reflects high model accuracy in prediction as the dark red colour is the correctly predicted label

3.3.6 Comparison between models:

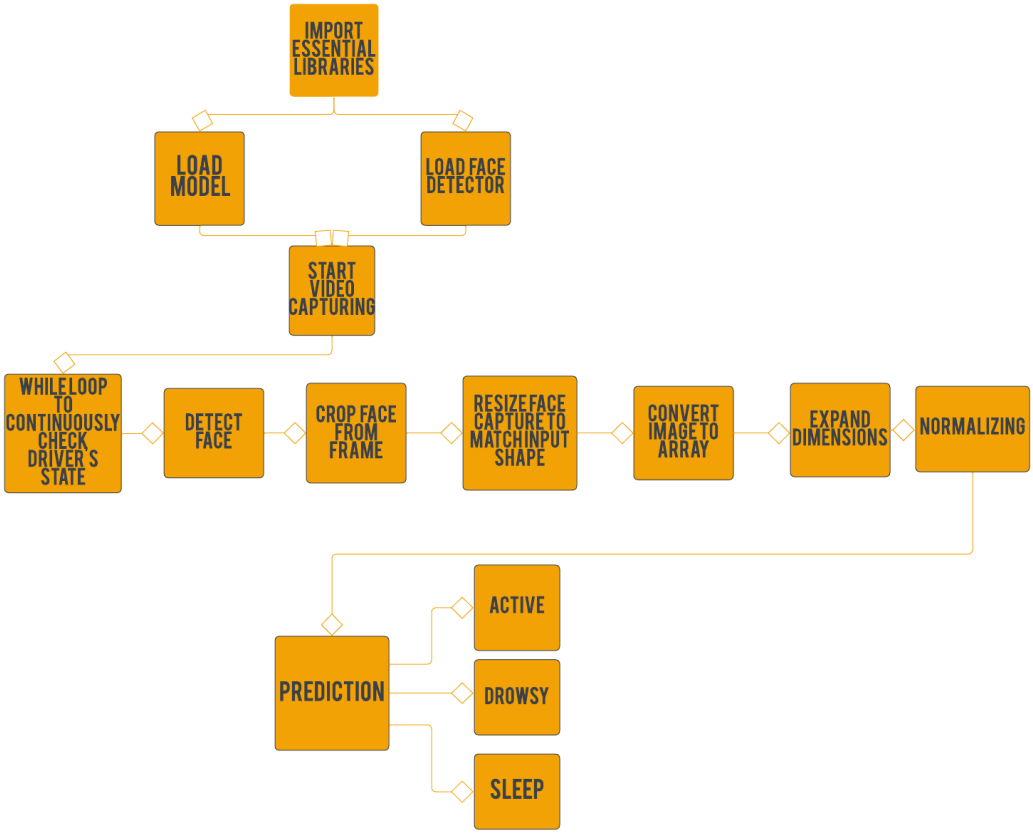
model	Training accuracy	Validation accuracy	Test accuracy
VGG-16	0.9431	0.9381	0.9438
Mobilenetv2	0.9663	0.9628	0.9187
Mobilnetv3small	0.9578	0.7692	0.8438
ResNet50v2	0.9605	0.9659	0.9562
ResNet50v2 with fine-tuning	0.9807	0.9567	0.9438
ResNet50v2 with fine-tuning and face-cropping dataset	0.9520	0.9692	0.9531

3.3.7 Model interference with camera:

The next phase is taking live predictions from camera and pass it to the model as image to make prediction but before image is passed to face detector in order to face-crop the image to avoid insignificant information from the image. Then resize the image to match model's input shape then convert the image to array and expand dimensions and finally normalizing.

Workflow

The workflow shown in the below figure is as follows: starting with importing essential libraries following with loading fatigue detection model and face detector. Then, start video capturing following with data pre-processing to match model input specifications and finally make predictions of 3 classes [Active-Drowsy-Sleep]



Code:

importing libraries

```
import cv2
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

Loading model

```
model=tf.keras.models.load_model("E:/Jupyter_Workstation/cropped_v2.h5")
```

Loading face detector

```
face_cascade= cv2.CascadeClassifier('E:/Jupyter_Workstation/haarcascade_frontalface_default.xml')
```

```
cap = cv2.VideoCapture(0)
```

```
while True:
    # Read a frame from the webcam
    ret, frame = cap.read()

    # Increment the frame counter
    count += 1

    # Only process every 10th frame
    if count % 10 == 0:


        # Detect faces using the Haar Cascade classifier
        faces= face_cascade.detectMultiScale(frame, 1.3, 5)

        # Loop over all detected faces
        for (x, y, w, h) in faces:
            # Crop the face from the frame
            face = frame[y:y+h, x:x+w]

            # Resize the face to match the model input shape
            face = cv2.resize(face, (224, 224))
            face= np.array(face)
            face=np.expand_dims(face, axis = 0)
            face=face/255.


            # Perform model prediction on the face
            result = model.predict(face)
            predicted_class = np.argmax(result)
            if predicted_class==0:
                prediction="Active"
                fcolor=(0,255,0)
            elif predicted_class==1:
                prediction="Drowsy"
                fcolor=(255,255,0)
            elif predicted_class==2:
                prediction="Sleep"
                fcolor=(255,0,0)
```

```
     # Draw a rectangle around the detected face
    cv2.rectangle(frame, (x, y), (x+w, y+h), fcolor, 2)

            # Add the predicted label to the frame
    cv2.putText(frame, prediction, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, fcolor, 2)
    # Display the resulting frame
    cv2.imshow('frame', frame)

    # Exit the loop if esc is pressed
    if cv2.waitKey(1) ==27:
        break

# Release the webcam and close all windows
cap.release()
cv2.destroyAllWindows()
```

3.3.8 Implementation of solution using Raspberry pi

Raspberry pi used along with a USB camera to continuously make predictions on driver's alertness level. The following code illustrates how the solution implemented.

Using face detection following with taking the face part only of frame to be predicted using the deep learning model.

Convert TensorFlow model to TensorFlow lite.

There are various advantages to moving a model from TensorFlow to TensorFlow Lite when using a Raspberry Pi. TensorFlow Lite offers lightweight, effective operations that make use of the Raspberry Pi's capabilities and is targeted for systems with limited resources. TensorFlow Lite models have a smaller memory and storage footprint, allowing them to run on the Raspberry Pi with less memory usage. In addition, TensorFlow Lite offers quicker inference speed because to hardware acceleration and effective methods. In addition to supporting well-known programming languages and taking advantage of hardware acceleration features like the Edge TPU, it works smoothly with the Raspberry Pi environment. TensorFlow Lite also enables offline functionality and privacy because models may be executed locally on the device without constant network connectivity.

Code:

```python
import tensorflow as tf

# Load the pre-trained TensorFlow model
model = tf.keras.models.load_model('cropped_v2.h5')

# Convert the TensorFlow model to a TFLite model
converter = tf.lite.TFLiteConverter.from_keras_model(model)

# Set any desired optimization options
converter.optimizations = [tf.lite.Optimize.DEFAULT]

# Convert the TFLite converter object to a TFLite model file
tflite_model = converter.convert()

# Save the TFLite model to disk
with open('my_model.tflite', 'wb') as f:
    f.write(tflite_model)
```

Model implementation on Raspberry pi using TensorFlow lite.

The implementation of making predictions on a Raspberry Pi using a deep learning model and TensorFlow Lite is demonstrated in the following code snippet. The code enables effective model deployment on resource-constrained devices like the Raspberry Pi by utilising TensorFlow Lite's power. It demonstrates how to load the model, prepare the input data, and produce predictions. The Raspberry Pi is now capable of carrying out real-time inference tasks thanks to the integration of deep learning and TensorFlow Lite.

```python
import cv2
import numpy as np
import tensorflow as tf

interpreter = tf.lite.Interpreter(model_path="/home/youssef/Desktop/fatigue_detection/my_model.tflite")
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

face_cascade= cv2.CascadeClassifier('/home/youssef/Desktop/fatigue_detection/haarcascade_frontalface_default.xml')
cap = cv2.VideoCapture(0)
count = 0
x=0
y=0
w=0
h=0
prediction=""
fcolor=(0,0,0)


while True:
    # Read a frame from the webcam
    ret, frame = cap.read()

    # Increment the frame counter
    count += 1

    # Only process every 10th frame
    if count % 10 == 0:
```

```
    # Detect faces using the Haar Cascade classifier
    faces= face_cascade.detectMultiScale(frame, 1.3, 5)

    # Loop over all detected faces
    for (x, y, w, h) in faces:
        # Crop the face from the frame
        face = frame[y:y+h, x:x+w]

        # Resize the face to match the model input shape
        face = cv2.resize(face, (224, 224))
        face= np.array(face)
        face=np.expand_dims(face, axis = 0)
        face = np.float32(face/255.)
```

```
            # Perform model inference on the face
            interpreter.set_tensor(input_details[0]['index'], face)
            interpreter.invoke()
            result = interpreter.get_tensor(output_details[0]['index'])
            predicted_class = np.argmax(result)
            if predicted_class==0:
                prediction="Active"
                fcolor=(0,255,0)
            elif predicted_class==1:
                prediction="Drowsy"
                fcolor=(0,255,255)
            elif predicted_class==2:
                prediction="Sleep"
                fcolor=(0,0,255)
```

```
    # Draw a rectangle around the detected face
    cv2.rectangle(frame, (x, y), (x+w, y+h), fcolor, 2)

            # Add the predicted label to the frame
    cv2.putText(frame, prediction, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, fcolor, 2)
    # Display the resulting frame
    cv2.imshow('frame', frame)

    # Exit the loop if esc is pressed
    if cv2.waitKey(1) ==27:
        break

# Release the webcam and close all windows
cap.release()
cv2.destroyAllWindows()
```

This chapter focused on the use of OpenCV or deep learning to identify fatigue using Artificial Intelligence (AI) technologies. Deep learning, specifically the ResNet50v2 architecture, was suggested as the approach. The implementation of employing a camera to make model predictions and improving the model for deployment on a Raspberry Pi were also covered in this chapter.

TensorFlow Lite, a lightweight version appropriate for deployment on resource-constrained devices like the Raspberry Pi, was used to convert the deep learning model during the process. The model's execution on the Raspberry Pi was efficient and effective thanks to this optimization phase.

Using the trained deep learning model, the implementation on the Raspberry Pi enabled for real-time fatigue detection. The system might use the camera feed to continuously check for signs of exhaustion, send out alerts in a timely manner, or take the necessary precautions to reduce dangers.

This chapter demonstrated a workable solution for the real-world application of AI in fatigue detection scenarios by providing a thorough overview of the

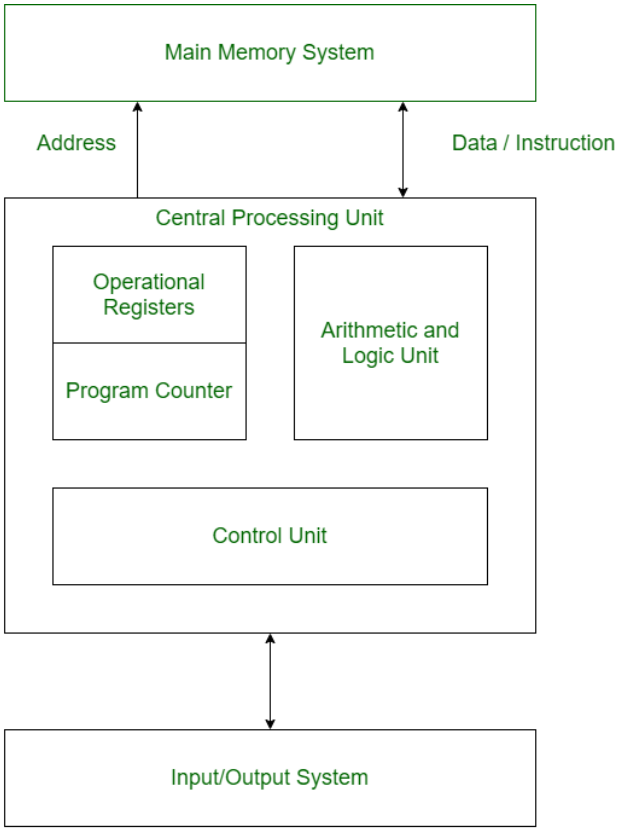
complete pipeline, from creating the deep learning model for tiredness detection to its implementation on the Raspberry Pi

Chapter 4: Embedded Systems Unit Design & Implementation

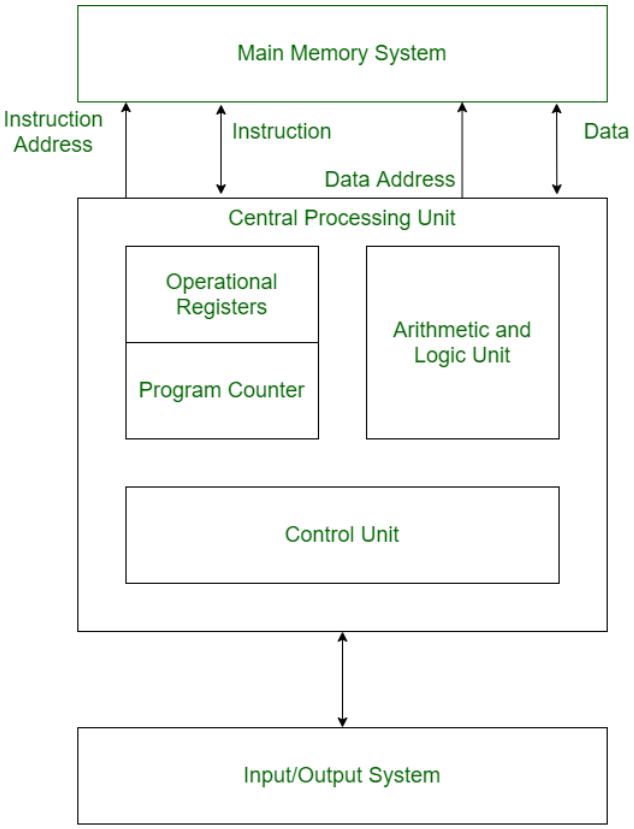
This chapter is meant to discuss the embedded systems part of the project. It provides a brief introduction on embedded systems concepts, and the needed peripherals for this project. Finally, it dives right into the modules interface with the used MCU ATmega32.

4.1 Embedded Systems Concepts

Embedded systems are computing systems with limited resources (processors, memory & I/O) used in order to perform a specific task. A microcontroller is a system on chip (SOC) that consists of a processor, memory and input & output peripherals, so in brief it looks like an IC although carrying all the needed elements for an embedded system ready for programming e.g. AVR or ARM. A processor is responsible for performing instructions that a computing system is supposed to perform. Memories hold a crucial role in any computing system. People throw away their mobile phones if they run out of memory. Simple as this, a device with a powerful processor and a good state can be thrown away in a matter of seconds. Memories are used to store data. There are two types of them: Volatile, these are memories that lose the data they carry once the device is powered off e.g. RAM, and non-volatile, these never lose their data even if the devices are powered off e.g. ROM. There are two system architectures for embedded systems: Harvard architecture, a computer architecture in which separate storage and buses (paths) are assigned to data and to instructions. Second is Von Neumann architecture, where program instructions and data share the same memory and pathways.



Von Neumann Architecture



Harvard Architecture

Now that the processor, memory, and internal architecture parts have been discussed, it is time to study how embedded systems interact with users. This is the input & output peripherals' job! They act as a channel between the user and the processor. Their assigned registers in the microcontroller are called "GPIO" registers which stands for General Purpose Input Output. Some examples for these peripherals are:

- Digital Input Output (DIO)
- Analog To Digital Converter (ADC)
- Liquid Crystal Display (LCD)
- Digital To Analog Converter (DAC)
- Timers & Pulse Width Modulators (PWM)
- Serial Peripheral Interface (SPI)
- Inter Integrated Circuit (I2C)
- Universal Asynchronous Receiver Transmitter (UART)

The last three are serial communication protocols, which means that they are ways through which two microcontrollers can communicate together and exchange information.

In the upcoming sections, some of these peripherals will be discussed due to their use in this project. They will be explained and their drivers will, additionally, be provided.

4.2 Atmega32 Microcontroller Interface

The microcontroller chosen for this project is the 8-bit AVR ATmega32 from Atmel. It is known for its powerful processor, its high performance, and its low power consumption. Unlike "Arduino", it is not an open source controller which means that in order to use it, one is required

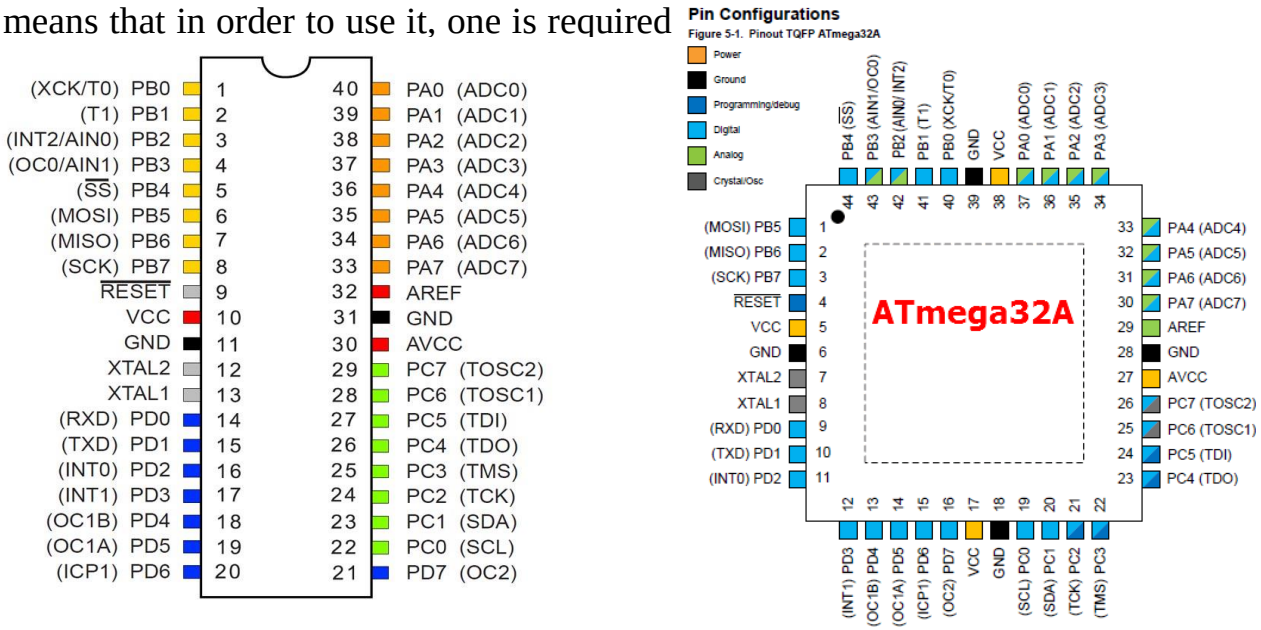


Figure 12 ATmega32 Pin Configuration

4.2.1 Digital Input Output Peripheral (DIO)

In this project, the DIO driver function is used in every code in order to define the direction and value of each port & pin that the sensors & modules are connected to.

A digital input output peripheral deals with digital signals either by generating it, and this way it is called an output or by receiving it, and this way it is called an input.

- **Digital Input**: It holds a certain threshold, if the input exceeds it then it is assigned as a 1, if it does not then it is a 0.
- **Digital Output**: It allows the developer to control the voltage using a computer. If he wants the output to be high, then the peripheral will produce an output that's 5v or 3.3v. If he wants it low then it is connected to the ground terminal.

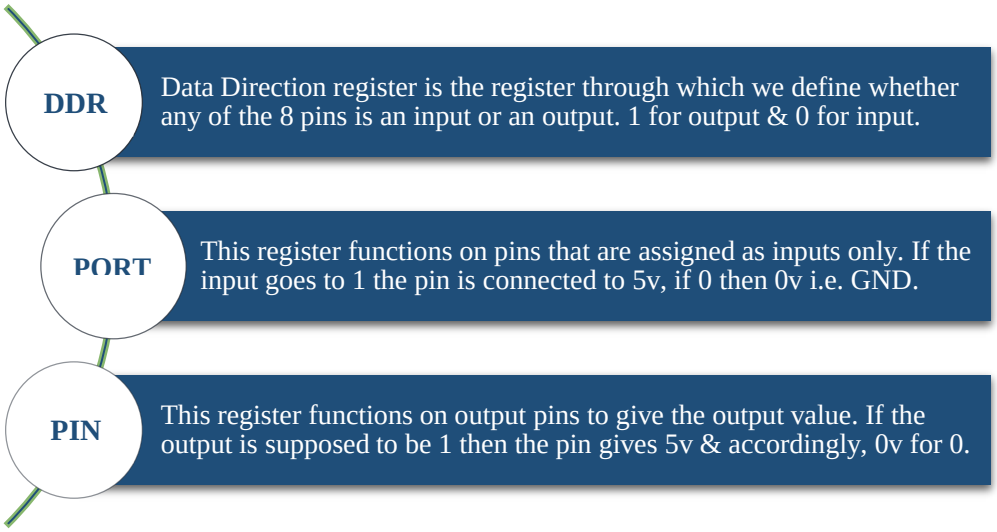
The building block of the DIO is the Tristate Buffer.

ATmega32 microcontrollers have Four DIO ports i.e. 32 DIO pins grouped as follows:

- **Port A**: Carries 8 DIO pins from A0 to A7
- **Port B**: Carries 8 DIO pins from B0 to B7
- **Port C**: Carries 8 DIO pins from C0 to C7
- **Port D**: Carries 8 DIO pins from D0 to D7

Refer to section 4.2 where the pins for every port are highlighted.

Every pin can work both ways either it be an input or an output. In addition, every port has three control registers, each of the size 8 bits, every bit of them corresponds to one pin in the port. The 3 registers are shown below:



For example: In order to configure pin A0 as a 5v output, this is what the code line should look as follows:

- DDRA = 0b 00000001
- PORTA = 0b 00000001

This means that only the 1st pin will act as an output & it will give 5v as an output.

Note that: The values of the registers can be assign in any numbering system either it be binary decimal, hexadecimal….etc. For example, if we want to assign all port D pins as outputs that give 0v we can write them this way:

- DDRD = 255
- PORTD = 0x 0F

However, as shown, binary and hexadecimal numbering systems are preceded by 0b & 0x respectively.

But, attention!!! Is it rational to write a code, especially if it is a huge one, that is full of numbers this way ? Is it a good practice ? Of Course not, this is why drivers are needed. Drivers are codes that written and kept to be used in every other code in order to make it readable or understandable. For instance, it is better to say, in your code, that “DIO Pin 7 in port A = Output” instead of DDRA = 10000000.

This is exactly what drivers do. Drivers are written in the form of .c and .h files and are kept to be used in every code in the future. They are written in the form of understandable functions so that they can be called in other codes easily. Shown below is the driver of the DIO peripheral: First is the DIO.h File: This holds the .c file functions prototype and the #define (renamed) values.

```c
#ifndef DIO_H_
#define DIO_H_

#include "STD_Types.h"
#include "BitMath.h"
#include "DIO_HW.h"
#include "DIO_Types.h"

void DIO_Write(DIO_ChannelTypes ChannelID,STD_LevelTypes Level);
STD_LevelTypes DIO_Read(DIO_ChannelTypes ChannelID);
void DIO_Toggle(DIO_ChannelTypes ChannelID);

#endif /* DIO_H_ */
```

Figure 13 DIO.H Driver

This is how the functions look from the inside in the .c file:

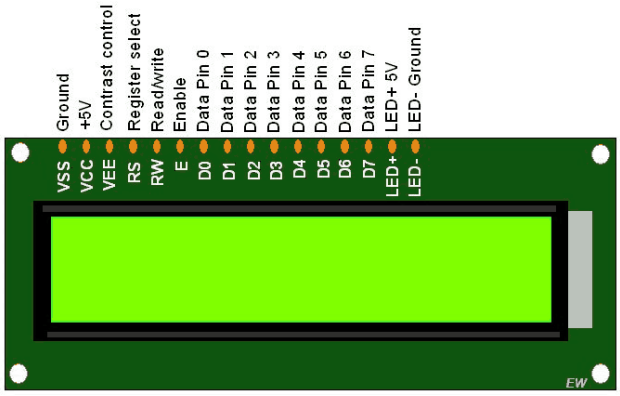
```c
#include "DIO.h"

void DIO_Write(DIO_ChannelTypes ChannelID,STD_LevelTypes Level){
    DIO_PortTypes Portx = ChannelID/8;
    DIO_ChannelTypes ChannelPos = ChannelID%8;
    switch(Portx){
        case DIO_PortA:
        if(Level == STD_High){
            SetBit(PORTA_Reg,ChannelPos);
        }
        else{
            ClearBit(PORTA_Reg,ChannelPos);
        }
        break;
        case DIO_PortB:
        if(Level == STD_High){
            SetBit(PORTB_Reg,ChannelPos);
        }
        else{
            ClearBit(PORTB_Reg,ChannelPos);
        }
        break;
        case DIO_PortC:
        if(Level == STD_High){
            SetBit(PORTC_Reg,ChannelPos);
        }
        else{
            ClearBit(PORTC_Reg,ChannelPos);
        }
        break;
        case DIO_PortD:
        if(Level == STD_High){
            SetBit(PORTD_Reg,ChannelPos);
        }
        else{
            ClearBit(PORTD_Reg,ChannelPos);
        }
        break;
    }
}
```

```c
STD_LevelTypes DIO_Read(DIO_ChannelTypes ChannelID){
    STD_LevelTypes Level = STD_Low;
    DIO_PortTypes Portx = ChannelID/8;
    DIO_ChannelTypes ChannelPos = ChannelID%8;
    switch(Portx){
        case DIO_PortA:
        Level = GetBit(PINA_Reg,ChannelPos);
        break;
        case DIO_PortB:
        Level = GetBit(PINB_Reg,ChannelPos);
        break;
        case DIO_PortC:
        Level = GetBit(PINC_Reg,ChannelPos);
        break;
        case DIO_PortD:
        Level = GetBit(PIND_Reg,ChannelPos);
        break;
    }
    return Level;
}
void DIO_Toggle(DIO_ChannelTypes ChannelID){
    DIO_PortTypes Portx = ChannelID/8;
    DIO_ChannelTypes ChannelPos = ChannelID%8;
    switch(Portx){
        case DIO_PortA:
        ToggleBit(PORTA_Reg,ChannelPos);
        break;
        case DIO_PortB:
        ToggleBit(PORTB_Reg,ChannelPos);
        break;
        case DIO_PortC:
        ToggleBit(PORTC_Reg,ChannelPos);
        break;
        case DIO_PortD:
        ToggleBit(PORTD_Reg,ChannelPos);
        break;
    }
}
```

4.2.2 Liquid Crystal Display

In our system, the driver is given clear indications of their current alertness level, heart rate, and the presence of obstacles using an LCD display, ensuring effective real-time communication of crucial information.

In embedded systems, status or parameter display is done using LCDs (Liquid Crystal Displays).



Eight data pins (D0-D7) and three control pins (RS, RW, and EN) make up the 16-pin LCD 16x2 device. The LCD's power supply and backlight are connected to the remaining 5 pins.

We can arrange the LCD in command mode or data mode with the aid of the control pins. They also assist in setting the read or write mode and the appropriate times to read or write.

Depending on the needs of the application, LCD 16x2 can be used in either 4-bit or 8-bit mode. We must transmit specific commands to the LCD in command mode in order to use it. and after the LCD has been set up to meet the needs, we can transfer the necessary data in data mode.

It has a register select (RS) and two registers:

i)**Command Register** (RS=0): Instructions for processing, initializing, controlling the display, and clearing the screen are stored in this register .

ii) **Data Register (RS=1):** Holds the information that will be shown on the LCD. Each character is saved with its ASCII value.

Driver

As mentioned before, the LCD can used in both 4- bit and 8-bit modes. In order to reduce the number of pins utilized by LCD on ATmega32 , 4-bit mode is used.

LCD uses DIO.

LCD pins:

- RS: PB1
- RW: PB2
- E: PB3
- Data pins (D4-D7)

LCD.H File :

```
 *  Author: youssef rashad
 */


#ifndef LCD_H_
#define LCD_H_
#include "DIO.h"
#define F_CPU 16000000UL
#include <util/delay.h>

#define LCDPort PORTA_Reg


#define RS DIO_ChannelB1
#define RW DIO_ChannelB2
#define E DIO_ChannelB3
void LCD_Init();
void LCD_Cmd(Uint8 cmd);
void LCD_Char(Uint8 data);
void LCD_String(Sint8 * string);
void LCD_String_xy (char row, char pos, char *str);



#endif /* LCD_H_ */
```

LCD.C File:

```
#include "LCD.h"
void LCD_Init()
{
    DIO_Write(RW,STD_Low);
    DIO_Write(E,STD_High);
    _delay_ms(20);//from data sheet start with delay
    LCD_Cmd(0x33);//from data sheet send 3 then 3
    _delay_us(100);
    LCD_Cmd(0x32);
    LCD_Cmd(0x28);//function set from data sheet
    LCD_Cmd(0x06);//entry mode set from data sheet
    LCD_Cmd(0x0C);//DISPLAY ON
    LCD_Cmd(0X01);//CLEAR DISPLAY
    _delay_ms(2);


}
void LCD_Cmd(Uint8 cmd)
{
    LCDPort=(LCDPort& 0X0F)|(cmd&0xF0);//SEND HIGH NIPPLE
    DIO_Write(RS,STD_Low); //0 COMMAND REGISTER from data sheet
    DIO_Write(E,STD_Low);
    _delay_us(100);
    DIO_Write(E,STD_High);

    _delay_ms(5);

    LCDPort=(LCDPort& 0X0F)|(cmd<<4);//SEND LOW NIPPLE
    DIO_Write(RS,STD_Low); //0 COMMAND REGISTER from data sheet
    DIO_Write(E,STD_Low); //0 start to latch data to LCD from data sheet
    _delay_us(100);
    DIO_Write(E,STD_High);
    _delay_ms(2);
}
```

```
void LCD_Char(Uint8 data)
{
    LCDPort=(LCDPort& 0X0F)|(data&0xF0);//SEND HIGH NIPPLE
    DIO_Write(RS,STD_High); //1 DATA REGISTER from data sheet
    DIO_Write(E,STD_Low);
    _delay_us(100);
    DIO_Write(E,STD_High);

    _delay_ms(5);

    LCDPort=(LCDPort& 0X0F)|(data<<4);//SEND LOW NIPPLE
    DIO_Write(RS,STD_High); //1 DATA REGISTER from data sheett
    DIO_Write(E,STD_Low); //0 start to latch data to LCD from data shee
    _delay_us(100);
    DIO_Write(E,STD_High);
    _delay_ms(2);
}
void LCD_String(Sint8 * string)
{
    Uint8 i=0;
    while(string[i] !='\0')//lehad maylay2y null
    {
        LCD_Char(string[i]);
        i++;
    }

}
void LCD_String_xy (char row, char pos, char *str)/* Send string to LCD with xy position */
{
    if (row == 0 && pos<16)
    LCD_Cmd((pos & 0x0F)|0x80); /* Command of first row and required position<16 */
    else if (row == 1 && pos<16)
    LCD_Cmd((pos & 0x0F)|0xC0); /* Command of first row and required position<16 */
    LCD_String(str);        /* Call LCD string function */
}
```

Figure 15 LCD.C Driver

4.2.3 Analog To Digital Converter (ADC)

Peripheral used to interfase the PPG pulse sensor with the ATmega32 AVR.

Every natural signal in our world is an analog signal. That is exactly why analog to digital converters are some of the main components in any embedded system. Computing systems only understand digital signals (1s & 0s) so the physical value sensed by sensors are translated into digital values using ADCs in order to be processed.

- **Analog Signals**: Signals that are continuous (Carry values) all the time. They also have an infinite number of magnitudes (Continuous in magnitude).

- **Digital Signals**: Signals that are discrete in time and magnitude.

ADCs are the gate between the actual physical world and the microcontrollers. Sensors are devices that sense the physical environment conditions such as the temperature, pressure, motion, light intensity…etc. and transform them into electrical signal values in volts, if not, then a voltage divider is needed. ADCs only accept inputs as electrical voltage signals. Sensors are also a form of transducers because they transform one form of energy to another.

Simply put, an ADC captures an analog voltage in one instant in time and produces a digital output code which represents this analog voltage value. The number of the output digital binary digits used to represent the analog voltage is called the "**Resolution**" of the ADC. ADC circuits mostly depend on comparators. This will be explained later.

There are five major types of ADC that are used today, and these are: Delta-Sigma, Dual-Slope, Pipelined, Flash ADC, and Successive Approximation ADC.

ATmega32 ADC type is the **Successive Approximation** ADC. This type of ADC uses a series of comparisons to determine each bit of the output result. The resolution of SAR (Successive Approximation Register) in ADCs ranges from 8 to 18 bits. They consist of 4 parts:

- **Sample & Hold Circuit**: It acquires the input voltage Vin that is the signal coming from the sensor. It samples this signal and then it is compared with the specific output signal of the DAC. To determine the output bit.
- **Analog Voltage Comparator**: It compares V_{in} to the output of the internal DAC and outputs the result of the comparison to the successive-approximation register (SAR).
- **Successive Approximation Register**: It operates by successively dividing the voltage range by half, as explained in the following steps:

(1) The MSB is initially set to 1 with the remaining three bits set as 000. The digital equivalent voltage is compared with the unknown analog input voltage,

(2) If the analog input voltage is greater than the digital equivalent voltage, the MSB is kept as 1 and the second MSB is set to 1. Otherwise, the MSB is set to 0 and the second MSB is set to 1. Comparison is made as given in step (1) to decide whether to retain or reset the second MSB.

- **Digital To Analog Converter (DAC):** For each bit, the SAR outputs a binary code to the DAC that is dependent on the current bit under test and the previous bits already approximated. The DAC then converts the binary code and sends it to the comparator. If the result of the DAC output subtracted from the analog input is less than 0 the bit under scrutiny is set to 0. And the process continues until the 8 bits are done with.

Once all bits have been approximated, the digital approximation is sent as an output at the end of the conversion (EOC).

The SAR operation is best explained as a binary search algorithm.

For example, If we consider the example of an analog input value of 0.425 V and a voltage reference of 1 V, we can approximate the output of an 8 bit ADC as follows:

(1) Set the first bit of the 8 bit output to 1 accordingly the output of DAC is 0.5

(2) 0.5 subtracted from 0.425 is less than 0, then set the first bit of the output to 0

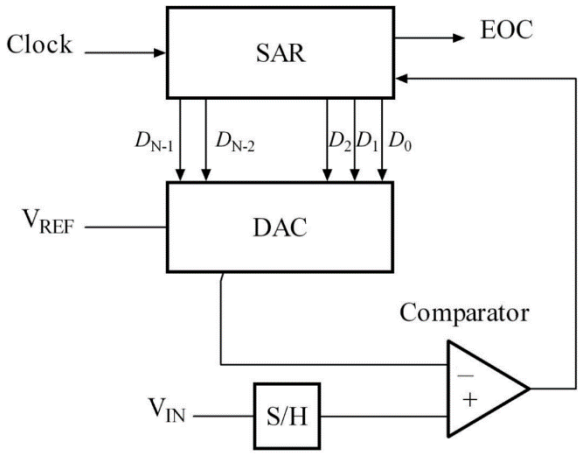


Figure 16 ADC Internal Circuit

(3) Set the second bit of the 8 bit output to 1, then the output of DAC is 0.25

(4) 0.25 subtracted from 0.425 is greater than 0, then the 2nd output bit is 1

(5) Set the third bit of the 8 bit output to 1, accordingly DAC output is 0.375

(6) 0.375 subtracted from 0.425 is greater than 0, the third bit of the output is 1

This process is repeated for all 8 bits until the output is determined to be:

01101100.

It is important when studying ADCs to understand their terminology:

- **Input voltage**: The analog signal coming from the sensor
- **Reference Voltage**: The maximum voltage that can be measured by the ADC
- **Resolution**: The number of bits in the output register i.e. The number of bits of the equivalent digital output.
- **Clock**: Controls the conversion speed. MCU works at 8MHz and ADC operates from 50kHz to 200KHz so a clock pre-scale is needed.
- **Step Size**: The minimum voltage that can be sensed by the ADC.

Note that: The atmega32 microcontroller holds a 10 bit ADC, meaning we will get a digital output range from 0 to 1023. i.e. When the input is 0v, the digital output will be 0v & when input is 5v (at vref=5v), we will get the highest digital output corresponding to 1023 steps, which is 5V. Here, it is important to state that the digital output value is defined as the number of steps from 0 to 1023. Now the step size plays an important role in this conversion. It is shown in the following equations:

$$\text{Step size} = \frac{\text{Reference voltage}}{2^n}$$

$$\text{Digital Output Value} = \frac{\text{Input voltage}}{\text{Step Size}}$$

In order to interface a sensor with the ADC in the MCU it has to be connected to a pin in "**PORT A**" because port A is the ADC port.

The ATmega32 datasheet provides all the details needed in order to interface any device with its ADC. There are some registers e.g. ADCSRA, ADMUX…etc. which are used to set the ADC to the desired functionality. For instance, the user can set the desired pre-scale value, the desired reference voltage, the adjustment of the bits in the registers…..etc.

The program file code for the ADC driver is provided below:

ADC_prog.c

```c
#include "stdtypes.h"
#include "ADC_private.h"
#include "macros.h"
#include "ADC_iterface.h"
#include "ADC_config.h"
void ADC_vidinit(void) // ADC initialisation
{ // Register adjustment & Vref. selection
    ADMUX = (RES_ADJ << ADLAR)|(Vref<<REFS0);
    // @vref = 3 in interface.h i.e 11 shift it 6 times bec. REF0=bit6. This chooses ref voltage = 2.56 (Datasheet)
    // ADC Enable and pre-scale selection
    ADCSRA = (1<<ADEN)|(1<<ADPS0)|(1<<ADPS1)|(1<<ADPS2); // pre-scale division factor = 128
}
// Function to get the digital reading from the ADC
u16 ADC_GET_READING (u8 ch)
{
    ch = ch & 0b00000111; // u8 is from 0 to 255, but we only have 7 ADC channels, so if user entered a larger no. >> mask it to 7

    ADMUX=(ADMUX & 0xF8)|ch; // clears the channels in ADMUX not to overwrite on a previously saved value

    ADCSRA= (1<<ADEN)|(1<<ADSC); // ADC enable again to wake from sleep mode (if happened) & start conversion (Data sheet)

    while (GET_BIT(ADCSRA, ADIF) ==0);
   //Polling, if conversion is still not done with i.e flag=0 keep looping, but if condition not satisfied i.e flag = 1 (conversion done with) leave the loop

    return ADC_RES; // after leaving the loop return the result of the ADC
}
```

Figure 17 ADC Program File Code

4.2.4 Universal Asynchronous Receiver Transmitter (UART)

This communication protocol is used for the SIM808 module interface with ATmega32.

Some applications are large enough to exceed one microcontroller's capability. For such cases, there has to be a way that allows the user to use more than one MCU, and that allows him to establish a secure communication channel between them so that the data and information can be exchanged easily and safely.

In order for a source and a destination to communicate, there has to be some specifications and rules. Shown below are the crucial conditions for any two objects to communicate:

- **Medium**: Could be **Wired**, cost depending on length yet has high security, or **Wireless**, can travel long distances yet has low security.
- **Data Direction:** Could be **Simplex, Half duplex or Full duplex**.
- **Synchronisation:** Communication Could be **Synchronous**, meaning one MCU sets the clock and the other follows, or **Asynchronous**, every MCU has its own clock.
- **Transmission Technique:** Could be parallel, all data bits sent at the same time, or serial, bit by bit are sent.
- **Receiver and Transmitter Relationship:** Could be **Peer to Peer**, communication can happen at any time without the need for a permission, or **Master VS Slave**, a slave can not send data except with the master's permission. Master VS slave can come in one of three types: Single master multi slaves, multi masters multi slaves, or multi master no slave.
- **Throughput:** It is the amount of useful data in all the transmitted data

In ATmega32 MCU there are three communication protocols, all are serial, these are: UART, SPI, and I2C. They will all be discussed in the upcoming sections.

The communication protocol is the method of communication with two aspects defined: The hardware interface & the data frame format.

In this section, the UART is discussed:

A) **The Hardware Interface**: The communication using this protocol is:

- Wired
- Peer to Peer
- Asynchronous

(ATmega32 also supports synchronous UART. It is called USART)

- Data Direction: Full Duplex
- Transmission Technique: Serial

B) **The Frame Format**: It is the pattern of the 1s & 0s occurring that differentiates every communication protocol from the other.

The idle state (The state when the MCU not transmitting) is the High state. The frame is shown in the figure down below:

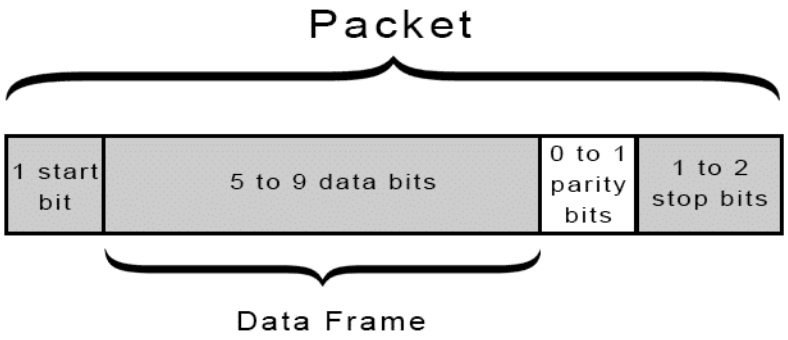


Figure 18 UART Data Frame

C) **The Throughput:** It is the minimum data bits that can be sent divided by the maximum frame bit number.

$$\text{Throughput} = \frac{5}{1+5+1+2} = \frac{5}{9} = 55\%$$

This means that 55% of the frame data are useful data bits. This is the minimum throughput of a UART frame. The maximum throughput is calculated as follows:

$$\text{Throughput} = \frac{9}{1+9+0+1} = 81\%$$

This means that 81% of the frame data bits are useful data bits.

The parity bit is optional, it is set to 0 if the number of 1 bits at the transmitter is an even number, and set to 1 if the number of bits at the transmitter is an odd number.

Upon referring to the ATnega32 datasheet, you can find the needed registers to develop the UART driver, and the bits to be set in them so as to give the needed baud rate & functionality.

The UART driver file is provided next:

UART.c

```c
#include<avr/io.h>
#include "stdtypes.h"
#include "UART.h"

#define F_CPU 8000000UL
#define  BAUD_PRESCALE (((F_CPU / (baud*16UL)))-1) //Equation of baud rate

void UART_INIT(u32 baud)
{
    UCSRB |= (1<<RXEN) | (1<<TXEN); // UART Receive and Transmit enable
    UCSRC |= (1<<URSEL) | (1<<UCSZ0) | (1<<UCSZ1); // Set character size to 8-bit, and choose UCSRC not UBRRH
    // UBRRH = 0; // Because 51 is less than 8 bits (255) so this is not needed
    // UBRRL = 51; // Set Baud Rate to 9600 at 8 MHz clock frequency
    UBRRH = BAUD_PRESCALE >> 8;
    UBRRL = BAUD_PRESCALE;
}

void UART_SendByte (u8 data)
{
    /* Wait for empty transmit buffer */
    while ( !( UCSRA & (1<<UDRE)) );
    /* Put data into buffer, sends the data */
    UDR = data;
}

u8 UART_ReceiveByte ()
{
    /* Wait for data to be received */
    while ( !(UCSRA & (1<<RXC)) );
    /* Get and return received data from buffer */
    return UDR;
}
```

Problems Tasks Console Properties AVR Device Explorer AVR Supported MCUs Debugger Console

Figure 19 UART Driver File

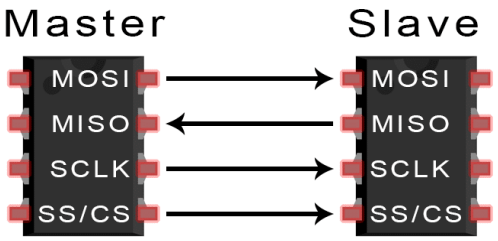
4.2.5 Serial Peripheral Interface (SPI)

In this project, the ATmega32 microcontroller and Raspberry Pi are connected using the SPI communication protocol, allowing the Raspberry Pi to continuously transmit the driver's status (active, drowsy, or sleep) to the AVR ATmega32 microcontroller.

Many devices use the SPI communication protocol, which is widely used. For instance, SPI is used to communicate with microcontrollers using 2.4 GHz wireless transmitters and receivers, SD card reader modules, and RFID card reader modules.

One particular benefit of SPI is the ability to send data uninterrupted. Any size stream of bits can be sent or received in it. With I2C and UART, data is sent in packets with a maximum amount of bits. Because start and stop conditions define the beginning and ending positions of each packet, data transmission is interrupted.

SPI Overview



•MOSI (Master Output/Slave Input)

- Line used for data transmission from the master to the slave.

•The Master Input/Slave Output (MISO) line is where the slave sends data to the master.

•SCLK (Clock) - The clock signal's line.

•SS/CS (Slave Select/Chip Select) - Line allowing the master to specify which slave should receive data.

Wires Used	4
Maximum Speed	Up to 10 Mbps
Synchronous or Asynchronous?	Synchronous
Serial or Parallel?	Serial
Max # of Masters	1
Max # of Slaves	Theoretically unlimited*

HOW SPI WORKS

- **The Clock**

The clock signal synchronizes the bits sampled by the slave with the bits output by the master. The frequency of the clock signal determines how rapidly data is sent since one bit of information is transferred during each clock cycle. As the

master configures and produces the clock signal, the master always initiates SPI communication.

When devices share a clock signal, a communication system is said to be synchronous. (SPI) Synchronous protocol interface. There are asynchronous methods available as well that do not rely on a clock signal. For instance, the time and speed of data transmission are controlled by the pre-configured baud rate that is defined for both participants in a UART discussion.

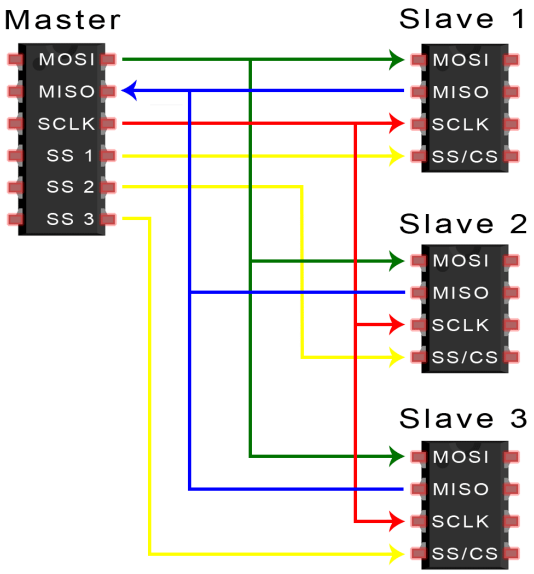
In SPI, the clock signal can be modified by adjusting the clock polarity and clock phase. When bits are emitting and when they are being sampled depend on these two qualities working together. By switching the clock's polarity, the master can enable bit output and sampling on either the rising or falling edges of the clock cycle. The clock phase can be set so that sampling and output occur on either the first or second edge regardless of whether the clock cycle is rising or falling.

- **Slave Select**

By lowering the voltage on the slave's CS/SS line, the master can select which slave it wants to communicate with. The slave select line is maintained at a high voltage level when it is off and not transmitting. The master may have multiple CS/SS pins, which enables the wiring of numerous slaves in parallel. Multiple slaves can be daisy-chained to the master if only one CS/SS pin is provided.

- **Multiple Slaves**

SPI can be set up to operate with just one master and slave or with multiple slaves under the command of one master. There are two ways to attach several slaves to the master. If the master has numerous slave choice pins, the slaves can be linked in parallel as shown below:



- **Advantages and Disadvantages of SPI**

❋ Advantages

- There are no start or stop bits, allowing for uninterrupted streaming of the data
- no convoluted I2C-like slave addressing system
- faster (nearly twice as fast) data transfer rate compared to I2C
- Separate MISO and MOSI lines allow for simultaneous data transmission and reception.

❋ Disadvantages

- (I2C and UARTs both utilize two wires)
- There isn't any confirmation that the data was correctly received (I2C has this).
- No error checking mechanism, unlike the parity bit in UART, and only supports a single master

❋ **SPI Driver:**

- SS-> PB4
- MOSI -> PB5
- MISO -> PB6

- SCK -> PB7

The following is the SPI.H driver File:

```
#ifndef SPI_H_
#define SPI_H_

#include <avr/io.h>
#define SPI_DataDirReg DDRB
#define SPI_PortReg PORTB
#define SS 4
#define MOSI 5
#define MISO 6
#define SCK 7

#define SPI_SlaveEn(Reg,BitNo) (Reg &=~ (1<<BitNo))
#define SPI_SlaveDIS(Reg,BitNo) (Reg |= (1<<BitNo))
typedef enum{
    Slave=0,Master
    }SPI_StateType;
void SPI_Init(SPI_StateType state);
unsigned char SPI_TXRX(unsigned char data);
unsigned char SPI_Rx(void);


#endif /* SPI_H_ */
```

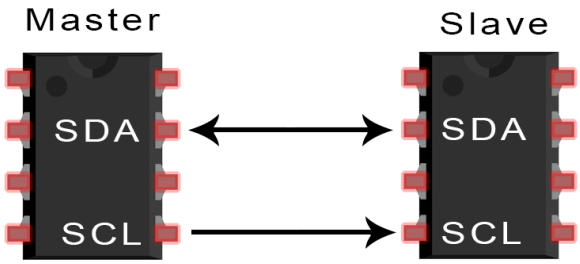
SPI.C File:

```
 *  Author: youssef rashad
 */
#include "spi.h"
void SPI_Init(SPI_StateType state)
{
    switch (state)
    {
        case Master:
        SPI_SlaveDIS(SPI_PortReg,SS);
        SPI_DataDirReg |= (1<<MOSI)|(1<<SS)|(1<<SCK);
        SPI_DataDirReg &=~ (1<<MISO);
        SPCR |= (1<<SPE)|(1<<MSTR);
        break;
        case Slave:
        SPI_DataDirReg &=~ (1<<MOSI)|(1<<SS)|(1<<SCK);
        SPI_DataDirReg |= (1<<MISO);
        PORTB |= (1<<SS); //shaghalt l pull-up resistor l i/p 3ando by default 1
        SPCR |= (1<<SPE);
        break;

    }
}
unsigned char SPI_TXRX(unsigned char data)
{
    SPDR = data;
    while (!(SPSR & (1<<SPIF)));
    return SPDR;
}
unsigned char SPI_Rx(void)
{
    while (!(SPSR & (1<<SPIF)));
    return SPDR;
}
```

4.2.6 I2C Communication Protocol Between IMU6050 and AVR

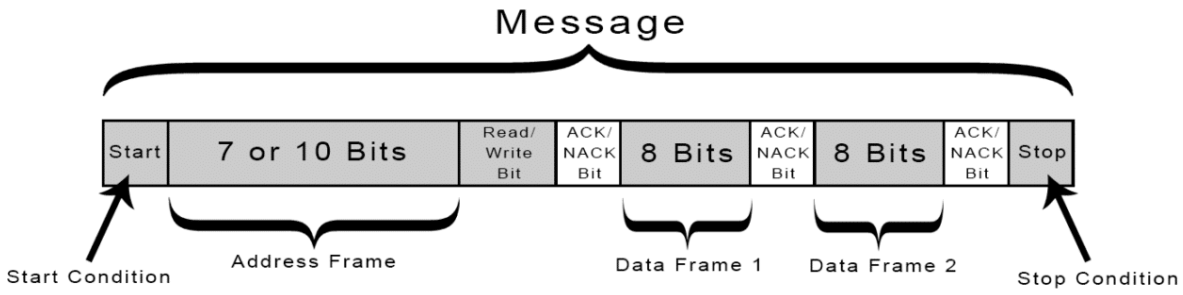
The usage of I2C in our project it the communication protocol between the AVR ATmega32 and IMU 6050 and this is how the I2C works.The best aspects of SPI and UARTs are combined in I2C. Similar to SPI, I2C enables the connection of several slaves to a single master and the control of one or more slaves by more than one master. When you wish to have multiple microcontrollers logging data to a single memory card or showing text on a single LCD, this is incredibly helpful.

❋ **Frame Format**:



•**SDA (Serial Data)** – The line for the master and slave to send and receive data.

•**SCL (Serial Clock)** – The line that carries the clock signal.

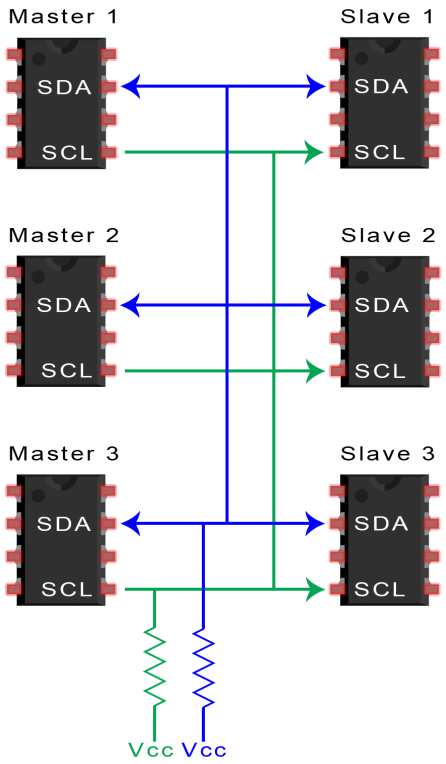


Start Condition: Before the SCL line changes from high to low, the SDA line changes from a high voltage level to a low voltagelevel.

Stop Condition: Following the SCL line's switch from low tohigh voltage, the SDA line goes from a low voltage level to a high voltage level.

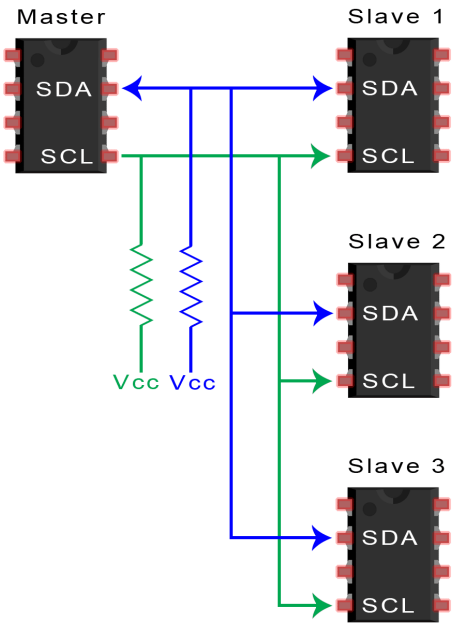
➢ Multiple Masters with Multiple Slaves

A single slave or a number of slaves can be tied to many masters. When two masters attempt to send or receive data simultaneously over the SDA line, the issue of having multiple masters in the same system arises. Each master must determine if the SDA line is low or high before broadcasting a message in order to fix this issue. The master should hold off on sending the message if the SDA line is low, as this indicates that another master is in charge of the bus. It is safe to send the message if the SDA line is high. Use the following schematic, connecting the SDA and SCL lines to Vcc using 4.7K Ohm pull-up resistors, to link several masters to numerous slaves:



➢ Single Master with Multiple Slaves

I2C employs addressing, which enables a single master to govern many slaves. There are 128 (27) distinct addresses possible using a 7-bit address. Although it's uncommon, using 10 bit addresses yields 1,024 (210) unique addresses. The SDA and SCL lines should be connected to Vcc using 4.7K Ohm pull-up resistors to connect numerous slaves to a single master



Advantages and Disadvantages of I2C

- **Advantages:**

•Only uses two wires

•Supports multiple masters and multiple slaves

•ACK/NACK bit gives confirmation that each frame is transferred successfully

•Hardware is less complicated than with UARTs

•Well known and widely used protocol

- **Disadvantages:**

.•Less rapid data transmission than SPI

•The data frame can only be 8 bits in size.

•Hardware implementation requires more complexity than SPI

4.2.4 Additional MCU Features (Timers & Interrupts):

These features have been widely used in the project such as in the ultrasonic, the pulse sensor, and the servomotor codes.

- **Timers** i.e. Counters are much needed in the microcontroller world. They are used for creating delays, counting events, controlling sampling frequency, generating waveforms, and triggering interrupts.

The three ATmega32 timers: Timer0, Timer1, and Timer2 can operate in different modes such as the normal (overflow), compare (CTC), fast PWM, phase correct PWM modes.

- **Interrupts** allow the processor to leave the function it is executing in order to do a more important one before returning to continue its execution job

again. It is more efficient than the polling technique that usually drains the processor.

There are two types of interrupts: Internal (Software) e.g. Division by 0 & external (Hardware) e.g. Some peripheral interrupt (UART, ADC, SPI..etc).

Events that trigger interrupt requests are provided in the interrupt vector table in ATmega32 datasheet.

In this project, timers and interrupts have been used without the need for drivers.

4.3 ADAS Unit Sensors and Modules Interface

This section is meant to discuss the embedded systems part of the project. Within it, any sensor, microcontroller or actuator interface with the MCU will be explained. Such sensors are PPG pulse rate sensor, ultrasonic sensor, gyroscope (MPU6050), & SIM808 GSM & GPS module. Actuators used in the project are Buzzer, LED, and servomotor.

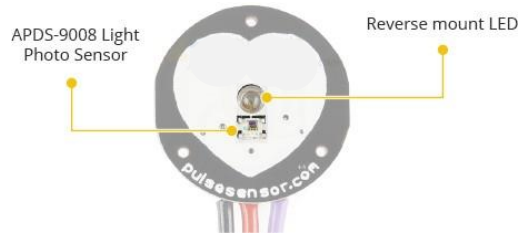
4.3.2 Photoplethysmography Pulse Sensor

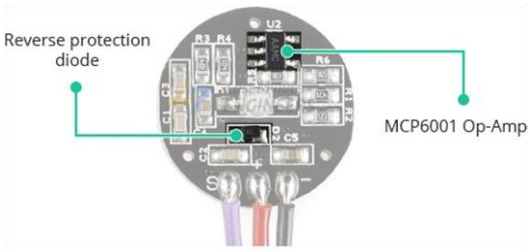
The heart is a fist-sized, vital, part of our bodies, it is the central organ in our cardiovascular systems. Additionally, it represents our bodies' last line of defence beside our brains. In normal cases, an adult's heart rate ranges between 60 to 100 beats per minute (BPM) depending on whether he is relaxed or nervous. It is easy to track a person's heart rate, or to save him in case of cardiac arrests or heart attacks if he's at home or at work, because simply people would notice. However, in real-time situations, such as driving on the road, it can be very challenging and risky. If a sudden heart attack strikes on the road, how can a driver be saved? Moreover, how can the surrounding drivers' lives be saved? It is critical to take immediate action in such cases.

A cardiac muscle functions in 2 phases: First is the **systolic** phase, in which it contracts to pump blood to the body. Here, in this phase, the pulse wave rises.

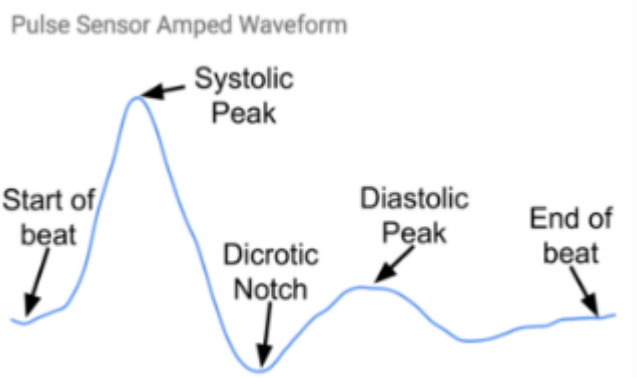
Second, is the **diastolic** phase in which the heart muscle expands so that the blood recedes from our body in order to fill its chambers. Again here, the pulse wave starts falling.

A photoplethismographic pulse sensor holds an infrared light emitting diode & a photo detector. In the back, you can find a low frequency operational amplifier that will amplify the sensed heart beat, and a noise cancellation circuit.





And why exactly is a noise cancellation circuit much needed? Well, back at the diastolic phase, when a pulse wave is falling, an uncontrollable dicrotic notch occurs. Of course this is not the only noise source, but it has a really bad effect on the sensor readings. However, its effect can be avoided while developing the code as this document will show later on.



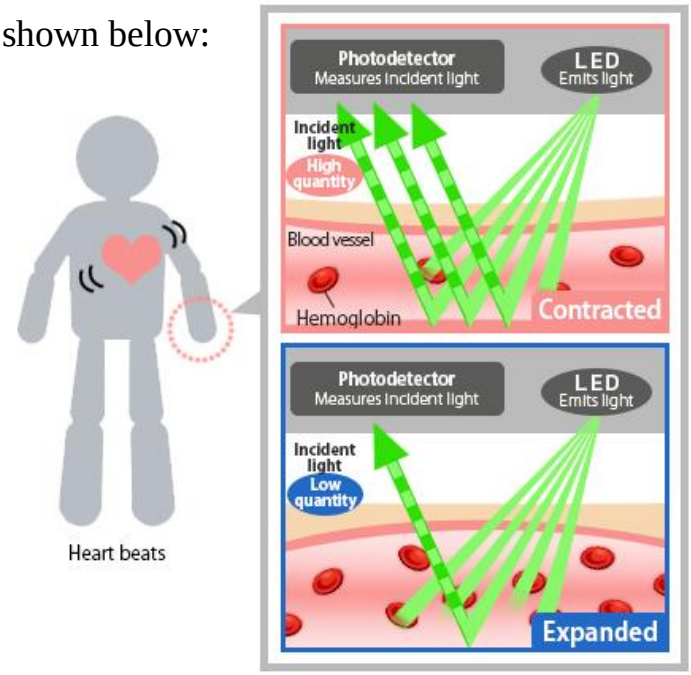
A PPG sensor calculates heart beats by measuring the inter-beat intervals between them. IBI is the time between a point location on one beat to the same point location in the successive beat.

❋ Mechanism of Action:

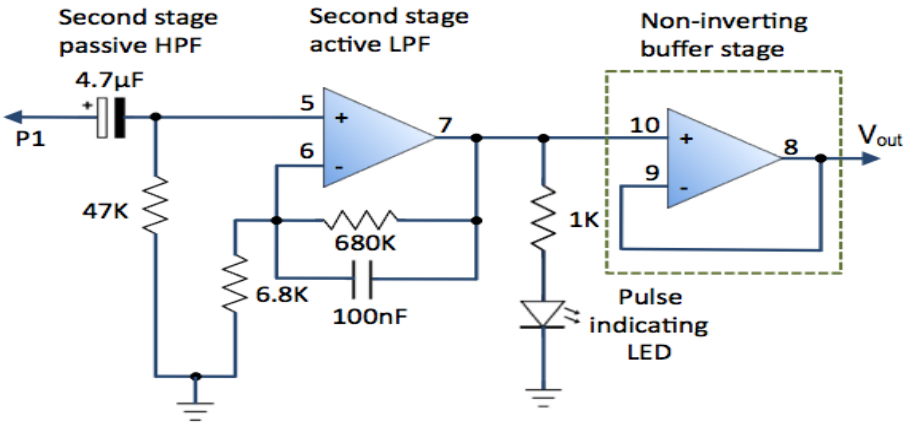
When the heart muscle contracts the blood fills our capillaries causing them to swell or expand, so imagine our finger in contact with the sensor:

The LED is continuously emitting infrared light, swelled capillaries filled with haemoglobin will absorb this light & only a little will be reflected back. The photodiode receives and measures the reflected light.

On the other hand, when the blood recedes from the capillaries, they contract giving room for the light to hit and be reflected back in large amounts to the photodiode as shown below:



This is the pulse sensor circuit. It consists of a band pass filter for noise cancellation. Also to provide the necessary frequency that corresponds to the normal human heart rate. Remember, the typical heart rate of an adult is between 60 to 100 BPM. In frequency terms, these values correspond to the range 1 to 1.7 Hz. So the band pass filter is designed to provide such frequency. There is also an amplifier because heart beats are weak in areas like the fingers and ear lobes.



This is the general block diagram of the heart rate measuring system:

Amplification & Filtering

Using LM358 band pass Filter

Output BPM

The algorithm running on the MCU calculations & displays the BPM on LCD

Input Signal

Obtained By the 3mm Infrared LED

Sampling & Quantization

Performed By the Atmega32 MCU

- Sensor Interface with MCU

The following are the specifications of the pulse sensor:

Maximum Ratings	VCC	3.0 – 5.5V
	IMax (Maximum Current Draw)	< 4mA
	VOut (Output Voltage Range)	0.3V to Vcc
Wavelength	LED Output	565nm
	Sensor Input	525nm
Dimensions	L x W (PCB)	15.8mm (0.625")
	Lead Length	20cm (7.8")

Figure 20 Specifications of PPG Pulse Sensor

(1) pulse sensor transforms physical heart beats to analog fluctuations in volts.

(2) The 10-bit ADC receives the sensor readings through pin A5 (Port A pin 5) on MCU and stars conversion using successive approximation *(Previously discussed in section 4.2.2.)*

(3) A threshold is set in the code midway between 0 and 1023 (ADC Levels) to sense heart beats & avoid noise. If the signal value exceeds the threshold then it is a heart beat, otherwise, it's just noise.

(4) Timer0 is programmed to act as a sample counter in order to trigger an interrupt every 2ms to take a sample of the wave. Thus the sampling rate = 500 Hz. The timer is also programmed to work on compare mode (CTC) so as to trigger an interrupt function every time the counter register TCNT0 is equal to OCR.

(5) Once the interrupt service routine function (ISR) is triggered, the sample captured is compared to the threshold. If sample value > Threshold, therefore, it is counted as an appropriate sample, the IBI value is measured from 50% of a sample's amplitude to 50% of the next appropriate sample's amplitude. Next, every 10 successive IBI values are stored in an array and are then used to measure the BPM where

$$\text{BPM} = \frac{\text{IBI array values}}{60000}$$

Why 60000? Because the samples are taken in milliseconds and the desired number of beats is required in minutes. Also note that in the code, the dicrotic notch is avoided by waiting i.e. Not counting 3/5 of the last IBI value.

Another important note to consider: When the MCU receives a sample, it cannot just decide that a heart beat occurred, because such way, every sample that exceeds the threshold will be counted as a beat itself when the truth is not so. The truth is that every sample is just part of the pulse wave.

The following is the code flow chart of the pulse sensor:

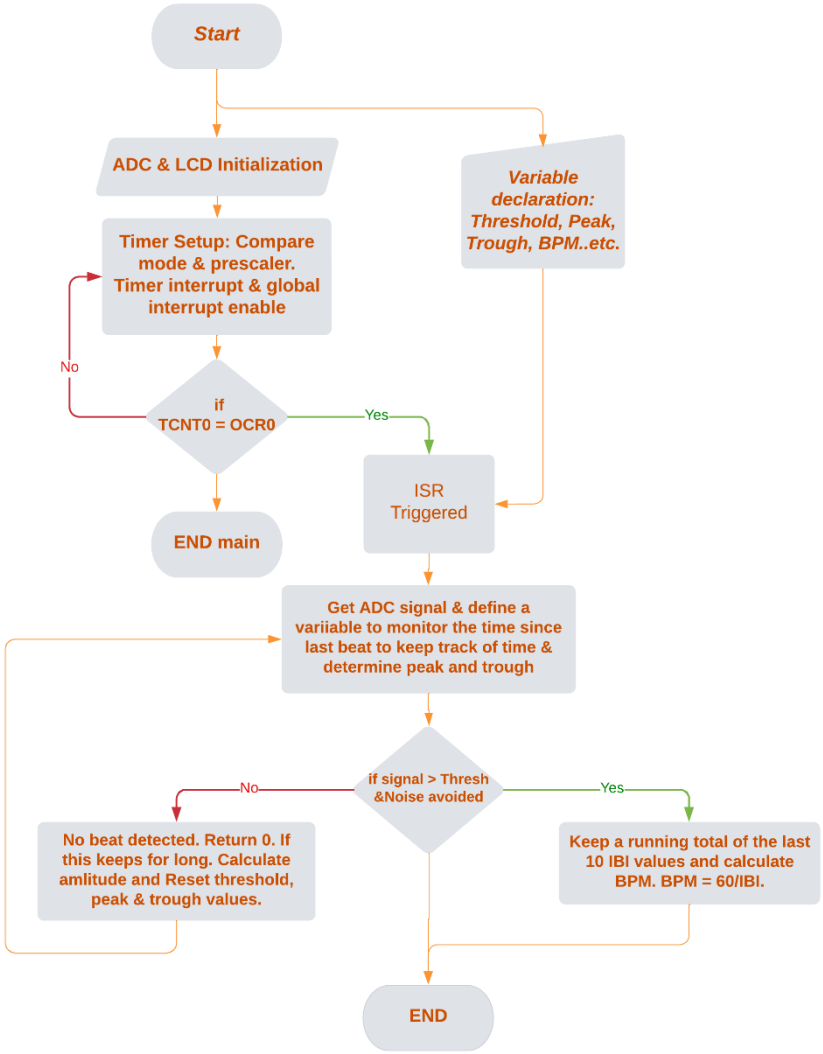


Figure 21 PPG Sensor Flow Chart

The PPG sensor code is provided in the Appendix section at the end of this document.

4.3.3 Ultrasonic Sensor for Obstacle Detection

The ultrasonic sensor is an essential part of our project because it provides a dependable method for detecting car obstacles, enabling the system to recognise and avoid them in real time.

The SONAR and RADAR systems are the basis for how the Ultrasonic Module HC-SR04 operates.

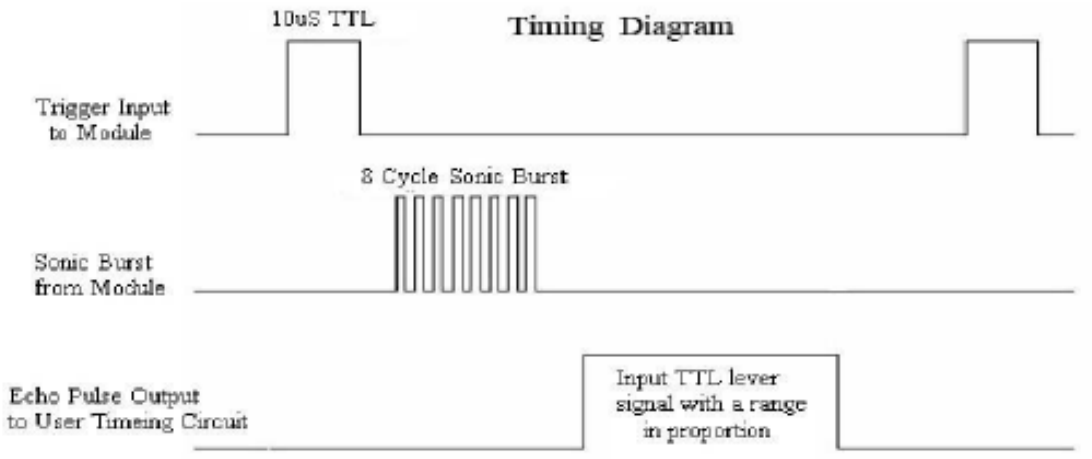
On a single board, the HC-SR-04 module houses an ultrasonic transmitter, receiver, and control circuit.

Only 4 pins—Vcc, Gnd, Trig, and Echo—are present on the module.

The sensor consists of two transducers. One acts as a transmitter that converts electrical signal into 40KHZ ultrasonic sound pulses , and the other one acts as a receiver.

Idea of operation

To begin the ranging process, the sensor needs to be supplied with (10 uS) pulse at the trigger input. Thereafter, the module will emit an 8-cycle burst of ultrasound at a frequency of 40 kHz, raising the echo. This eight-pulse pattern was specifically created to allow the receiver to differentiate between the transmitted pulses and background ultrasonic noise. The Echo is a distant object that is proportional to range and pulse width.



Calculations

$$\boldsymbol{Speed\ of\ sound} = \frac{\boldsymbol{Total\ distance(to\ travel\ and\ arrive\ from\ object)}}{\boldsymbol{Time}}$$

$$\boldsymbol{Speed\ of\ sound} = 343\frac{\boldsymbol{m}}{\boldsymbol{s}} = 34300\frac{\boldsymbol{cm}}{\boldsymbol{s}}$$

$$\boldsymbol{distance\ to\ an\ object(in\ cm)} = \frac{\boldsymbol{34300 * timer\ value}}{\boldsymbol{2}}$$

$$= \boldsymbol{17150 * timervalue}$$

$$\boldsymbol{Time\ to\ execute\ one\ instruction} = \frac{\boldsymbol{1}}{\boldsymbol{8 * 10\text{^}6}} = \boldsymbol{1.25 * 10^{-7}}$$

$$= \boldsymbol{0.125us}$$

Timer gets incremented after 0.125us

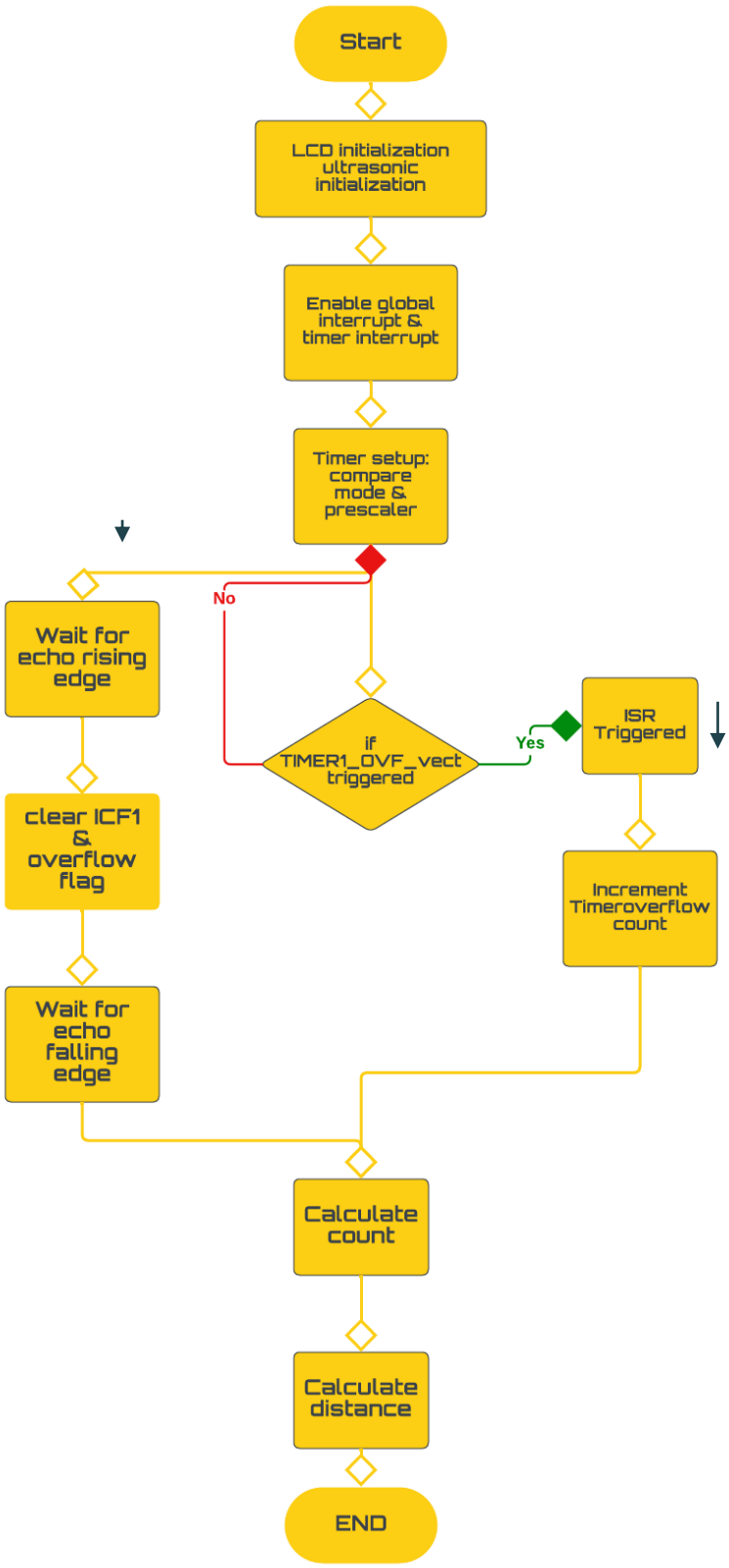
$$\boldsymbol{velocity\ of\ sound\ in}\frac{\boldsymbol{cm}}{\boldsymbol{s}} = \frac{\boldsymbol{distance\ (cm)}}{\boldsymbol{time}}$$

$$\boldsymbol{distance\ (cm)} = \boldsymbol{17150 * Timer\ value * 0.125 * 10^{-6}}$$

$$= \boldsymbol{0.125 * Timer\ value}\ \boldsymbol{58.3} = \frac{\boldsymbol{Timer\ value}}{\boldsymbol{466.47}}$$

(The above equation is the one used in the ultrasonic driver below to compute distance)

Flow chart


Start
LCD initialization
ultrasonic
initialization
Enable global
interrupt &
timer interrupt
Timer setup:
compare
mode &
prescaler
No
Wait for
echo rising
edge
if
TIMER1_OVF_vect
triggered
Yes
ISR
Triggered
clear ICF1
&
overflow
flag
Increment
Timeroverflow
count
Wait for
echo
falling
edge
Calculate
count
Calculate
distance
END

Configuration and driver

The ultrasonic sensor driver utilizes DIO, TIMER1 and interrupt where:

- Trig pin (PD5): set as output.
- Echo pin (PD6): set as input.
- Timer1: Enable timer 1 (Normal mode) and overflow interrupt.
  enable clock source and no pre-scaling and capture with rising (positive) edge, wait for rising edge then capture on falling edge
  enable timer with no pre-scaling and capture on falling edge.

```
//ultrasonic begin
double usdistance;
char string[10];
DDRD |= (1<<5); //MAKE TRIG PIN O/P
DDRD &=~ (1<<6); //MAKE ECHO PIN I/P
PORTD |= (1<<6); //TURN ON PULL-UP

TIMSK = (1<<TOIE1); //enable Timer1 ovf interrupts
TCCR1A = 0; //set all bit to zero (normal operation)
//ultrasonic end

ISR(TIMER1_OVF_vect)
{
    TimerOverFlow++;    /* Increment Timer Overflow count */
}
```

Ultrasonic.h:

```
#ifndef ULTRASONIC_H_
#define ULTRASONIC_H_
#define Trig_pin 5
#define echo_pin 6
#include "avr/io.h"
#define F_CPU 16000000UL
#include "util/delay.h"
#include <string.h>
#include <stdlib.h>
#include "DIO_GFC.h"
#include <stdio.h>
#include "LCD.h"

void ultrasonic_init();
double ultrasonic_measure();

extern int TimerOverFlow;
long count;
double distance;

#endif /* ULTRASONIC_H_ */
```

Ultrasonic.c:

```
#include "ultrasonic.h"

void ultrasonic_init()
{
    //GIVE 10US TRIGGER PULSE ON TRIG PIN
    PORTD |= (1<<Trig_pin);
    _delay_us(10);
    PORTD &=~ (1<<Trig_pin);

    TCNT1 = 0; //clear timer counter
    TCCR1B = 0X41;
    //TCCR1B = (1<< CS10) | (1<<ICES1); //enable clock source and no pre-scaling and capture with rising (positive) edge
    TIFR = (1<<ICF1);  //clear ICF1 by writing logic one
    TIFR = (1<<TOV1); //clear overflow flag
}
double ultrasonic_measure()
{
    while ((TIFR & (1 << ICF1)) == 0); //wait for rising edge
    TCNT1 = 0;
    //TCCR1B &=~ (1<<ICES1); //capture on falling edge
    //TCCR1B = (1<<CS10); //enable timer with no prescaling and capture on falling edge
    TCCR1B = 0X01;
    TIFR = (1<<ICF1);  //clear ICF1 by writing logic one
    TIFR = (1<<TOV1); //clear overflow flag
    TimerOverFlow = 0;

    while ((TIFR & (1 << ICF1)) == 0);/* Wait for falling edge */
    count = ICR1 + (65535 * TimerOverFlow); //take count
    distance = (double)count /466.47;


    return distance;
}
```

4.3.4Driver Safety Alert Using Buzzer

A buzzer is incorporated into the project to improve safety measures, allowing the system to send out timely alerts in the form of audible alarms if the driver is found to be dozing off.

An embedded system frequently uses buzzers to deliver auditory feedback or alerts. It is an electromechanical device that produces a certain tone or rapidly vibrates a diaphragm to produce sound. Buzzers are frequently employed in embedded systems for a variety of functions, such as signaling system status, delivering alarms or notifications, and producing sound effects. They are frequently managed by microcontrollers or other electrical parts, which enables programmers to set up particular sound durations, frequencies, and patterns. Buzzers are small, affordable, and simple to incorporate into embedded systems.

Buzzer consists of two pins one connected to ground and the other connected to PA3.

Buzzer utilizes DIO where buzzer pin is set as output and it operates when setting logical 1 to its pin (PA3).

```
DDRA |= (1<<3); //make buzzer pin output
```

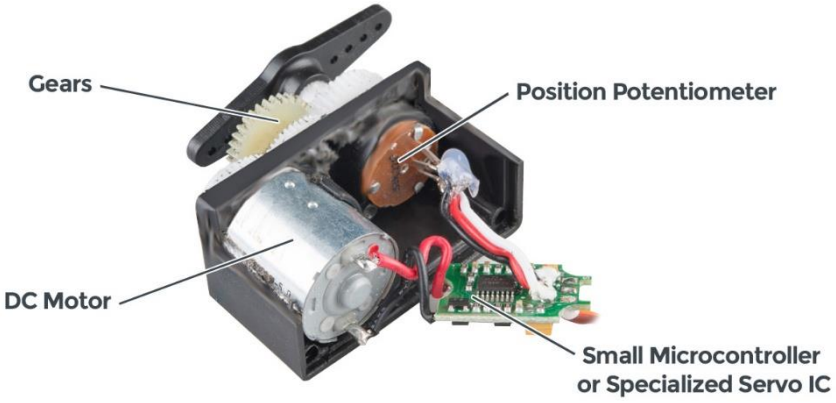
4.3.5Vehicle Side and Rear View Mirror adjustment for Blind spots

As we use servo motor in our The mirrors, side and back, will be adjustable in a way which follows the incline of the vehicle.

For example, if the vehicle goes down a steep incline, the back-mirror Automatically tilts and follows the same angle. Same situation goes for the round-about, sideways mirror get adjusted automatically to avoid most of the blind spots in each position of the vehicle and the Servo Motor will be responsible to move these mirrors

And the this how servo motor works Through the control wire, a pulse with a variable width, also known as pulse width modulation (PWM), is used to control servos. The minimum, maximum, and repetition rate of the pulses are all defined. Typically, a servo motor can only rotate 180°, or 90° in either direction.

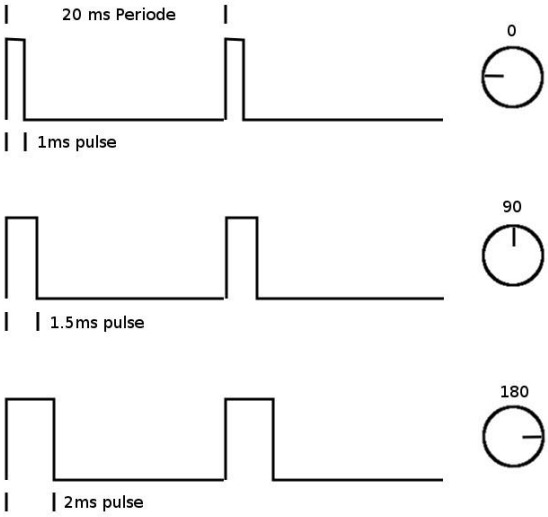
- Supply Voltage: 5V
- Torque: 2.5kg/cm
- Rotation: 0-180 degrees
- Gearbox
- Potentiometer
- DC Motor
- Controller



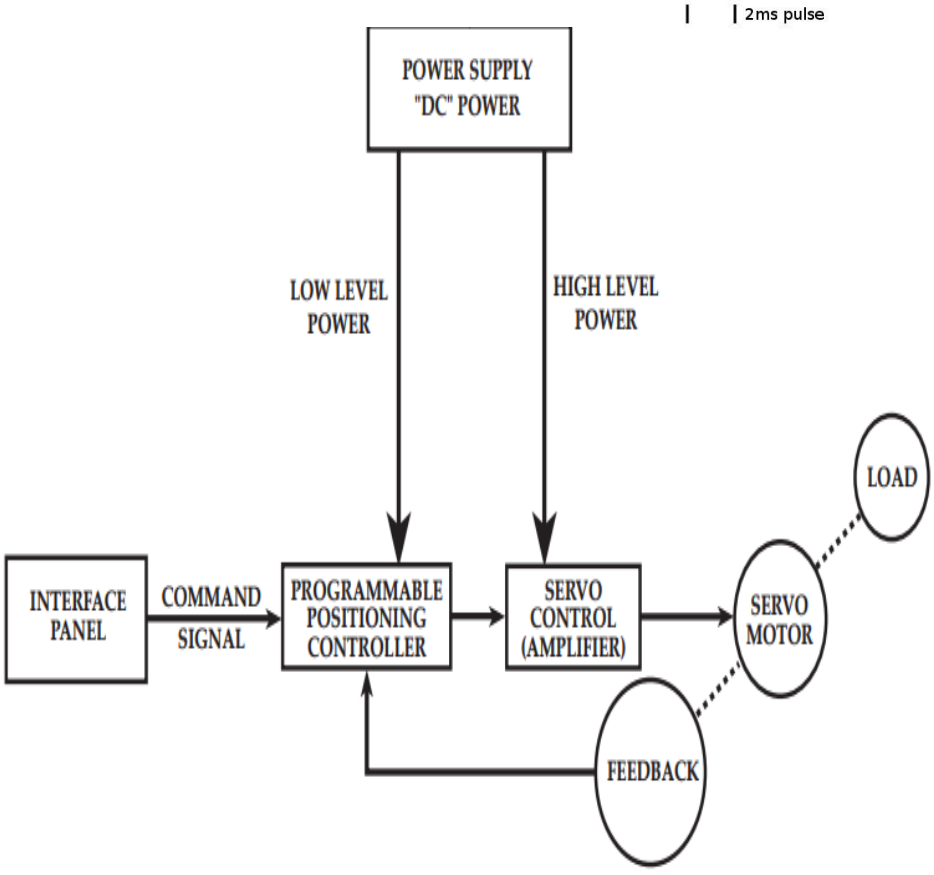
- Brown –ve supply (ground)
- Yellow Control (Signal, PWM) Wire
- Red +ve supply (Vcc)

Servo Motor Control

- A 5-volt pulse is sent to the motor through the control line.
- The pulse's duration determines the angle.
- A 1.5 ms pulse indicates a 90-degree position.
- Less than 1.5 ms indicates a closer angle to 0°. Longer equals a position nearer 180°



Servo Motor Block Diagram



Servo Motor Driver with ATMEGA 32

- To control a servo motor, a pulse width modulation waveform is generated using TIMER2

(TCCR2 = (1<<WGM20)|(1<<WGM21)|(1<<CS22)|(1<<COM21);)

- (1 << WGM20) sets the WGM20 bit to 1, indicating that the WGM mode is being configured.
- (1 << WGM21) sets the WGM21 bit to 1. This, in combination with WGM20, sets the WGM mode to 3, which corresponds to Fast PWM mode with a non-inverted output.
- (1 << CS22) sets the CS22 bit to 1. This sets the prescaler for Timer2 to 64. The prescaler determines the speed at which the timer counts.
- (1 << COM21) sets the COM21 bit to 1. This configures the Timer2's Compare Match output mode, specifically setting it to non-inverting mode.

-Setting the direction of a specific pin in Port D to output mode as the control wire (yellow wire) of the servo motor will be connected with this pin

- DDRD|=(1<<PD7);

The FPWM Output Pin: PD7

-Setting the value of the Output

```
rotateServo (0);
rotateServo (90)
rotateServo (180);
```

```
int main()
{
```

```
    TCCR2 = (1<<WGM20)|(1<<WGM21)|(1<<CS22)|(1<<COM21);
    DDRD|=(1<<PD7);   //PWM Pins as Out

        while (1)
        {
                // Rotate the servo to 0 degrees
        rotateServo(0);
         //Rotate the servo to 90 degrees
                rotateServo(90);

                // Rotate the servo to 180 degrees
        rotateServo(180);

        }
        return 0;
}
```

4.3.6 Vehicle Orientation using MPU6050

The side and rear mirrors will be movable in a way that corresponds to the vehicle's incline.

For instance, if the car descends a steep hill, the rear-view mirror automatically follows the same angle and tilts and here is the IMU 6050 role it will get the right angle of inclination and rotation and gives this angle to the Servo Motor to rotate the side and back mirrors and then how the IMU 6050 works.

A comprehensive 6-axis motion tracking device is the MPU6050 sensor module. In a compact size, it includes a 3-axis gyroscope, 3-axis accelerometer, and a digital motion processor. Additionally, it incorporates an on-chip temperature sensor as an extra function. In order to connect with the microcontrollers, it has an I2C bus interface.

To interface with other sensor devices like a 3-axis magnetometer, a pressure sensor, etc., it features an auxiliary I2C bus.

A full 9-axis Motion Fusion output can be provided by MPU6050 if a 3-axis Magnetometer is attached to an additional I2C connection.

IMU Specifications

Gyroscope:

• A full-scale range of 250, 500, 1000, or 2000 degrees per second (dps) for 3-axis sensing

• Sensitivity values of 131, 65.5, 32.8, or 16.4 LSBs per dps

• A range of output data rates (ODR) from 8kHz to 1.25Hz

Accelerometer:

• A full-scale range of 2g, 4g, 8g, or 16g for 3-axis sensing

• ODR range of 8kHz to 1.25Hz;

• Sensitivity of 16384, 8192, 4096, or 2048 LSBs per g

•Temperature sensor: -40°C to +85° operating range; 340 LSBs per degree Celsius sensitivity

• Precision of 3°C

Supply Voltage:

•Operating voltage range of 2.375V to 3.46V for the MPU-6050, and 2.375V to 5.5V for the MPU-6050A

Communication Interface:

•I2C serial interface with a maximum clock frequency of 400kHz

•8-bit and 16-bit register access modes

Other Features:

•Digital Motion Processor (DMP) for complex motion processing

•On-chip 16-bit ADCs for accurate analog-to-digital conversion

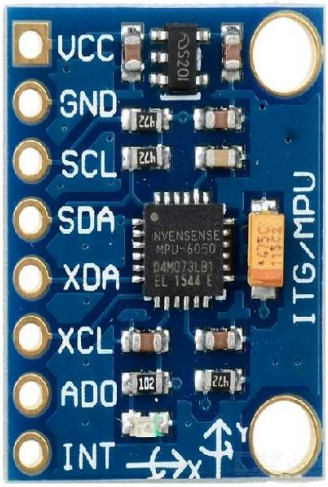
•Programmable digital filters for improved noise performance

•Interrupts for triggering events based on specific motion conditions

•Low-power consumption (3.9mA for full operation)

IMU 6050 Consist Of

- 3-Axis Gyroscope
- On-chip Temperature Sensor
- 3-Axis Accelerometer
- DMP (Digital Motion Processor)



MPU6050 Pin Description

The MPU-6050 module has 8 pins,

• INT: Interrupt digital output pin.

• AD0: I2C Slave Address LSB pin. This is 0th bit in 7-bit slave address of device. If connected to VCC then it is read as logic one and slave address changes.

• XCL: Auxiliary Serial Clock pin. This pin is used to connect other I2C interface enabled sensors SCL pin to MPU-6050.

• XDA: Auxiliary Serial Data pin. This pin is used to connect other I2C interface enabled sensors SDA pin to MPU-6050.

• SCL: Serial Clock pin. Connect this pin to microcontrollers SCL pin.

• SDA: Serial Data pin. Connect this pin to microcontrollers SDA pin.

• GND: Ground pin. Connect this pin to ground connection.

• VCC: Power supply pin. Connect this pin to +5V DC supply.

• MPU-6050 module has Slave address (When AD0 = 0, i.e. it is not connected to Vcc) as,

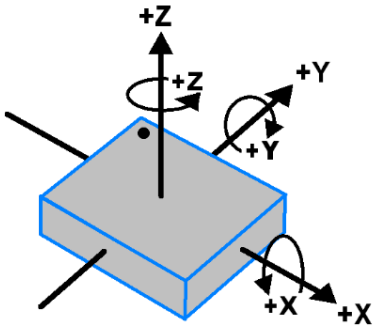
• Slave Write address(SLA+W): 0xD0

• Slave Read address(SLA+R): 0xD1

In our project scope we will use only the gyroscope sensor to get the inclination and rotation readings and then sends this reading to the AVR to move the SERVO Motor with the needed angles

IMU (Gyroscope) 6050 For Measuring Inclination and Rotation Angles:

Micro Electro Mechanical System (MEMS) technology is used in the MPU6050's 3-axis gyroscope. It is employed to determine the rotational speed along the X, Y, and Z axes.



MPU-6050
Orientation & Polarity of Rotation

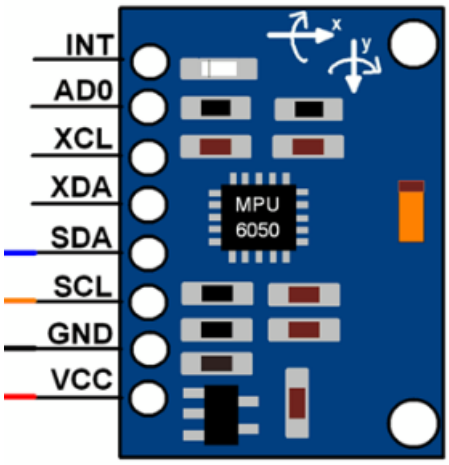
IMU (Gyroscope) Specifications

• A full-scale range of 250, 500, 1000, or 2000 degrees per second (dps) for 3-axis sensing

• Sensitivity values of 131, 65.5, 32.8, or 16.4 LSBs per dps

• A 8kHz to 1.25Hz output data rate (ODR) range

IMU (gyroscope) Wiring



- SDA Serial Data pin. Connect this pin to microcontrollers SDA pin.
- SCL **Serial** Clock pin. Connect this pin to microcontrollers SCL pin.
- VCC
- GROUND

IMU 6050 (Gyroscope) Driver with ATMEGA 32

Gyroscope sensor data from the MPU6050 module is calculated using raw 16-bit data in 2's complement form.

These ranges have been chosen by us:

- Sensitivity Scale Factor of 131 LSB (Count)/°/s and a gyroscope complete scale range of +/- 250 °/s

We must first do the 2's complement on the gyroscope sensor data in order to obtain the raw sensor data.

By dividing the raw data from the sensor by the sensitivity scale factor, we can calculate acceleration and angular velocity.

Angular velocity along the X axis = (Gyroscope X axis raw data/131) °/s.

Angular velocity along the Y axis = (Gyroscope Y axis raw data/131) °/s.

Angular velocity along the Z axis = (Gyroscope Z axis raw data/131) °/s.

(The code for the MPU6050 module is provided down below in appendix c)

4.3.7 Vehicle Positioning & Emergency Contact Using SIM808

What happens if the feared actually occurs? what if the driver really fell asleep, or lost consciousness? How does this ADAS unit deal with the situation? Under such circumstances, and when the time to react before a horrific incident occurs is too tight, it is never a shame to call for help. Yet how can the help arrive without knowing where they are headed? Exactly that is why an integrated GPS and GSM module is needed.

SIM808 is here to help get this job done. Instead of getting separate GPS & GSM modules, and using 2 communication protocols, one for each, this piece has it all! The module communicates with the ATmega32 MCU using the serial UART protocol.

The figure below shows the module and its different pins:

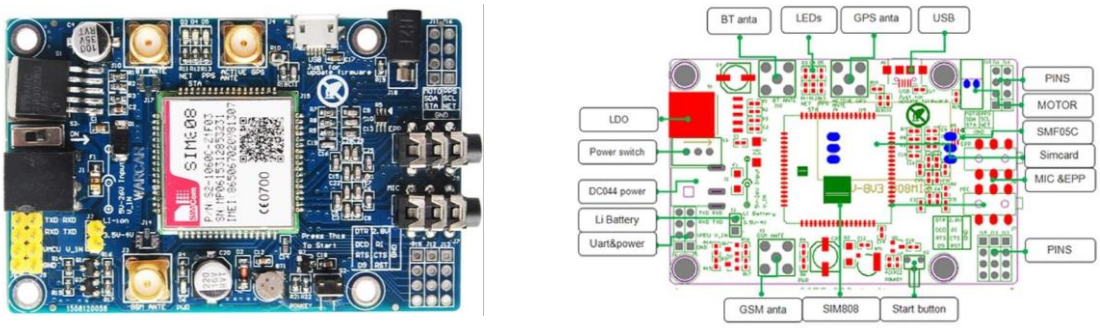


Figure 22 SIM808 Pin out

❋ Module Specifications:

- Quad-band 850/900/1800/1900MHz
- Baud Rate: From 1200bps to 115200bps
- Supports Control via AT commands
- Supports GPS NMEA format
- Supports SMS, video calls, voice calls…etc.
- Low power consumption
- Operation temperature:-40℃ ~85℃

More details about the module provided in its datasheet.

But what are GPS & GSM in the first place? And how do they work? The next part discusses each feature separately, how it works, and how to interface it with the MCU.

❋ GPS: Global Positioning System

The Global Positioning System is a space-based radio navigation system, which embraces a constellation of **31** well-spaced satellites that broadcast navigation signals. It also includes a network of ground, and satellite control stations used for monitoring and control. These satellites orbit the Earth at an altitude of approximately 11,000 miles providing users with accurate information on position, velocity, and time wherever they are in the world and in all weather conditions. The location accuracy lies between 100 to 10 meters for most gadgets. So, briefly, this system allows people with ground receivers (GPS Antennas) to locate their geographic location.

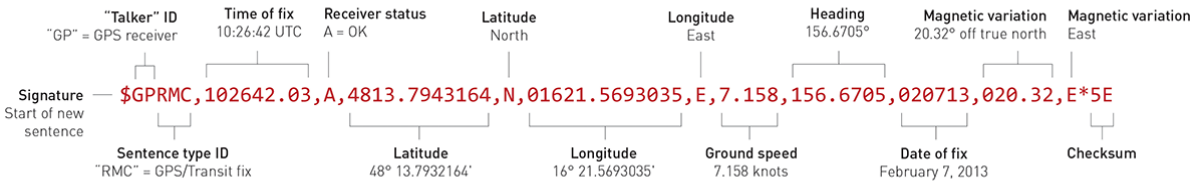
On the ground, any GPS receiver contains a computer that "triangulates" its position by acquiring bearings from three satellites. They have to be in its line of sight. The result is a geographic position that consists of longitude and latitude. If the user is moving, his receiver may also calculate his speed and direction of travel and provide him with estimated arrival times to specified destinations. What's surprising is that not only does the system provide users with the longitude & latitude, instead, it provides information about altitude, speed, date, and time in UTC (Universal Time Coordinates).

A GPS receiver (antenna) on earth measures the time it takes radio signals to travel from four or more satellites to its location, calculates the distance to each satellite, and from this calculation determines the user's position.

GPS is also known as **NAVSTAR** i.e. Navigation System with Time and Ranging.

The GPS module is connected to the MCU using a serial communication protocol. I provides its data in the National Marine Electronics Association (NMEA) format. This is a standard world wide format for GPS data so that they are understandable anywhere on earth. It holds different elements separated by commas, such as: Time, longitude, latitude, altitude...etc. Each string starts with "$" and ends with "*" followed by a two digit checksum representing a hexadecimal number that is the XOR of all the characters between the $ and *.

Structure of NMEA sentences:



There are different types of NMEA sentences based on the type of data they show:

GGA	GPS position data (latitude, longitude, number of satellites used, age of differential corrections, etc.)
GLL	Latitude and longitude data
GSA	GPS DOP and active satellite information
GST	GNSS pseudorange error statistics and position accuracy
GSV	GNSS satellites in view
RMC	Contains recommended minimum specific GNSS data (latitude, longitude, ground speed, navigational status, etc.)
RTCM3	Turn RTCM3 messages on or off
VTG	Course over ground and ground speed
ZDA	UTC time and date information

In most cases, CGRMC is the type of NMEA sentences extracted since it holds the most wanted data.

Note that: The GPS antenna should face the sky when you're trying to get location data. Additionally, it is important to check that the red LED, that indicates whether the module is receiving data or not, is continuously blinking every 1 second.

SIM808 supports GPS & GNSS (Global Navigation Satellite System) location services. Users can either use the NMEA format sentences or get the GSM to extract the data from the GPS antenna using AT commands which will be discussed down below.

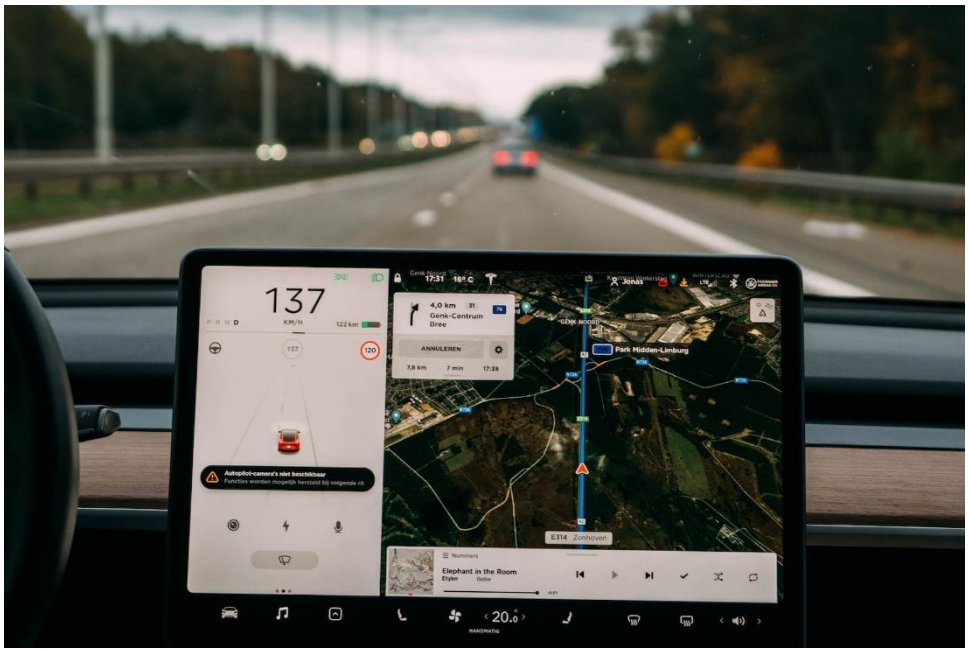


Figure 23 Global Positioning System

❋ GSM: Global System for Mobile Communication

It is a digital cellular technology used for mobile communication. It governs 2G networks by a set of standards and protocols. GSM provides voice & data services. The communication technology program operates on a wide geo area. Besides, it deploys digital radio channelling to develop audio, information, and multimedia communication systems. GSM, in addition to other technologies, has influenced the evolution of mobile wireless telecommunication services. This significant technology directs communication between mobile stations, base stations, and switching systems. GSM frequency bandwidths are divided into two paths: 900/1800 MHz and 850/1900 MHz Some geographical areas utilise 900/1800 MHz, others use 850/1900 MHz

The two communication techniques used by GSM are TDMA (Time Division Multiple Access), and FDMA (Frequency Division Multiple Access).

- **TDMA**: Same frequency for all users, yet, bandwidth divided into different time slots. One for every user.
- **FDMA**: Users assigned different frequencies, however, they all send their data at the same time.

In GSM, all geographical areas are grouped into hexagonal shaped cellular networks

Every user should have his own unique cell phone number, it is called the "Mobile Subscriber Identification Number". It is assigned to every mobile SIM card.

From the previously mentioned information, it can be deduced that a GSM module needs a SIM card in order to operate.

The data transmission rate ranges from 64 kbps to 120 Mbps.
GSM modems are addressed, and controlled using a series of AT commands. Stands for "Attention Commands" These are different instructions meant to attain different services from the modem. For example: SMS, MMS, voice calls, video calls...etc.

Such AT commands are exactly how a user contacts the SIM808 module. They can be used even to extract GPS data from the GPS antenna mounted atop the piece

- **SIM808 Interface with MCU:**

SIM808 is connected to the MCU through its serial interface port. In this project, the communication protocol used between them is, simply, the UART protocol.

- SIM808's Tx is connected to the MCU's Rx
- SIM808's Rx is connected to the MCU's Tx
- An adapter of 5v and 2 ampere power is plugged into its socket on the module. Note that the modem will not function properly unless the adapter is mounted.

- The start button beside the GPS antenna is pressed and held until the LEDS on the other side start blinking. This way, the antenna starts seeking satellite location information.
- A full size SIM card is mounted at the back of the module. This step has to be done in order for it to function correctly as well.

Next, the code is typed using AT commands & NMEA sentences, which are optional here since you can obtain location information from the GSM using AT commands as well.

The user manual, GPS manual, and datasheet of SIM808 hold all the needed information for the code.

❋ The following are the AT commands to interface GSM with the MCU:

Commands	**Descriptions**
AT	Make sure the module is working properly
AT+CSCA?	Get the core number of message
AT+CMGF=1	Select SMS message format
AT+CMGS="15124532672"	Set the message transmission number and send SMS message. After receiving the symbol >, the message Hello World!(*) can be sent out
SIM808 test	The context of message
1A	This is a terminator. Before sending it out, you should check the option Send As Hex

Figure 24 SIM808 User Manual GSM

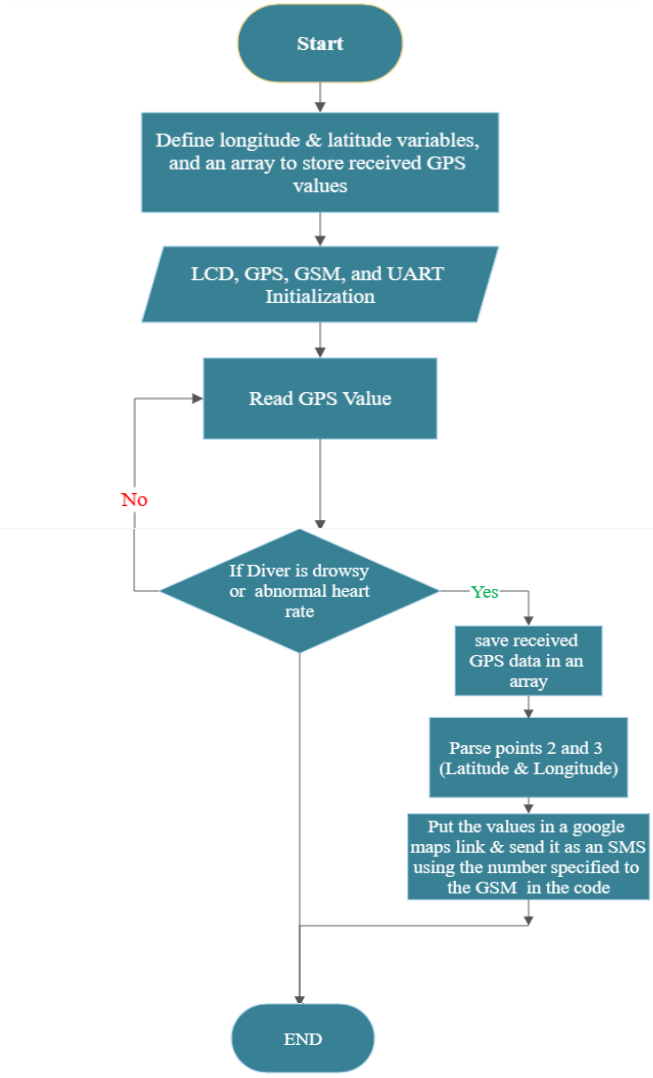
❋ The following are the AT commands to obtain location Information:

Commands	Descriptions
AT+CGPSPWR	GPS power control
AT+CGPSRST	GPS mode reset
AT+CGPSINF	Get current GPS location information
AT+CGPSOUT	GPS NMEA data output control Set to 255,
AT+CGPSSTATUS	Check GPS status

Figure 25 SIM808 GSM Location Information

Note that: In order to address GPS directly you can use the NMEA format in the code without the need for AT commands.

This is the flow chart of SIM808 module interface with ATmega32:



To wrap up, this chapter has discussed every used module, controller, sensor, and actuator in details. Their mechanism of action, how they communicate with the main microcontroller in the project, and even their interfacing codes. The chapter held the largest bulk of the project, and provided brilliant information to those who want to read.

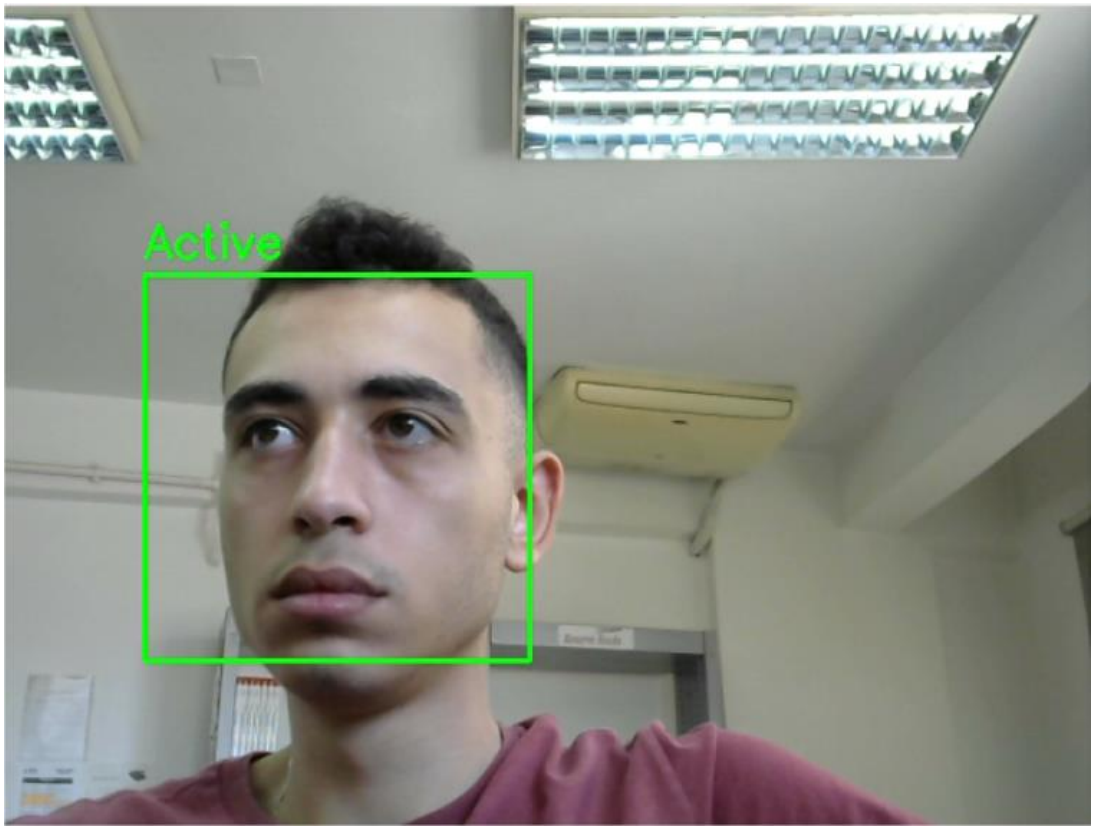
Chapter 5: System Integration

Prior to this chapter, each module has been thoroughly examined, and explained on its own. The features, the functionality, and how to interface it with ATmega32 MCU. In this chapter, every use case for the system is analysed and discussed, separately, providing its practical results before showing the whole integrated system on display.

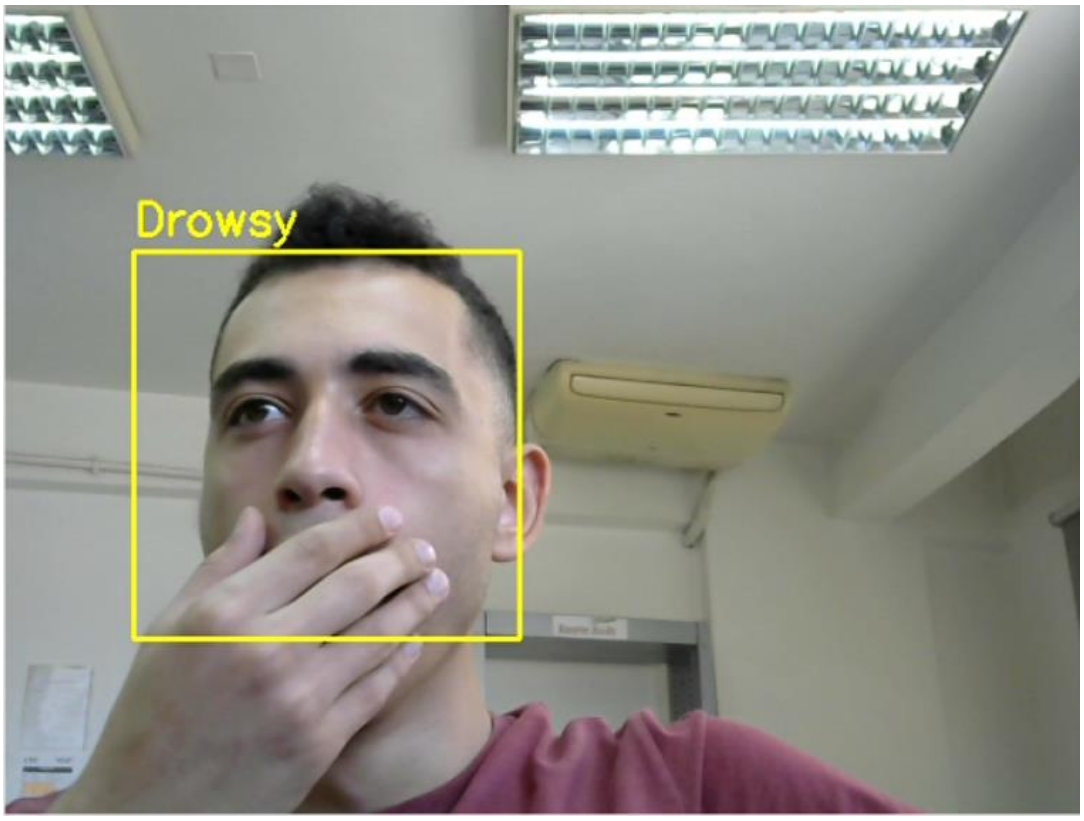
5.1 Use Case One: Fatigue Detection Using Deep Learning

This section considers the deployment of the deep learning fatigue detection model on Raspberry pi 4 using USB camera in order to make continuous predictions on driver's current alertness state.

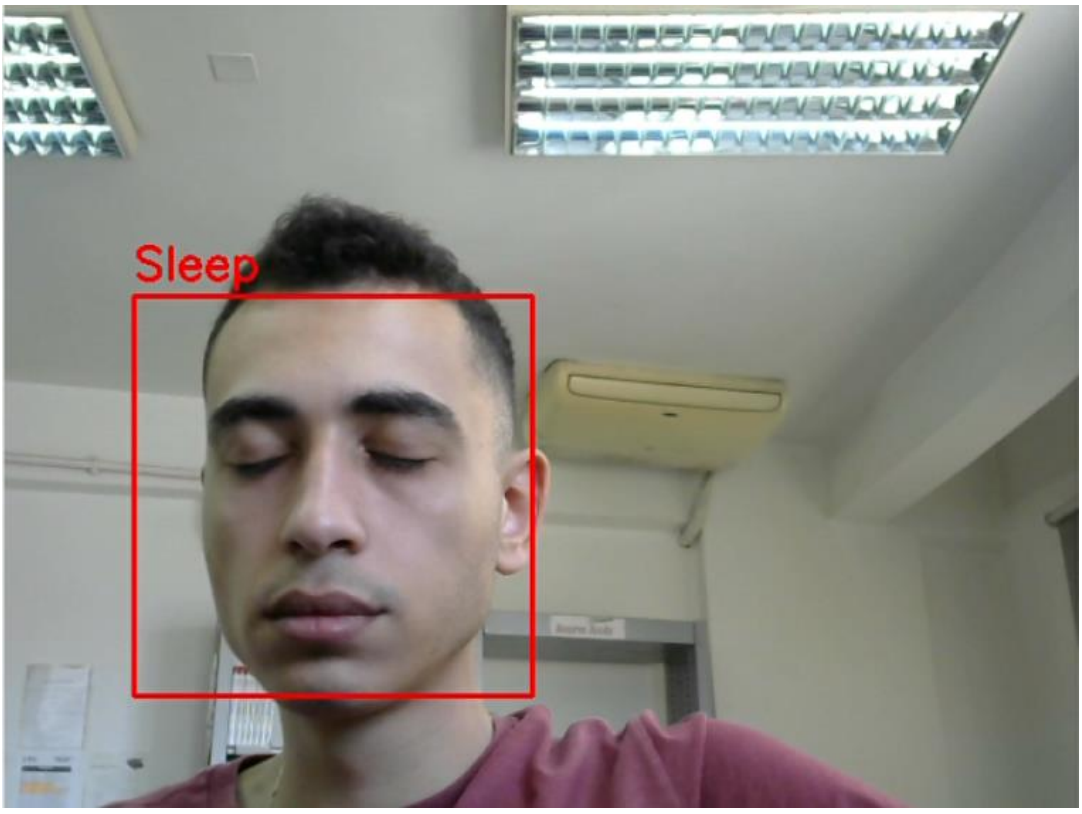
Case 1: Active state



Case 2: Drowsy state



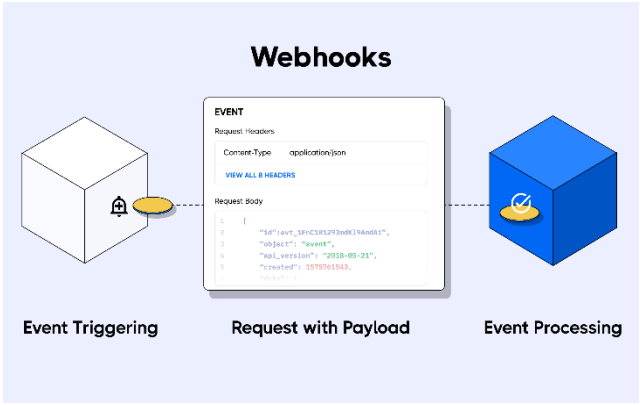
Case 3: Sleeping state



5.2 Use Case Two: Driver Asleep: Alerting Notification

This section considers sending an alerting notification to mobile phone using IFTTT to give an alert that the driver is currently sleeping.

- IFTTT stands for “If This Then That.” It’s a free web service that helps users automate web-based tasks.
- IFTTT application used to make an applet which autonomously indicates when webhook event is triggered then a notification is pushed through phone.
- HTTP (Webhook)
- Webhook is an event-driven call-back function that runs on the HTTP protocol.



- • In order to create a webhook, the client must give the server API a unique URL and the event it wants to be notified about. Instead of using the server's client pool once the webhook has been configured. Every time the specified event occurs, the server will send the relevant payload to the client's webhook URL. Webhooks are also referred to as push APIs or reverse APIs because they shift the responsibility of communication from the client to the server. As soon as the data is available, the server sends a single HTTP POST request to the client rather than the client sending HTTP requests and waiting for a response from the server.
- Using the Raspberry pi ‘s built-in WI-FI module. Triggering the Webhook upon specified condition was implemented by triggering the URL with special token key. Then Server make HTTP post request to the client which

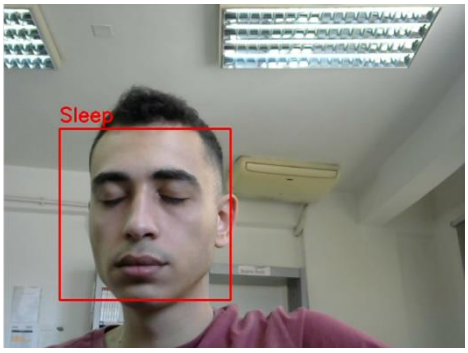
in our case is the IFTTT app. Then upon receiving the HTTP post request. Notification is sent to the phone based on the applet created.

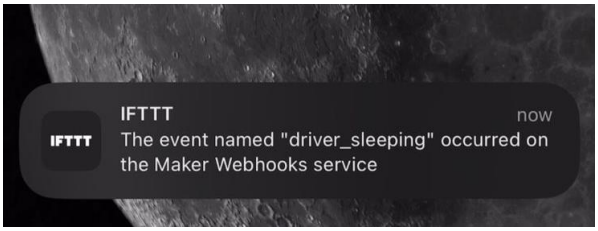
Raspberry pi code:

```
def send_notif():
    url = "https://maker.ifttt.com/trigger/driver_sleeping/with/key/d0ch0jiI0RGMYqW9yYm3mvk2cwSiLJD6z2NR9qQTB4e"
    value1 = "Sleeping" # Replace with your prediction variable
    payload = {"value1": value1}
    response = requests.post(url, data=payload)
    print(response.text)
```

Function used to send notification. The function is triggered upon driver detected sleeping.

Results:





5.3 Use Case Three: Intelligent Alert System for Drowsy or Sleeping Drivers

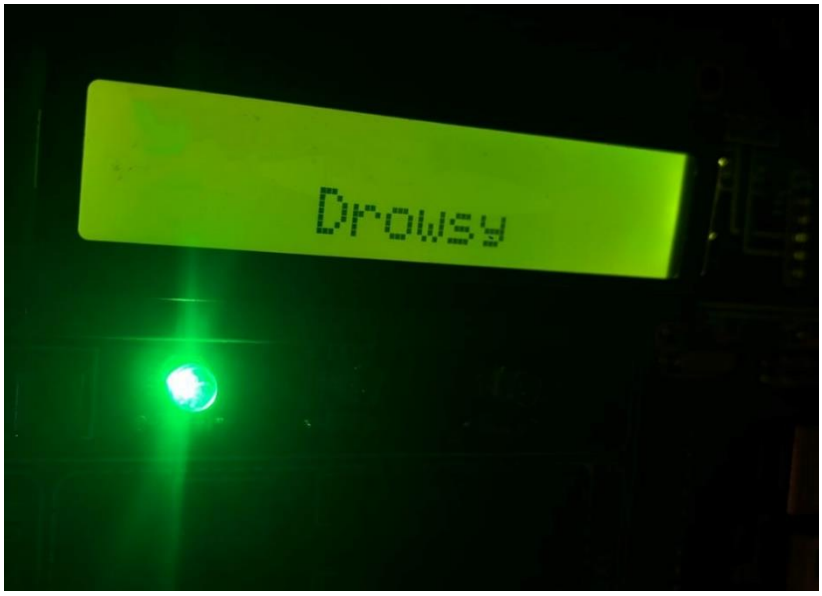
This section considers AVR ATmega32 microcontroller continuously receiving driver's current state from raspberry pi through Serial Peripheral Interface (SPI) communication protocol, and according to driver's alertness state. Actions should occur as follows:

Case 1: Active State

In case of active state: message displayed on LCD that that the driver is Active.

Case 2: Drowsy State

In case of drowsy state: message displayed on LCD that the driver is drowsy, and LED turns on.



Case 3: Sleeping State

In case of sleeping state: message displayed on LCD that the driver is sleeping, and buzzer is activated to alert the driver.

5.4 Use Case Four: Early Heart Attack Detection

This section considers the Integration of the heart rate sensor and the liquid crystal display (LCD) screen in an attempt to provide drivers with an accurate heart rate monitoring apparatus.

Upon integrating both codes in one main function code the system shall operate as follows: The pulse sensor shall continuously sense and calculate the driver's BPM providing real-time results. The results shall, always, be displayed on an LCD screen.

In this project, the implemented heart rate monitoring system has been compared to a commercial PPG pulse oximeter.

The results are shown below:

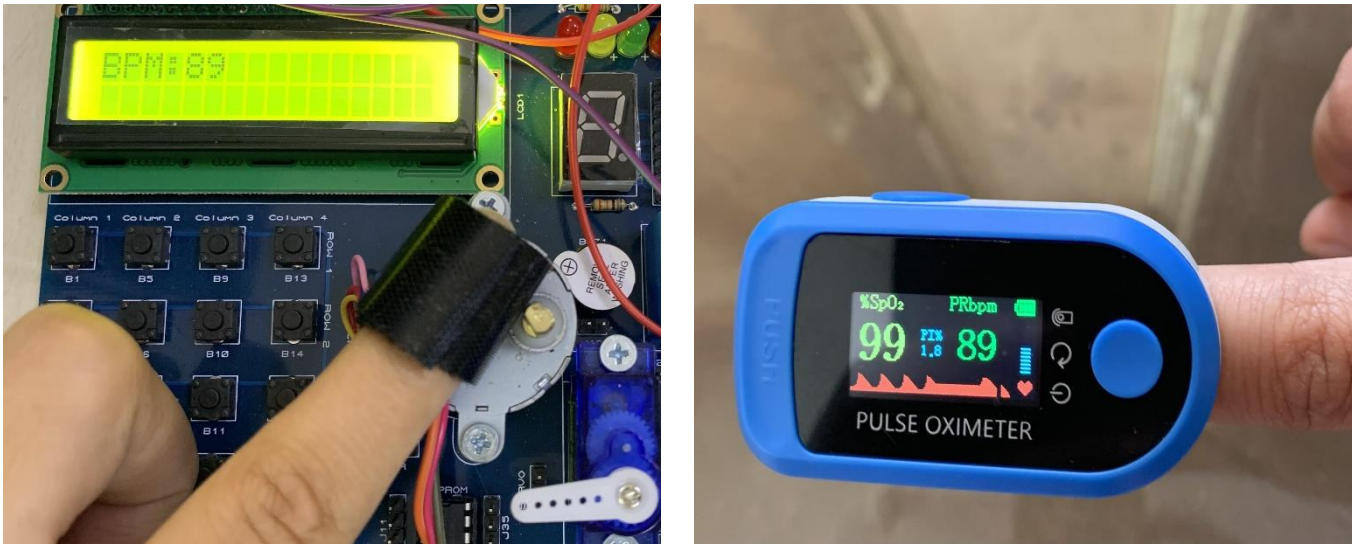


Figure 26 Heart Monitoring System Results

The results have been identical most of the time, however, sometimes they would differ in one or two beats. It is important to note that PPG pulse sensors are very sensitive to light or any surrounding noise, they can also give higher readings for

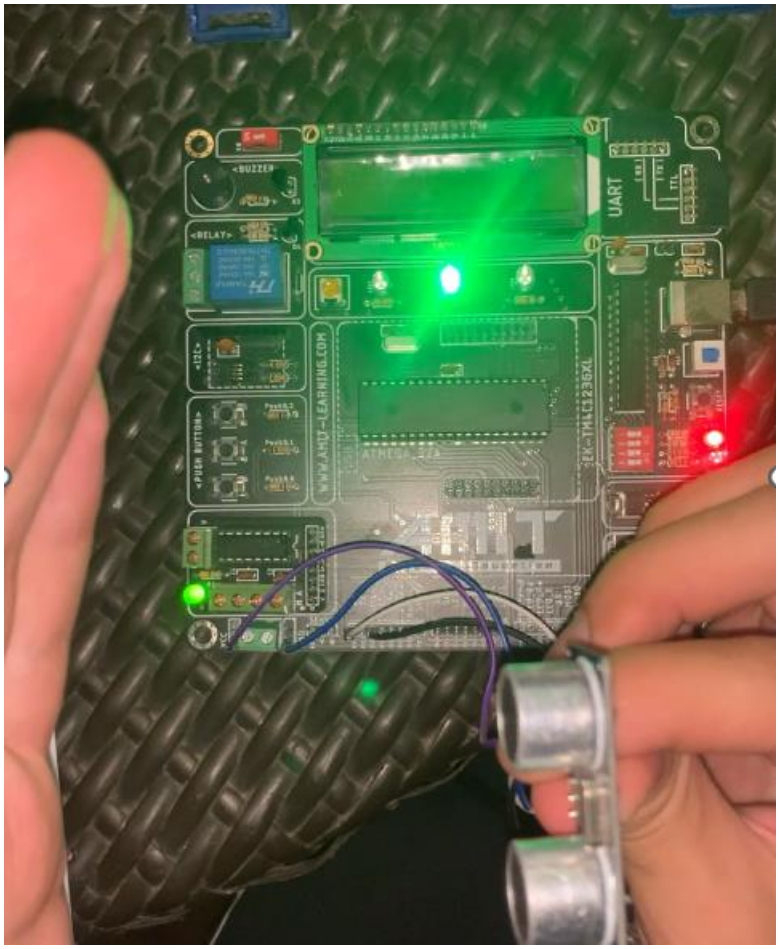
sweaty fingers, so it is important to stick the noise cancellation vinyl stickers and Velcro tape before any measuring approach.

5.5 Use Case Five: Adaptive Cruise Control

This section considers implementation of obstacle detection by using the ultrasonic sensor hc-sr04 and displaying on LCD the distance between the vehicle and an obstacle.

If there's an obstacle detected close to the vehicle, a LED is turned on.

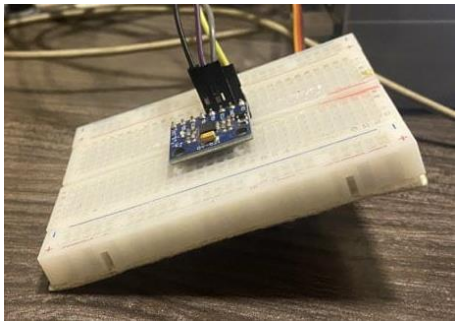
Results:

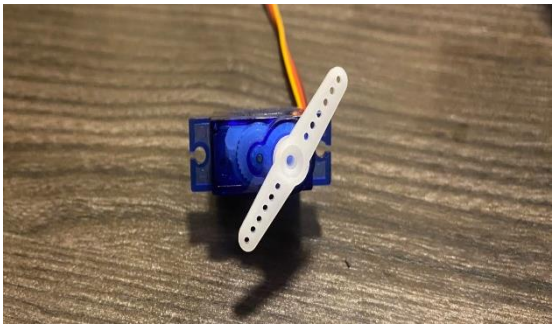


5.6 Use Case Six: Blind Spot Assistance

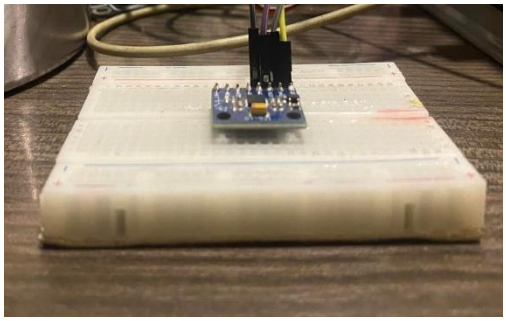
In this case we will consider the integration of the IMU 6050 (gyroscope) and the Micro Servo Motor (SG 90) first the IMU (gyroscope) will read the angles of inclination and rotation of the car and will give the Servo Motor these reading through the Communication Protocol I2C to the AVR so the Servo Motor to rotate the side and middle car mirrors the eliminate all the blind spots of the car.

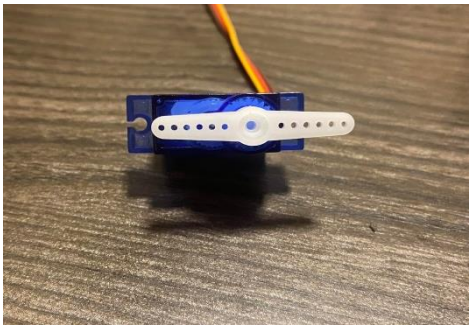
The First case if the car is moving Upwards the IMU (gyroscope) will give the servo motor +ve reading therefor the Servo Motor will move the mirrors with +90 degree upwards to eliminate all blind spots as the figures below.



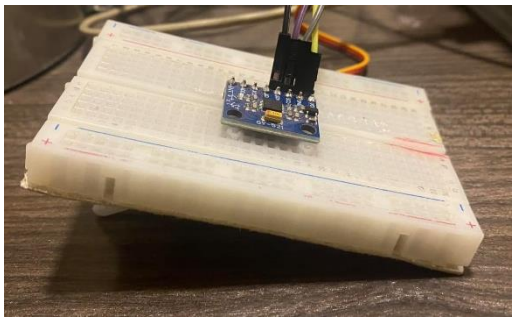


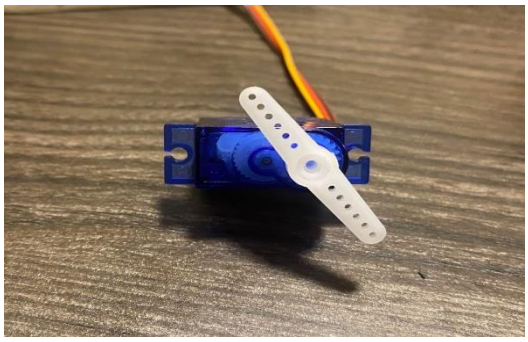
The second case if the car is moving with no angle of inclination the IMU (gyroscope) will give Servo Motor zero reading so the servo will remain at angle zero





Third case if the car is moving downward the IMU (gyroscope) will give the servo motor -ve reading therefor the Servo Motor will move the mirrors with -ve 90 degree downwards to eliminate all blind spots as the figures below.





Last but not least, in these three scenarios, all blind spots mirror spots will be eradicated when driving the vehicle by utilising the blind spot assistance on the car mirrors.

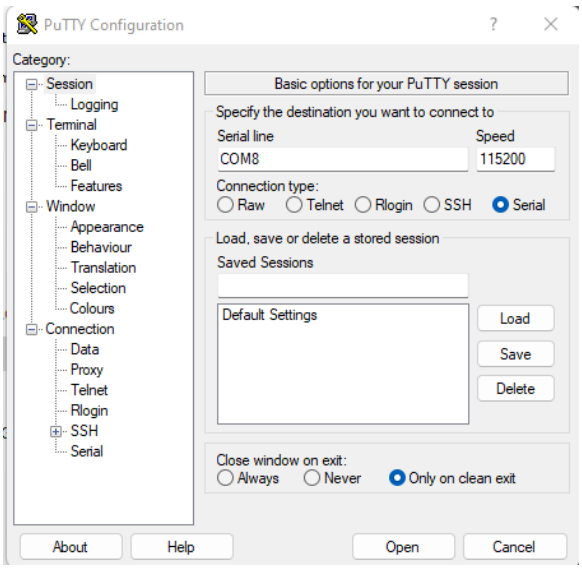
5.7 Use Case Seven: Location Identification and Emergency Contact

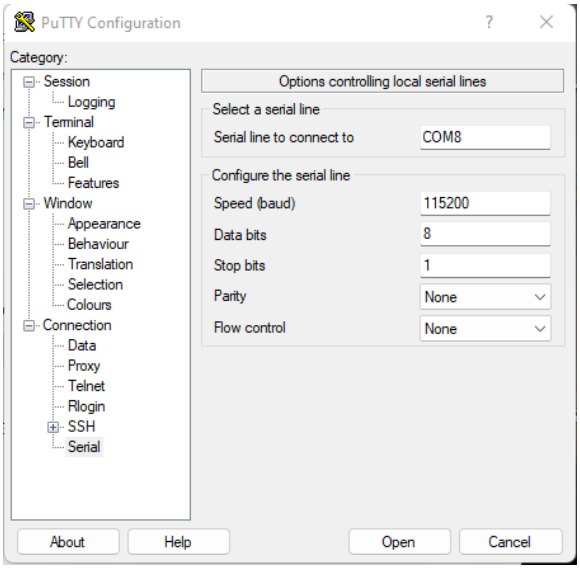
This is the part where the system actually responds to the driver's mistakes. It shows different approaches to display the GPS location data. First, it is tested on a hyper terminal (PUTTY) using a USB to TTL converter. The longitude and latitude data are in the form of "decimal-minutes" so they have to be transformed using a GPS format transformation tool. In our case, the used website is called "EDDmaps.org" then the location data extracted from the website and plugged into a Google maps link to show the results.

- **Hyper Terminal Approach:**

The user shall start the hyper terminal after running the PL2303 application on the laptop as an administrator, and connecting the USB to TTL converter's Rx to the module's Tx & vice versa. Do not forget the common ground! He must also check the ports section in his "Device Manager". To what port is the TTL converter connected?

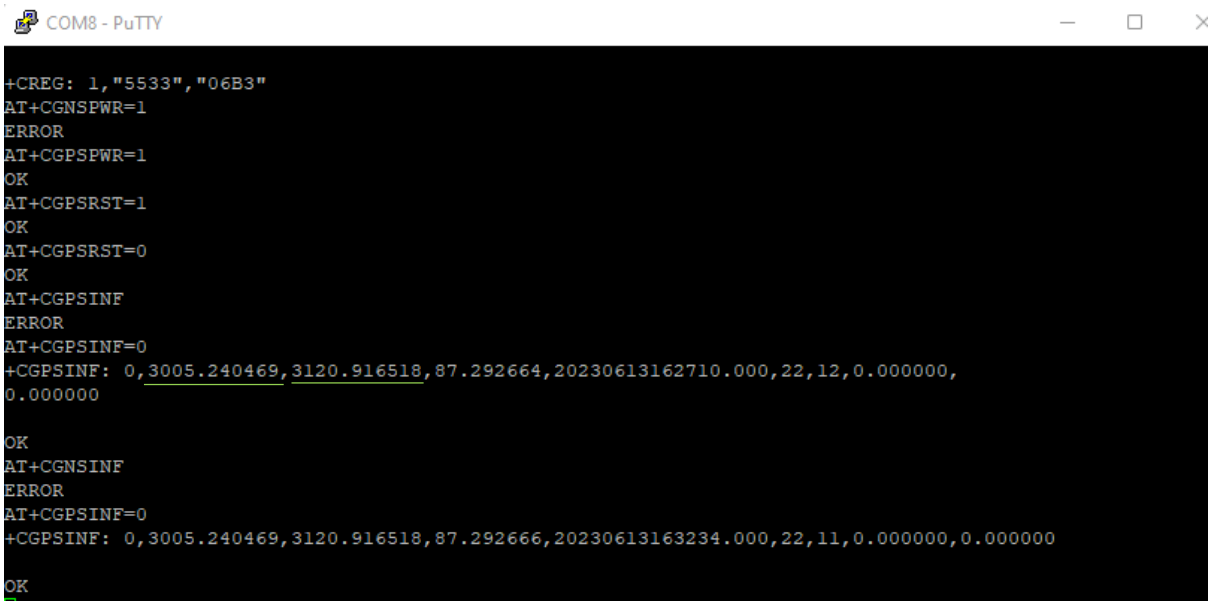
Upon starting the terminal, the following settings should be selected:



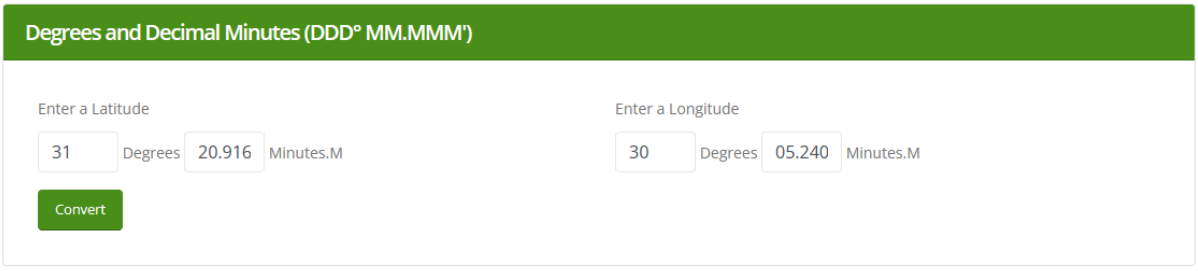


Next, you can start typing your commands and clicking “Enter” to see the results.

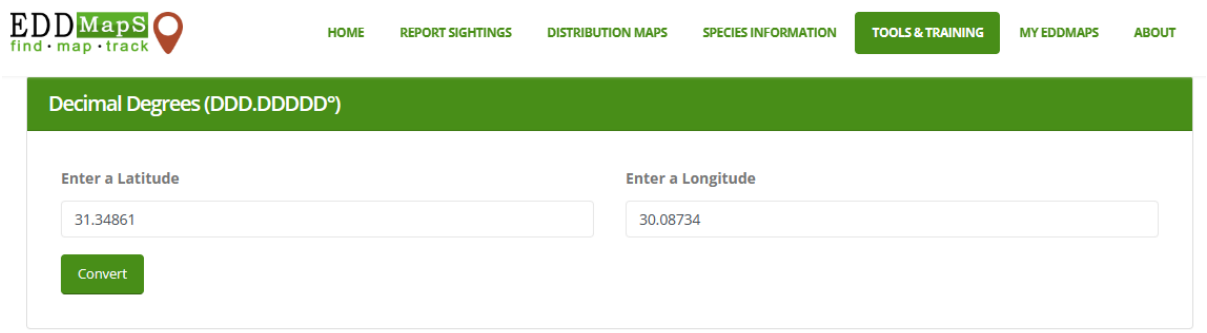
As shown in the figure down below, these are the location and time information provided by the GPS antenna:



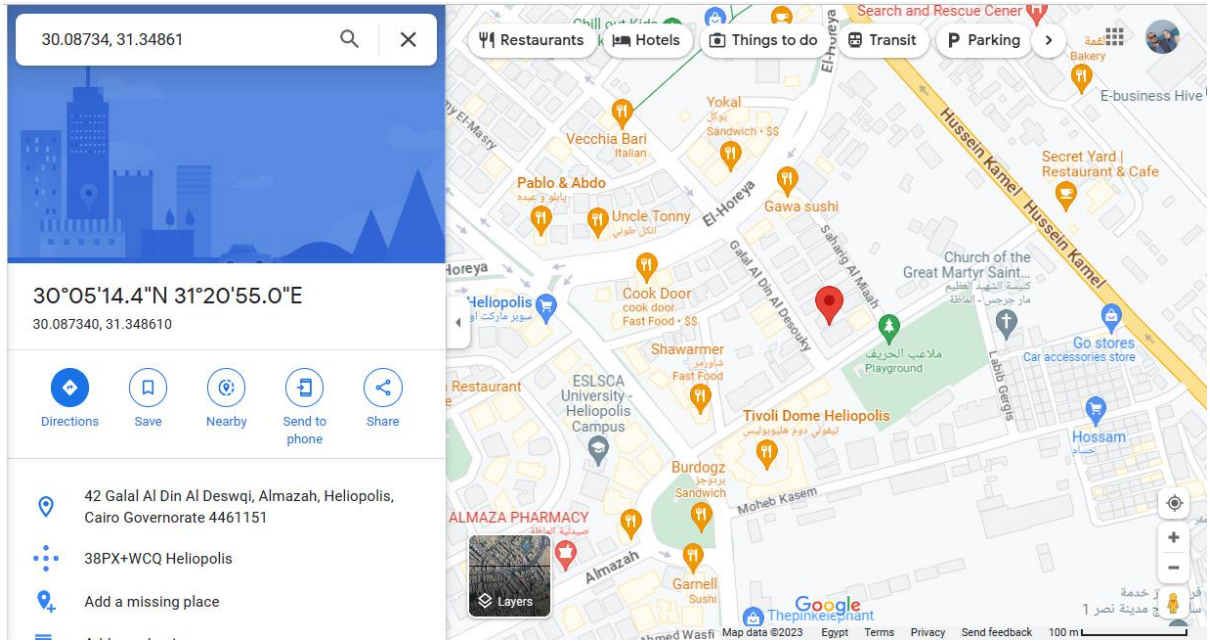
By typing the coordinates in decimal minutes to the website this way:



The desired coordinates in decimal degrees are displayed as shown below:



These coordinates are typed into Google maps to provide the location this way:



Conclusion

In the next few years, not only doctors will be the lifesaving “White Knights”. Engineers shall also have the honour for they are working day & night to develop powerful safety systems that are significantly changing the world for the better, and saving thousands of lives. Automotive embedded systems are significantly making a difference. The field is much needed worldwide nowadays and shall continue to grow for, in the end, our cars are crucial in our everyday lives.

With each new dawn, modern technologies come to life. Artificial intelligence and augmented reality, embedded systems and IOT, Cloud and edge computing, automation, machinery, robotics and many more to go. So, are we responsible enough to bring these diverse technologies to our benefit?

This project is an approach towards such goal. It is a step into the field of “Advanced Driver Assistance Systems” in order to help drivers on the road who are subject to many life-threatening risks. One uncontrollable threat is a horrendous heart attack another is sleep, and many more to those who count.

In this work, we have developed a system that deploys artificial intelligence in order to monitor a driver’s behaviour, and send a notification over Wi-Fi to alert his emergency contacts. Over and above that, we have studied embedded systems concepts and applied them using the ATmega32 MCU in order to detect a heart attack, or a cardiac arrest once it hits, and send an emergency message to the closest emergency unit before it is too late. Furthermore, the message provides the car’s exact location based on a constellation of 31 satellites above the earth. Finally, and to wrap this up, the system also yields blind spot assistance, and adjusts the mirrors accordingly. The project has been a huge step for us on the road of artificial intelligence and embedded systems. It has been an immensely enriching and beneficial journey, which we hope, will result in less car accidents in the near future.

References

[1] Character LCD Datasheet

lcd016n002bcfhet.pdf (vishay.com)

[2] Creating a pulse sensor from scratch group project by: Ramithuh et al. GitHub - ramithuh/Pulse-Sensor: A project carried out for the module EN1093

[3] An Evolution of Vehicle Technology. From steam driven vehicles to self driving electric vehicles by BBC. An evolution of vehicle technology (bbc.com)

[4] Pulse sensor playground toolbox. Pulse sensor walkthrough. PulseSensor Playground Toolbox – World Famous Electronics llc.

[5] EDD maps Latitude-Longitude converter. Latitude/Longitude Converter - EDDMapS

[6] SIM808 user manual V1.2. Microsoft Word - SIM808 V2.2.5 user manual V1.0.docx (robu.in)

[7] SIM808 hardware design V1.00. SIM808_Hardware+Design_V1.00.pdf (adafruit.com)

[8] SIM808 GPS application note V1.00. SIM808_GPS_Application_Note_V1.00.pdf (adafruit.com)

[9] Arafa Mycrosis project master "Pulse Sensor". Projects/PulseSensor at master · Arafa-microsys/Projects · GitHub

[10] What is GSM (Global System For Mobile Communication)? Meaning, working, architecture, and applications by: Spice Works. GSM Working, Architecture, Applications (spiceworks.com)

[11] Satellite Navigation- Global Positioning System by: Federal Aviation Administration. Satellite Navigation - Global Positioning System (GPS) | Federal Aviation Administration (faa.gov)

[12] Ultrasonic sensor datasheet.
https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

[13] Raspberry Pi datasheet. https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

[14] Transfer learning using tensorflow.
https://www.tensorflow.org/tutorials/images/transfer_learning

[15] How to dropout layers to avoid over fitting by keras.
https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout

[16] Keras guide for transfer learning. https://keras.io/guides/transfer_learning/

[17]CNN on tensorflow guide. https://cnvrg.io/cnn-tensorflow/

[18] Modern convolutional neural network. https://www.d2l.ai/chapter_convolutional-modern/index.html

[19] Face detection using cascade classifier on python. https://www.geeksforgeeks.org/face-detection-using-cascade-classifier-using-opencv-python/

[20]Preprocessing facial images. https://medium.com/yottabytes/a-quick-guide-on-preprocessing-facial-images-for-neural-networks-using-opencv-in-python-47ee3438abd4

[21] RESNET50V2 Function. https://keras.io/api/applications/resnet/#resnet50v2-function

[22] How to program R-PI. https://raspberrypi-aa.github.io/session3/spi.html

Appendix

This section only provides the large system drivers that have given the previously shown results, and finally it shows the full main system code.

Appendix A: Pulse Sensor Code

main.c

```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/delay.h>
#include"stdtypes.h"
#include <stdbool.h>
#include "DIO.h"
#include "LCD.h"
#include "ADC_iterface.h"
#include "ADC_config.h"

#define false 0
#define true 1

volatile u16 signal; // To store ADC data
volatile u16 threshold = 550;   // To know which signal to count as a beat
u16 BPM = 0; // used to hold the pulse rate. Updated every 2 ms
volatile u16 IBI=600; // Inter-beat interval (The time between 2 beats) used to find instant moment of heart beat
volatile u8 pulse = false; // true when pulse wave is high, false when it's low
// these variables are volatile because they are used during the interrupt service routine
volatile QS = false; //false
volatile u16 rate[10]; //array to hold the last 10 IBI values
volatile u32 samplecounter = 0; // used to determine pulse timing
volatile u32 lastbeattime = 0; //used to find IBI
volatile u16 P = 512; //To find peak in pulse wave
volatile u16 T = 512; //Used to find trough in pulse wave
volatile u8 amp = 100; // to hold amplitude of pulse wave
volatile firstbeat = true; // Used to start from a reasonable beat (used to seed the rate array so we startup with reasonable BPM)
volatile secondbeat = false; // used to seed the rate array so we startup with reasonable BPM
u16 array[4] = {'B','P','M',':'};
int main(void)
{ // To set the pin on which the sensor will be connected to ADC port
DIO_SET_PIN_DIRECTION(PORTA, PIN5, INPUT);
DIO_SET_PIN_VALUE(PORTA, PIN5, ACTIVATE_PULLUP);
// To blink a LED with every heart beat
```

main.c

```c
// To blink a LED with every heart beat
DIO_SET_PIN_DIRECTION(PORTB, PIN0, OUTPUT);
ADC_vidinit(); // ADC initialisation
LCD_INIT(); // LCD initialisation
// Timer 0 setup to trigger an interrupt every 2 ms
//TCNT0 = 0; // Initialisation of counter register
TCCR0 |= (1<<WGM01)| (1<<CS02); //   | (1<<COM01) //  Go To CTC Mode & 256 prescaler
OCR0 = 0X7C; // Set the max no. of counts to 124 for 500 Hz sample rate
TIMSK |= (1<<OCIE0); // Enable Interrupt on match between TCNT0 Register & OCR0
SREG  |= (1<<7); // Enable Global Interrupt
    while (1)
    {
        // DIO_SET_PIN_VALUE(PORTB, PIN0, HIGH);
         if (QS == true) // A Heart beat Was Found
         {
              // BPM and IBI have been Determined
              // Quantified Self "QS" true when avr finds a heart beat
            for(u8 j=0; j<4; j++)
               {
               LCD_SEND_CHAR (array[j]);
               }
               LCD_PRINT_INTEGER (BPM); // Show BPM On LCD
             QS = false; // reset the Quantified Self flag for next time
     }

          _delay_ms (2000);  //  take a break



          LCD_Send_Command (clear_display);

          LCD_Send_Command (return_home);
    }
}
```

*main.c

```
ISR (TIMER0_COMP_vect)

{
SREG  |= (0<<7); // Disable global interrupt so nothing interrupts the BPM calculation
signal = ADC_GET_READING(5); // Get the now digital reading from the ADC (Signal coming from pulse sensor)
samplecounter +=2; // This counter keeps track of time (Interrupt triggered every 2 ms)
u16 N = samplecounter - lastbeattime; // Monitors the time since the last beat to avoid noise & Helps to find peak & trough of pulse wave

 if(signal < threshold && N > (IBI/5)*3) // Avoid Dicrotic noise by waiting 3/5 of the last IBI
{
    if(signal < T) // T = Trough
    {
        T = signal; // Keep track of the lowest point in pulse wave
    }
}

  if(signal > threshold && signal > P) // Threshold condition helps avoid noise
{
    P = signal; // keep track of the highest point of pulse wave
}
// Time to look for a heart beat
  if (N > 250) // Avoiding High Frequency Noise at low time value i.e. 0.25 seconds
{
    if ((signal > threshold) && (pulse == false) && (N > (IBI/5)*3)) // Signal already shifted to 3/5 of IBI
    {
        pulse = true; // Set the pulse flag upon sensing a pulse
        IBI = samplecounter - lastbeattime; // Measure the time between beats in ms
        lastbeattime = samplecounter; // keep track of time for next pulse

        if(secondbeat) // If this is the 2nd beat i.e If second beat = True (because we do not count the 1st beat)
        {
            secondbeat = false; // Clear the 2nd beat flag
            for(u8 i=0; i<=9; i++) // Seed the running total to get a realistic BPM at startup (seed = cause to begin or grow)
            {
```

*main.c

```
            for(u8 i=0; i<=9; i++) // Seed the running total to get a realistic BPM at startup (seed = cause to begin or grow)
            {
                rate[i] = IBI; // Get 10 IBI readings and save in "rate" array
            }
        }

        if(firstbeat) // If it's the first time we've found a beat
        {
            firstbeat = false; // Clear 1st beat flag
            secondbeat = true; // Set the 2nd beat flag
            SREG  |= (1<<7); // Enable global interrupt
            return; // Discard IBI value because it's unreliable still unstable
        }
        // Keep a running total of the last 10 IBI values
        u32 runningtotal = 0; // clear the running total variable
        for(u8 i=0; i<=8; i++)
        { // shift data in the rate array and:
            rate[i] = rate[i+1]; // Drop the oldest IBI value to make room for a new one
            runningtotal += rate[i]; // Add up the remaining 9 IBI values
        }
        rate[9] = IBI; // add the latest IBI to the rate array
        runningtotal += rate[9]; // add the latest IBI to the running total
        runningtotal /= 10; // average the last 10 IBI values
        BPM = 60000/runningtotal; // How many beats can fit into a minute ? that's BPM
        QS = true; // Set the Quantified Self flag. We've found a beat!!
    }
}
   if (signal < threshold && pulse == true) // When the values are going down i.e The beat is over
{
    pulse = false; // Return the pulse flag to 0 so we can do this again
    amp = P-T; // Get the amplitude of the pulse wave
    threshold = amp/2 + T; // set the threshold at 50% of the pulse wave
    P = threshold; // reset those for next time
    T = threshold;
}
    if (N > 2500) // If 2.5 seconds go by without a beat
{
threshold = 512; // set threshold to default
P = 512; // set peak default
T = 512; // set trough default
lastbeattime = samplecounter; // Bring the last beat time up to date
firstbeat = true; // set these to avoid noise (1st beat not counted to avoid noise)
secondbeat = false;
}
SREG  |= (1<<7); // Enable Global Interrupt

}
```

Appendix B: MPU6050 Code

```
#define F_CPU 16000000UL                                    /* Define CPU clock Frequency e.g. here its 8MHz */
#include <avr/io.h>                                        /* Include AVR std. library file */
#include <util/delay.h>                                    /* Include delay header file */
#include <inttypes.h>                                      /* Include integer type header file */
#include <stdlib.h>                                        /* Include standard library file */
#include <stdio.h>                                         /* Include standard library file */
#include "MPU6050_res_define.h"                            /* Include MPU6050 register define file */
#include "I2C_Master_H_file.h"                             /* Include I2C Master header file */
#include "DIO.h"
void MPU6050_Init()                                            /* Gyro initialization function */
{
    _delay_ms(150);                                            /* Power up time >100ms */
    I2C_Start_Wait(0xD0);                                      /* Start with device write address */
    I2C_Write(SMPLRT_DIV);                                     /* Write to sample rate register */
    I2C_Write(0x07);                                           /* 1KHz sample rate */
    I2C_Stop();

    I2C_Start_Wait(0xD0);
    I2C_Write(PWR_MGMT_1);                                     /* Write to power management register */
    I2C_Write(0x01);                                           /* X axis gyroscope reference frequency */
    I2C_Stop();

    I2C_Start_Wait(0xD0);
    I2C_Write(CONFIG);                                         /* Write to Configuration register */
    I2C_Write(0x00);                                           /* Fs = 8KHz */
    I2C_Stop();

    I2C_Start_Wait(0xD0);
    I2C_Write(GYRO_CONFIG);                                    /* Write to Gyro configuration register */
    I2C_Write(0x18);                                           /* Full scale range +/- 2000 degree/C */
    I2C_Stop();

    I2C_Start_Wait(0xD0);
    I2C_Write(INT_ENABLE);                                     /* Write to interrupt enable register */
    I2C_Write(0x01);
    I2C_Stop();
}
     void MPU_Start_Loc()
     {
         I2C_Start_Wait(0xD0);                             /* I2C start with device write address */
         I2C_Write(ACCEL_XOUT_H);                          /* Write start location address from where to read */
         I2C_Repeated_Start(0xD1);                         /* I2C start with device read address */
     }


    void Read_RawValue()
    {
        MPU_Start_Loc();                                   /* Read Gyro values */
        /*Acc_x = (((int)I2C_Read_Ack()<<8) | (int)I2C_Read_Ack());
        Acc_y = (((int)I2C_Read_Ack()<<8) | (int)I2C_Read_Ack());
        Acc_z = (((int)I2C_Read_Ack()<<8) | (int)I2C_Read_Ack());
        Temperature = (((int)I2C_Read_Ack()<<8) | (int)I2C_Read_Ack());
        */
        Gyro_x = (((int)I2C_Read_Ack()<<8) | (int)I2C_Read_Ack());
        Gyro_y = (((int)I2C_Read_Ack()<<8) | (int)I2C_Read_Ack());
        Gyro_z = (((int)I2C_Read_Ack()<<8) | (int)I2C_Read_Nack());
        I2C_Stop();
    }
```

```c
int main()
{
    char buffer[20], floatxg[10], floatyg[10], floatzg[10];
    float Xa,Ya,Za,t;
    float Xg=0,Yg=0,Zg=0;
    DIO_Init();

    LCD_Init();
    //USART_Init(9600);
    I2C_Init();                                 /* Initialize I2C */
    MPU6050_Init();                             /* Initialize MPU6050 */                        /* Initialize USART with 9600 baud rate */
```

```c
 while(1)
 {
     Read_RawValue();

     //Xa = Acc_x/16384.0;                               /* Divide raw value by sensitivity scale factor to get real values */
     //Ya = Acc_y/16384.0;
     //Za = Acc_z/16384.0;

     Xg = Gyro_x/16.4;
     Yg = Gyro_y/16.4;
     Zg = Gyro_z/16.4;

     //t = (Temperature/340.00)+36.53;                   /* Convert temperature in °/c using formula */
```

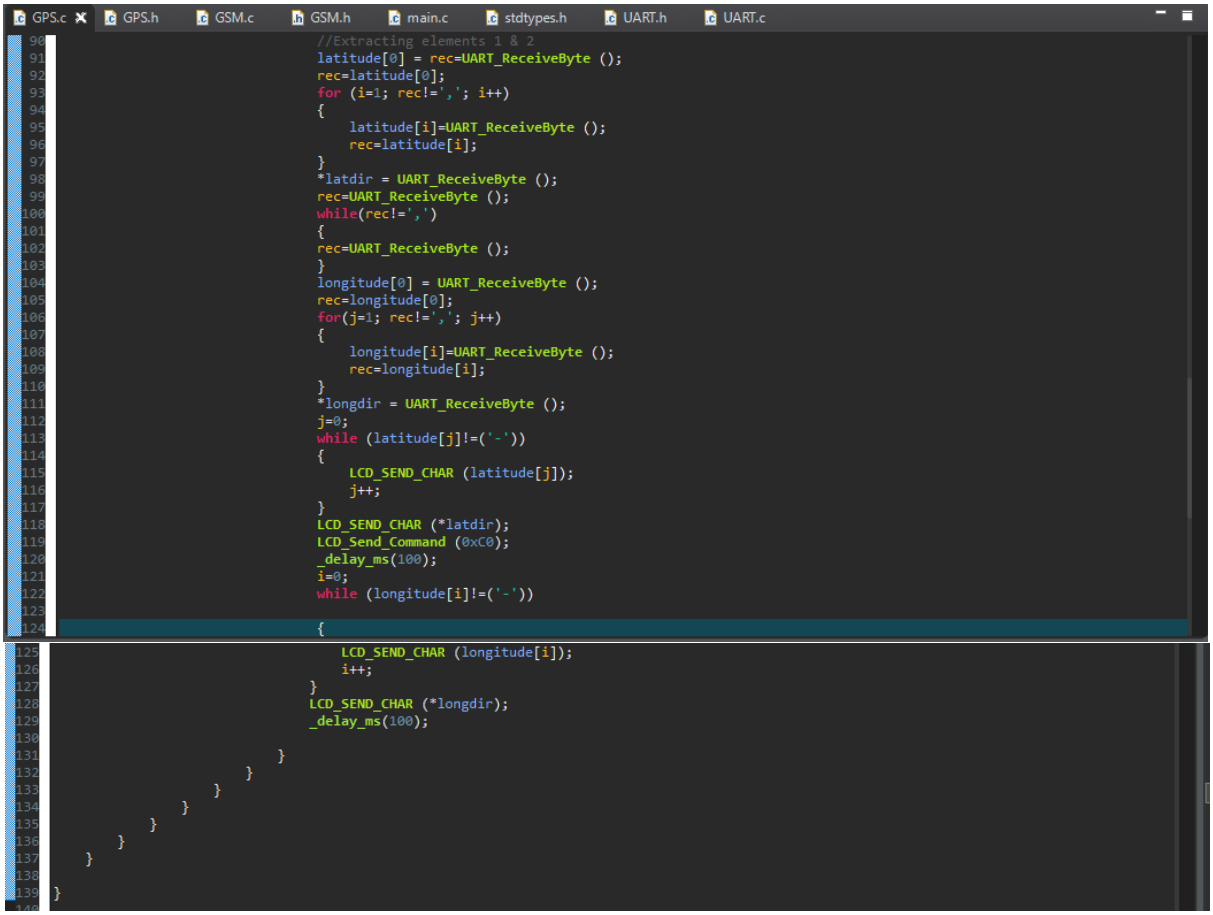
Appendix C: GPS Code

GPS.c | GPS.h | GSM.c | GSM.h | main.c | stdtypes.h | UART.h | UART.c

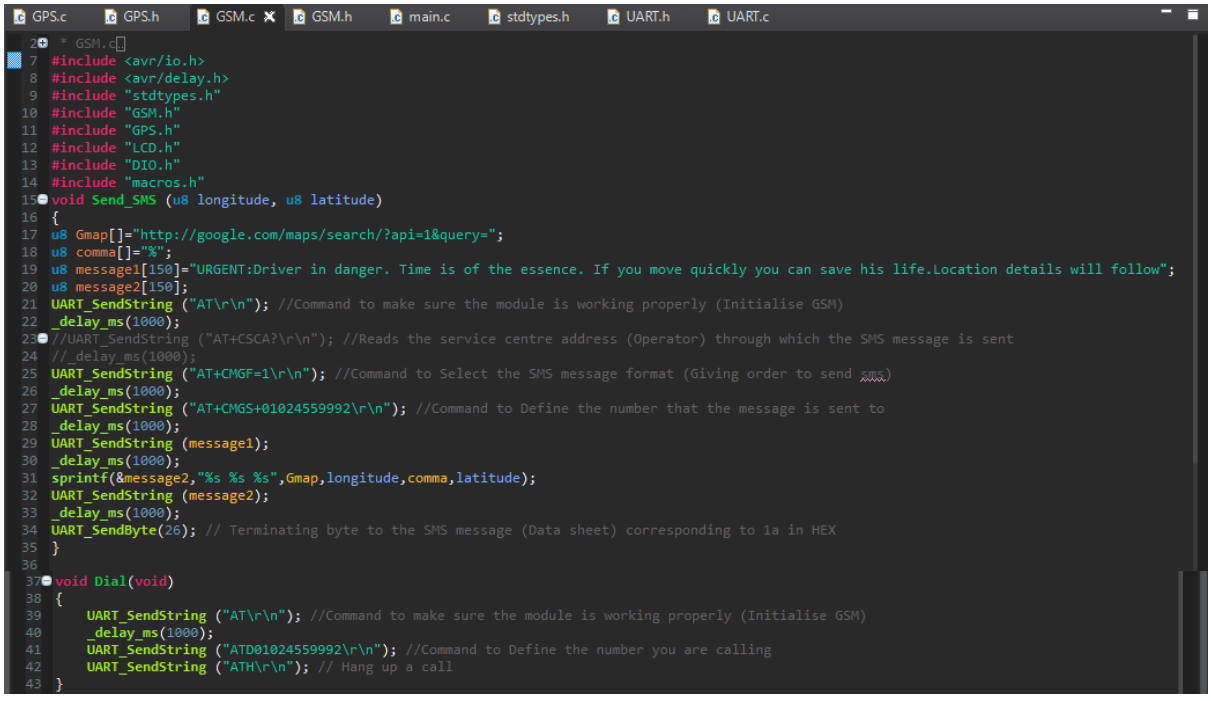
```c
/*
 * GPS.c
 */
#include<avr/io.h>
#include<string.h>
#include <util/delay.h>
#include "stdtypes.h"
#include "UART.h"
#include "GPS.h"
#include "LCD.h"
#include "DIO.h"
void GPS_INIT(void)
{
    UART_SendString("AT+CGPSPWR=1\r\n"); // Power on GPS system
    _delay_ms(100);
    UART_SendString("AT+CGPSRST=1\r\n"); // Set to cold start mode (Reset GPS Receiver)
    _delay_ms(100);
    //UART_SendString("AT+CGPS=1\r\n"); // Turn on GPS functionality (Sets GPS to connect to satellites)
    //_delay_ms(100);
    //UART_SendString("AT+GPSRD=5\r\n"); // Command that reads GPS data in NMEA format
    // delay_ms(100);
    UART_SendString("AT+CGPSINF=0\r\n");
    _delay_ms(100);
    UART_SendString("AT+CGPSOUT=255\r\n");
}
```

GPS.c | GPS.h | GSM.c | GSM.h | main.c | stdtypes.h | UART.h | UART.c

```c
void Read_GPS(u8 longitude[15],u8* longdir,u8 latitude[15],u8* latdir)
{ // Defining the variables needed for parsing NMEA sentence
    u8 rec; //Variable to store every received byte
    u8 i,j; //variable to know where we are in the NMEA sentence
    u8 parsevalue[50]; //for saving the split NMEA values to extract the needed values
    u8 *token; //every NMEA value extracted is put here before it's stored in parsevalue arr.
    //Extracting the RMC NMEA sentence from received values
    rec=UART_ReceiveByte ();
    if(rec=='$')
    {
        rec=UART_ReceiveByte ();
        if (rec=='G')
        {
            rec=UART_ReceiveByte ();
            if (rec=='P')
            {
                rec=UART_ReceiveByte ();
                if (rec=='G')
                {
                    rec=UART_ReceiveByte ();
                    if (rec=='L')
                    {
                        rec=UART_ReceiveByte ();
                        if (rec=='L')
                        {
                            rec=UART_ReceiveByte ();
                            if (rec==',')
                            {
                                rec=UART_ReceiveByte ();
                                while(rec!=',')
                                {
                                    rec=UART_ReceiveByte ();
                                }
                                //Parse the frame to get longitude & latitude only
                                //Extracting elements 1 & 2
```



Appendix D: GSM Code



Appendix E: Integrated System Embedded Code (Microchip Studio)

```
#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 16000000UL
#include <util/delay.h>
#include"STD_types.h"
#include <stdbool.h>
#include "spi.h"
#include "DIO.h"
#include "LCD.h"
#include "ADC_iterface.h"
#include "ADC_config.h"
#include "DIO_GFC.h"
#include "ultrasonic.h"

#include "MPU6050_res_define.h"
#include "I2C Master H file.h"
//ultrasonic begin
int TimerOverFlow =0;
//ultrasonic end
//pulse sensor begin
char BPMS[16];
volatile unsigned int signal; // To store ADC data
volatile unsigned int threshold = 550;   // To know which signal to count as a beat
unsigned int BPM = 0; // used to hold the pulse rate. Updated every 2 ms
volatile unsigned int IBI=600; // Inter-beat interval (The time between 2 beats) used to find instant moment of heart beat
volatile unsigned char pulse = false; // true when pulse wave is high, false when it's low
// these variables are volatile because they are used during the interrupt service routine
volatile QS = false; //false
volatile unsigned int rate[10]; //array to hold the last 10 IBI values
volatile unsigned long samplecounter = 0; // used to determine pulse timing
volatile unsigned long lastbeattime = 0; //used to find IBI
volatile unsigned int P = 512; //To find peak in pulse wave
volatile unsigned int T = 512; //Used to find trough in pulse wave
volatile unsigned amp = 100; // to hold amplitude of pulse wave
volatile firstbeat = true; // Used to start from a reasonable beat (used to seed the rate array so we startup with reasonable BPM)
volatile secondbeat = false; // used to seed the rate array so we startup with reasonable BPM
unsigned int array[4] = {'B','P','M',':'};
//pulse sensor end


//servoimu
#define SERVO_MIN_POS 0
#define SERVO_MAX_POS 255
#define DELAY_TIME_MS 500
float Gyro_x,Gyro_y,Gyro_z;
void MPU6050_Init()
                              /* Gyro initialization function */
{
    _delay_ms(150);                                  /* Power up time >100ms */
    I2C_Start_Wait(0xD0);                            /* Start with device write address */
    I2C_Write(SMPLRT_DIV);                           /* Write to sample rate register */
    I2C_Write(0x07);                                 /* 1KHz sample rate */
    I2C_Stop();

    I2C_Start_Wait(0xD0);
    I2C_Write(PWR_MGMT_1);                           /* Write to power management register */
    I2C_Write(0x01);                                 /* X axis gyroscope reference frequency */
    I2C_Stop();

    I2C_Start_Wait(0xD0);
    I2C_Write(CONFIG);                               /* Write to Configuration register */
    I2C_Write(0x00);                                 /* Fs = 8KHz */
    I2C_Stop();

    I2C_Start_Wait(0xD0);
    I2C_Write(GYRO_CONFIG);                          /* Write to Gyro configuration register */
    I2C_Write(0x18);                                 /* Full scale range +/- 2000 degree/C */
    I2C_Stop();

    I2C_Start_Wait(0xD0);
    I2C_Write(INT_ENABLE);                           /* Write to interrupt enable register */
    I2C_Write(0x01);
    I2C_Stop();
}
```

```
void MPU_Start_Loc()
{
    I2C_Start_Wait(0xD0);                           /* I2C start with device write address */
    I2C_Write(ACCEL_XOUT_H);                        /* Write start location address from where to read */
    I2C_Repeated_Start(0xD1);                       /* I2C start with device read address */
}
void Read_RawValue()
{
    MPU_Start_Loc();                                /* Read Gyro values */

    Gyro_x = (((int)I2C_Read_Ack()<<8) | (int)I2C_Read_Ack());
    Gyro_y = (((int)I2C_Read_Ack()<<8) | (int)I2C_Read_Ack());
    Gyro_z = (((int)I2C_Read_Ack()<<8) | (int)I2C_Read_Nack());
    I2C_Stop();
}
void rotateServo(uint8_t angle)
{
    // Calculate the position for the given angle
    uint16_t position = ((angle * (SERVO_MAX_POS - SERVO_MIN_POS)) / 180) + SERVO_MIN_POS;

    // Set the position of the servo motor
    OCR2 = (uint8_t)position;

    // Wait for the servo to reach the desired position
    _delay_ms(DELAY_TIME_MS);

    // Add a small delay to stabilize the servo position
    _delay_ms(10);
}
int main(void)
{
    //servoimu
    char buffer[20], floatxg[10], floatyg[10], floatzg[10];
    float Xa,Ya,Za,t;
    float Xg=0,Yg=0,Zg=0;

    I2C_Init();                                     /* Initialize I2C */
    MPU6050_Init();                                 /* Initialize MPU6050 */

    //Servoo
    TCCR2 = (1<<WGM20)|(1<<WGM21)|(1<<CS22)|(1<<COM21);
    DDRD|=(1<<PD7);   //PWM Pins as Out


    //Servoo ends


    //sservoimu
    //ultrasonic begin
    double usdistance;
    char string[10];
    DDRD |= (1<<5); //MAKE TRIG PIN O/P
    DDRD &=~ (1<<6); //MAKE ECHO PIN I/P
    PORTD |= (1<<6); //TURN ON PULL-UP

    TIMSK = (1<<TOIE1); //enable Timer1 ovf interrupts
    TCCR1A = 0; //set all bit to zero (normal operation)
    //ultrasonic end
```

```
/* Pulse sensor start */
DDRA &=~(1<<0);
PORTA |= (1<<0);
// To blink a LED with every heart beat
DIO_Init();
ADC_vidinit(); // ADC initialisation
LCD_Init(); // LCD initialisation
// Timer 0 setup to trigger an interrupt every 2 ms
TCNT0 = 0; // Initialisation of counter register
TCCR0 |= (1<<WGM01)| (1<<CS02); //  | (1<<COM01) //  Go To CTC Mode & 256 prescaler
OCR0 = 0X7C; // Set the max no. of counts to 124 for 500 Hz sample rate
TIMSK |= (1<<OCIE0); // Emanle Interrupt on match between TCNT0 Register & OCR0
sei(); // Enable Global Interrupt

/* Pulse sensor ends */

//spi begin
DDRC |= (1<<2); //make led pin output (LED0)
DDRC |= (1<<7); //make led pin output (LED1)
DDRD |= (1<<3); //make led pin output (LED2)
DDRB &=~ (1<<4);
DDRA |= (1<<3); //make buzzer pin output
SPI_Init(Slave);

SPI_SlaveEn(SPI_PortReg,SS);
LCD_String("Fatigue Detection System");
_delay_ms(1000);
LCD_Cmd(0x01);
//spi ends
while (1)
{
    unsigned char rxdata = SPI_Rx();

    //servoimu
    Read_RawValue();



    Xg = Gyro_x/16.4;
    Yg = Gyro_y/16.4;
    Zg = Gyro_z/16.4;



    if (Yg>100)
    {
        rotateServo(90);

    }
    else if (Yg<0 && Yg>-400)
    {
        rotateServo(0);


    }
    else if (Yg<0 && Yg<-400 )
    {
        rotateServo(90);

    }

    //sservoimu
    //ultrasonic begin
    ultrasonic_init();
    usdistance = ultrasonic_measure();
    dtostrf(usdistance, 2, 1, string);/* distance to string */
    strcat(string, " cm   ");   /* Concat unit i.e.cm */
```

```
    _delay_ms(200);

    if (usdistance <= 20)
    {
        PORTD ^= (1<<3);
    }
    else
    {
        PORTD &=~ (1<<3);
    }



    //ultrasonic ends


    /*pulse sensor begin */

    if (QS == true) // A Heart beat Was Found
    {
        // BPM and IBI have been Determined
        // Quantified Self "QS" true when avr finds a heart beat


        itoa(BPM,BPMS,10);
        LCD_String_xy (1,0,BPMS); // Show BPM On LCD
        QS = false; // reset the Quantified Self flag for next time

    }
          switch (rxdata)
          {

              case 3:
              LCD_Cmd(0x01);
              LCD_String_xy(0,0,"BPM:");
              LCD_String_xy(0,5,"vig:");
              LCD_String_xy(1,5,"Active");
              PORTA &=~ (1<<3);
              PORTC &=~ (1<<7); //TURN OFF LED1
              LCD_String_xy(0,9, string);
              break;
              case 4:
              LCD_Cmd(0x01);
              LCD_String_xy(0,0,"BPM:");
              LCD_String_xy(0,5,"vig:");
              LCD_String_xy(1,5,"Drowsy");
              PORTA &=~ (1<<3);
              PORTC ^= (1<<7); //TOGGLE LED1
              LCD_String_xy(0,9, string);
              break;
              case 5:
              LCD_Cmd(0x01);
              LCD_String_xy(1,0,"Switching to ACC");
              PORTA |= (1<<3); //TURN ON BUZZER
              PORTD |= (1<<3);
              PORTC &=~ (1<<7); //TURN OFF LED1
              break;
              default:
              break;
          }
          //spi ends

    }

}
```

```
ISR (TIMER0_COMP_vect)

{
    cli(); // Disable global interrupt so nothing interrupts the BPM calculation
    signal = ADC_GET_READING(5); // Get the now digital reading from the ADC (Signal coming from pulse sensor)
    samplecounter +=100; // This counter keeps track of time (Interrupt triggered every 2 ms)
    unsigned int N = samplecounter - lastbeattime; // Monitors the time since the last beat to avoid noise & Helps to find peak & trough of pulse wave

    if(signal < threshold && N > (IBI/5)*3) // Avoid Dichrotic noise by waiting 3/5 of the last IBI
    {
        if(signal < T) // T = Trough
        {
            T = signal; // Keep track of the lowest point in pulse wave
        }
    }

    if(signal > threshold && signal > P) // Threshold condition helps avoid noise
    {
        P = signal; // keep track of the highest point of pulse wave
    }
    // Time to look for a heart beat
    if (N > 250) // Avoiding High Frequency Noise at low time value i.e. 0.25 seconds
    {
        if ((signal > threshold) && (pulse == false) && (N > (IBI/5)*3)) // Signal already shifted to 3/5 of IBI
        {
            pulse = true; // Set the pulse flag upon sensing a pulse
            IBI = samplecounter - lastbeattime; // Measure the time between beats in ms
            lastbeattime = samplecounter; // keep track of time for next pulse

            if(secondbeat) // If this is the 2nd beat i.e If second beat = True (because we do not count the 1st beat)
            {
                secondbeat = false; // Clear the 2nd beat flag
                for(unsigned char i=0; i<=9; i++) // Seed the running total to get a realistic BPM at startup (seed = cause to begin or grow)
                {
                    rate[i] = IBI; // Get 10 IBI readings and save in "rate" array
                }
            }


               if(firstbeat) // If it's the first time we've found a beat
               {
                   firstbeat = false; // Clear 1st beat flag
                   secondbeat = true; // Set the 2nd beat flag
                   //sei(); // Enable global interrupt
                   return; // Discard IBI value because it's unreliable still unstable
               }
               // Keep a running total of the last 10 IBI values
               unsigned long runningtotal = 0; // clear the running total variable
               for(unsigned char i=0; i<=8; i++)
               { // shift data in the rate array and:
                   rate[i] = rate[i+1]; // Drop the oldest IBI value to make room for a new one
                   runningtotal += rate[i]; // Add up the remaining 9 IBI values
               }
               rate[9] = IBI; // add the latest IBI to the rate array
               runningtotal += rate[9]; // add the latest IBI to the running total
               runningtotal /= 10; // average the last 10 IBI values
               BPM = 60000/runningtotal; // How many beats can fit into a minute ? that's BPM
               QS = true; // Set the Quantified Self flag. We've found a beat!!
           }
       }
       if (signal < threshold && pulse == true) // When the values are going down i.e The beat is over
       {
           pulse = false; // Return the pulse flag to 0 so we can do this again
           amp = P-T; // Get the amplitude of the pulse wave
           threshold = amp/2 + T; // set the threshold at 50% of the pulse wave
           P = threshold; // reset those for next time
           T = threshold;
       }
       if (N > 2500) // If 2.5 seconds go by without a beat
       {
           threshold = 512; // set threshold to default
           P = 512; // set peak default
           T = 512; // set trough default
           lastbeattime = samplecounter; // Bring the last beat time up to date
           firstbeat = true; // set these to avoid noise (1st beat not counted to avoid noise)
           secondbeat = false;
       }

       sei(); // Enable Global Interrupt
}
```

```
ISR(TIMER1_OVF_vect)
{
    TimerOverFlow++;    /* Increment Timer Overflow count */
}
```

Appendix F: Integrated System R-Pi & SPI Code (Thonny IDE)

```
import requests
import cv2
import numpy as np
import tensorflow as tf
import time
import spidev



spi = spidev.SpiDev()
spi.open(0,0)
spi.max_speed_hz = 500

def send_notif():
    url = "https://maker.ifttt.com/trigger/driver_sleeping/with/key/d0ch0jiI0RGMYqW9yYm3mvk2cwSiLJD6z2NR9qQTB4e"
    value1 = "Sleeping" # Replace with your prediction variable
    payload = {"value1": value1}
    response = requests.post(url, data=payload)
    print(response.text)

interpreter = tf.lite.Interpreter(model_path="/home/youssef/Desktop/fatigue_detection/my_model.tflite")
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

face_cascade= cv2.CascadeClassifier('/home/youssef/Desktop/fatigue_detection/haarcascade_frontalface_default.xml')
cap = cv2.VideoCapture(0)
count = 0
x=0
y=0
w=0
h=0
prediction=""
fcolor=(0,0,0)
```

```
while True:
    # Read a frame from the webcam
    ret, frame = cap.read()

    # Increment the frame counter
    count += 1

    # Only process every 10th frame
    if count % 15 == 0:



        # Detect faces using the Haar Cascade classifier
        faces= face_cascade.detectMultiScale(frame, 1.3, 5)

        # Loop over all detected faces
        for (x, y, w, h) in faces:
            # Crop the face from the frame
            face = frame[y:y+h, x:x+w]

            # Resize the face to match the model input shape
            face = cv2.resize(face, (224, 224))
            face= np.array(face)
            face=np.expand_dims(face, axis = 0)
            face = np.float32(face/255.)
```

```
            # Perform model inference on the face
            interpreter.set_tensor(input_details[0]['index'], face)
            interpreter.invoke()
            result = interpreter.get_tensor(output_details[0]['index'])
            predicted_class = np.argmax(result)
            if predicted_class==0:
                txdata = 3
                prediction="Active"
                fcolor=(0,255,0)
            elif predicted_class==1:
                txdata = 4
                prediction="Drowsy"
                fcolor=(0,255,255)
            elif predicted_class==2:
                txdata = 5
                prediction="Sleep"
                fcolor=(0,0,255)
                send_notif()
```

```
            spi.xfer([int(txdata)])




    # Draw a rectangle around the detected face
    cv2.rectangle(frame, (x, y), (x+w, y+h), fcolor, 2)

            # Add the predicted label to the frame
    cv2.putText(frame, prediction, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, fcolor, 2)
    # Display the resulting frame
    cv2.imshow('frame', frame)

    # Exit the loop if esc is pressed
    if cv2.waitKey(1) ==27:
        break

# Release the webcam and close all windows
cap.release()
cv2.destroyAllWindows()
spi.close()
```