
## Faculty of Engineering

## Department of Electronics and Communication

# "Design and Implementation of a BCI-Based V2V System for Drowsiness Detection"

**Submitted by:**

| | |
|---|---|
| Yasmin Amr Ahmed Helmy | 2018/03452 |
| Hadeer Omar Ahmed Mahmoud | 2018/05604 |
| Youssef Ayman Sobhy Kelliny | 2017/06144 |

**Supervised by:**

***Prof. Alaa Hamdy***

Professor in Computer Science Department MIU

June, 2023

# Acknowledgements

This project has been a great learning experience for us, allowing us to delve into the fields of embedded systems and artificial intelligence. It has inspired us to pursue an embedded systems diploma, conduct thorough research, and develop essential skills such as critical thinking and problem-solving through various challenges. We would like to express our heartfelt gratitude to Dr. Alaa Hamdy for his guidance and support, which played a significant role in the successful completion of this project.

# Abstract

Drowsy driving is a widespread and dangerous problem that arises when individuals drive while feeling sleepy or fatigued. This state of drowsiness significantly increases the likelihood of accidents and tragic outcomes. Various factors contribute to drowsiness, including lack of sleep, untreated sleep disorders, medications, alcohol consumption, shift work, sleep apnea, and diabetic coma. Even if drivers manage to stay awake, drowsiness impairs their driving abilities by reducing attention, reaction time, and decision-making skills.

In this study, we present an innovative approach to address the problem of drowsy driving. Our approach involves simulating a Brain-Computer Interface (BCI) integrated with Vehicle-to-Vehicle (V2V) communication system.

To measure drowsiness, we obtained EEG signals from the BCI device and used an LSTM Bidirectional AI model to classify the level of drowsiness. This model demonstrates a promising accuracy rate of 78.042%.

The V2V system enables vehicles to exchange information wirelessly, enhancing situational awareness from a comprehensive 360-degree perspective. Our V2V system exchanges real-time information using an RF module, enabling timely warnings about potential hazards caused by drowsy drivers. These warnings include vital details such as the GPS location and speed of the drowsy driver, providing valuable insights into their proximity and the speed at which they are approaching your vehicle.

The results of this study demonstrate the feasibility and effectiveness of our integrated BCI-based V2V drowsiness detection system. With its impressive model accuracy of

78.042%, this system holds promise for real-world deployment and can play a crucial role in preventing accidents and saving lives.

# Table of Contents

# Acronyms or Abbreviations

| | |
|---|---|
| ADC | Analog to Digital Converter |
| AI | Artificial Intelligence |
| ASCII | American Standard Code for Information Interchange |
| ASIC | Application-Specific Integrated Circuit |
| AVR | Advanced Virtual RISC |
| BCI | Brain-Computer Interface |
| CMOS | Complementary Metal-Oxide-Semiconductor |
| COM | Communication |
| CSV | Coma Separated Values |
| DAC | Digital to Analog Converter |
| dBm | Deci Bell Milliwatt |
| DC | Direct Current |
| DDRx | Data Direction Register |
| DIO | Digital Input Output |
| EEG | Electroencephalograph |
| FP | Prefrontal |
| GSM | Global System for Mobile Communication |
| GHZ | Giga Heartz |
| GPIO | General Purpose Input/Output |
| GPS | Global Positioning System |
| I2C | Two wire interface |
| IC | Integrated Circuit |
| IDE | Integrated Development Environment |
| ISR | Interrupt Service Routine |
| LCD | Liquid-Crystal Display |

| | |
|---|---|
| LED | Light Emitting Diode |
| LSTM | Long Short-Term Memory |
| Mbps | Megabits per second |
| MHz | Mega Heartz |
| MISO | Master Input Slave Output |
| MOSI | Master Output Slave Input |
| NACK | Negative Acknowledgement |
| NMEA | National Marine Electronics Association |
| PC | Personal Computer |
| PINx | Input Register |
| PORTx | Output Register |
| PVT | Position, Velocity, Time |
| PWM | Pulse Width Modulation |
| RISC | Reduced Instruction Set Computer |
| RNN | Recurrent Neural Network |
| RPM | Revolution Per Minute |
| RPS | Revolution Per Second |
| RF | Radio Frequency |
| RTOS | Real-Time Operating System |
| RX | Receiver |
| RXC | Receiver Complete |
| SCK | Serial Clock |
| SPI | Serial Peripheral Interface |
| SS/CS | Slave Select / Chip Select |
| TGC | Think Gear Connector |
| TX | Transmitter |
| UDR | UART Data Register |

| | |
|---|---|
| uS | Microseconds |
| USART | Universal Serial Asynchronous Receiver Transmitter |
| USB | Universal Serial Bus |
| V2V | Vehicle to Vehicle Communication |
| Wi-Fi | Wireless Fidelity |

# List of Figures

# List of tables

# Chapter 1: Introduction

Chapter 1 provides an overview of the problem of drowsy driving and introduces the project's objectives of using a brain-computer interface (BCI) sensor and vehicle-to-vehicle communication (V2V) to address this issue.

## 1.1 Problem Statement

Drowsy driving is a major problem all around the world. The risk, danger, and often tragic results of drowsy driving are alarming. Drowsy driving is the dangerous combination of driving and sleepiness or fatigue. This usually happens when a driver has not slept enough, but it can also happen because of untreated sleep disorders, medications, drinking alcohol, shift work, sleep apnea, and diabetic coma. No one knows the exact moment when sleep comes over their body. Falling asleep at the wheel is clearly dangerous but being sleepy affects your ability to drive safely even if you don’t fall asleep.

Drowsiness:

- Makes you less able to pay attention to the road.
- Slows reaction time if you must brake or steer suddenly.
- Affects your ability to make good decisions.

The National Highway Traffic Safety Administration estimates that in 2017 drowsy driving was responsible for 91,000 crashes—resulting in 50,000 injuries and nearly 800 deaths. However, these numbers are underestimated, and up to 6,000 fatal crashes each year may be caused by drowsy drivers.

## 1.2 System Objectives

We chose the BCI system to detect drowsiness because it extracts EEG signals directly from the brain giving reliable information about the individual's brain activity. By using artificial intelligence, we can analyze the EEG data to identify patterns and features that are indicative of drowsiness. This will allow us to accurately detect drowsiness in real-time.

Our goal is to monitor driver drowsiness constantly. When drowsiness is detected, an alert buzzer will notify the driver. However, we recognize that relying only on the buzzer may not be enough, as drivers may not always notice or react quickly to the alert. To address this, we will implement a Vehicle-to-Vehicle (V2V) communication system to wirelessly notify other drivers to keep a safe distance from the drowsy driver. This proactive approach aims to prevent accidents. By combining the reliability of the V2V system with the BCI-based monitoring, we expect to significantly reduce global accident rates.

## 1.3 Motivation

The main reason for implementing a system that detects driver drowsiness and uses V2V communication is to overcome the limitations of relying solely on a driver's alertness to prevent accidents. When a driver is tired or sleepy, their ability to react quickly and appropriately to an approaching accident may be compromised. In such cases, depending only on the driver's alertness may not be enough to avoid collisions.

By utilizing V2V communication, we can expand the safety measures beyond the fatigued driver to include nearby vehicles. When the system detects drowsiness in the

driver, it sends wireless alerts to nearby vehicles, allowing them to take proactive steps to protect themselves and keep a safe distance. This collaborative approach ensures that even if the fatigued driver doesn't react promptly, the surrounding vehicles are informed about the potential danger and can adjust their driving accordingly.

## 1.4 Project Scope

- Embedded Systems
- Vehicle to Vehicle Communication
- Artificial Intelligence

## 1.5 Utilized Software

- Eclipse: Used for embedded C programming for V2V Communication.
- Python: Utilized for machine learning and data extraction from the BCI.
- Arduino IDE: Employed for serial communication between the PC and the BCI.

# Chapter 2: BCI and V2V Concept Generation

Chapter 2 explores the concepts of Brain-Computer Interface (BCI) and Vehicle-to-Vehicle (V2V) communication. It discusses the use of EEG headsets, such as the Neurosky Mindwave, to capture electrical impulses in the brain and detect mental states related to drowsiness. Additionally, it delves into the V2V system, which enables vehicles to exchange information wirelessly, improving awareness and safety on the road. The chapter highlights the potential of combining BCI and V2V technologies to enhance drowsy driving detection and prevent accidents.

## 2.1 EEG Headset

The brain contains approximately 100 billion linked neurons. For local networks of neurons and distant networks to synchronize with one another and with one another to coordinate complex behaviors, the brain employs rhythmic syncopation at various timescales. This implies that neurons interact more effectively with one another when they fire simultaneously as opposed to when they don't. The Electroencephalograph (EEG) is a tool used to capture the rhythmic synchrony of electrical impulses in the brain as microvolt oscillations on the scalp (white trace) (EEG).
Most of the frequency components in the EEG pattern that an electroencephalograph equipment record is below 30Hz. According to the EEG waveform (Brain Wave Generator, 2004), four mental states can be distinguished as shown in Fig.1.

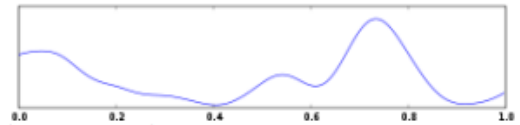
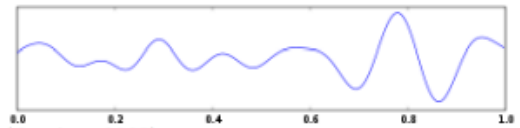
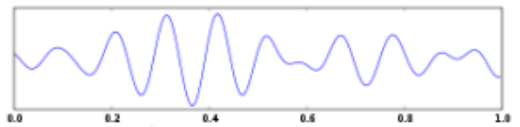
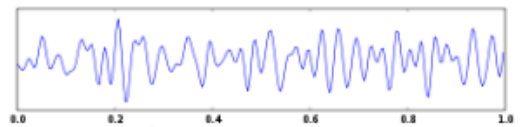
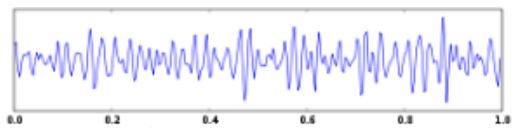
| State | Frequency Range | State of Mind | |
|---|---|---|---|
| Delta | 0.5Hz – 4Hz | Deep Sleep |  (Gamboa, 2005) |
| Theta | 4Hz – 8Hz | Drowsiness (1st stage of sleep) |  (Gamboa, 2005) |
| Alpha | 8Hz – 14Hz | Relaxed but alert |  (Gamboa, 2005) |
| Beta | 14Hz – 30Hz | Highly alert and focused |  (Gamboa, 2005) |
| Gamma | >35Hz | Peak performance |  (Gamboa, 2005) |

*Figure 1- EEG headset mental state frequencies*

The relationship between the frequencies and the state of the driver:

- Delta will detect the sleep
- Theta will detect the meditation
- Alpha will detect the relaxation
- Beta will detect the alertness
- Gamma will detect the focus

We will collect these frequencies and input them into our AI model to determine the correlation between drowsiness and these frequency patterns.

### 2.1.1 Neurosky Mindwave

After decades of research, neuroscientists have determined where specific activity occurs within the brain. For example, the motor control of limbs occurs at the top of the brain, vision is processed at the back of the brain. The pre-frontal cortex at the font of the brain is where higher thinking occurs i.e. Emotions, mental states, concentration, etc. this is the main reason why the main sensor of the MindWave is placed on a position known as FP1, on the forehead. This single sensor can measure multiple metal states simultaneously (NeuroSky, n.d.) such as attention, meditation, and blinking.

The official website for NeuroSky (NeuroSky, n.d.) says that an EEG sensor is like a microphone and that the physics of brainwaves and sound waves are almost identical. Furthermore, it is stated that all electrical appliances, such as computers, lightbulbs, wall sockets, etc., emit some degree of background "noise". Often, this noise is strong enough to mask brainwaves. Traditional EEG devices use a thick, conductive medical gel and a strictly noise-controlled atmosphere to boost the brainwave signal and guarantee there is no interference. According to NeuroSky, their gel-free headset can be used in noisy settings. This is done through:

1) Amplifying the raw brainwave signal.
2) Using filtering protocols to eliminate known noise frequencies such as muscle, pulse and electrical devices.
3) Using notch filters to eliminate electrical noise from the grid, which varies between 50Hz and 60Hz.
4) The headset comes with an ear clip which acts as a ground and reference. It allows NeuroSky's chip to filter out all of the electrical noise from the body and the ambient environment, and hone in on brainwaves.

## 2.2 V2V System

Vehicle-to-vehicle (V2V) communication enables vehicles to wirelessly exchange information about their speed, location, and heading. The technology behind V2V communication 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. Many of today's cars already come equipped with some advanced safety features, such as blind-spot detection, lane change assist and forward collision detection. These features rely on sensors such as radars, cameras, ultrasonic sensors and lidars to detect objects around the vehicle and either inform the driver of their presence or take actions to avoid collisions, such as automatically braking.

V2V enables another level of safety by allowing two moving vehicles to electronically communicate with each other (up to a range of about 300 meters), even if other objects are blocking line-of-sight. This ability to "see around corners" can be an important safety feature in a variety of common driving scenarios:

- Two vehicles approaching each other on a blind curve or a blind intersection
- Seeing around a large truck that is in front of the vehicle so it knows it can safely pass
- Instantaneously recognizing that cars ahead are suddenly braking in heavy traffic
- Spotting cars coming out of a driveway or a parking spot
- Alerting the driver that a vehicle up ahead has come to complete stop to make a left turn
- Improving awareness in adverse weather conditions
- Identifying nearby emergency vehicles and moving out of their way

Vehicles equipped with appropriate software (or safety applications) can use the messages from surrounding vehicles to determine potential crash threats as they develop. The technology can then employ visual, tactile, and audible alerts—or, a combination of these alerts—to warn drivers. These alerts allow drivers the ability to take action to avoid crashes.

These V2V communication messages have a range of more than 300 meters and can detect dangers obscured by traffic, terrain, or weather. V2V communication extends and enhances currently available crash avoidance systems that use radars and cameras to detect collision threats. This new technology doesn't just help drivers survive a crash—it helps them avoid the crash altogether.

# Chapter 3: Hardware Components and System Data Analysis

Chapter 3 provides an overview of the hardware components utilized in the project, along with a detailed outline of the proposed work for detecting drowsiness levels and enabling V2V communication. Additionally, the chapter presents the connection trials conducted to establish a link between the BCI and the microcontroller. Lastly, it concludes with a description of the data transmitted by the MindWave headset.

## 3.1 Hardware Components

1. **BCI Sensor (Neurosky Mind wave EEG):** The headset safely measures brainwave signals and monitors the attention and meditation levels of individuals as they interact with a variety of different apps. The device consists of a headset, an ear-clip, and a sensor arm. The headset's reference and ground electrodes are on the ear clip and the EEG electrode is on the sensor arm, resting on the forehead above the eye. It uses a single AAA battery with 8 hours of battery life. The sensor is shown in Fig.2.


*Figure 2- Neurosky Mindwave BCI*

2. **ATmega32 microcontroller:** The AVR microcontroller is based on the advanced Reduced Instruction Set Computer (RISC) architecture. ATmega32 microcontroller is a low power CMOS technology based controller. Due to RISC architecture AVR microcontroller can execute 1 million of instructions per second if cycle frequency is 1 MHz provided by crystal oscillator. The microcontroller is shown in Fig.3.

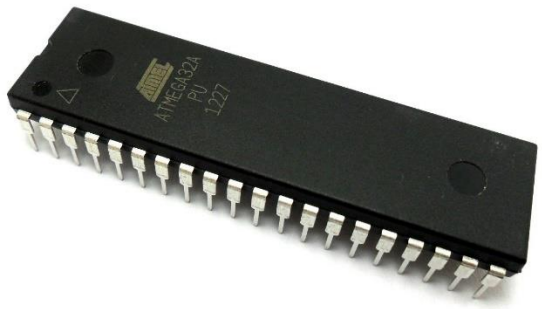



*Figure 3- Atmega32 Microcontroller*

We chose to use ATmega32 as our microcontroller as it has an extensive detailed datasheet with a wide community on the internet for help. It's cheap, fast, and easy to use. Furthermore, it has a board set of peripherals including:

- **GPIO** (General Purpose Input/Output)
- **ADC** (Analog to Digital Converter)
- **DAC** (Digital to Analog Converter)
- **Serial Communication**
- **I2C** (Two wire interface)
- **SPI** (Serial Peripheral Interface)
- **USART** (Universal Serial Asynchronous Receiver Transmitter)
- **Timers**
- **PWM** (Pulse Width Modulation)

The advantages ATmega32 has over Arduino Uno is that it has more pins and peripherals, therefore it is more useful for more complex projects such as V2V communication systems.

Note: STM32 would be a better option as it is more powerful than AVR in terms of processing power, peripherals, speed, and memory capacity. However, it is more complex, and the added functionalities would be more suitable for more complex applications than a V2V communication system, which is why we decided to use ATmega32 as the microcontroller for our system.

3. **Arduino** is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board. It will be used to interface with the BCI for simplicity. The microcontroller can be seen in Fig.4.

*Figure 4- Arduino Microcontroller*

4. **GPS Module:** will be used to detect the location of the vehicle. It uses USART communication to communicate with the controller. It can be used to calculate Latitude, and Longitude, and Altitude. The module comes with -161 dBm sensitivity patch antenna for receiving radio signals from GPS satellites. The Module can be seen in Fig.5.


*Figure 5- GPS Module*

**How GPS determines location:**

One of the global positioning system (GPS) devices utilizes data from satellites to locate a specific point on the Earth in a process named trilateration. Meanwhile, a GPS receiver measures the distances to satellites using radio signals to trilaterate. And trilateration is similar to triangulation, which measures angles. GPS modules contain tiny processors and antennas that directly receive data sent by satellites through dedicated RF frequencies. From there, it'll receive timestamp from each visible satellites, along with other pieces of data. If the module's antenna can spot 4 or more satellites, it's able to accurately calculate its position and time. The process of triangulation is briefly illustrated in Fig.6.

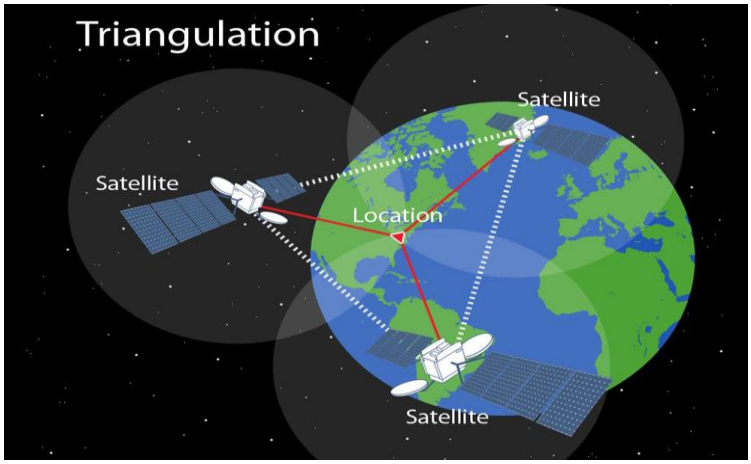



*Figure 6- GPS Triangulation Technique*

Once a location is determined, a red LED on the GPS device starts blinking. This indicates that a fix has been obtained.

5. **DC motor gears with wheel***:* will be used as a prototype for measuring the speed of the car. A visual representation of the modules are shown in Fig.7.

*Figure 7- DC Motor wheel with gear*

6. **DC motor driver*:*** Used for direction and speed control of DC motor. A visual representation of the module is shown in Fig.8.

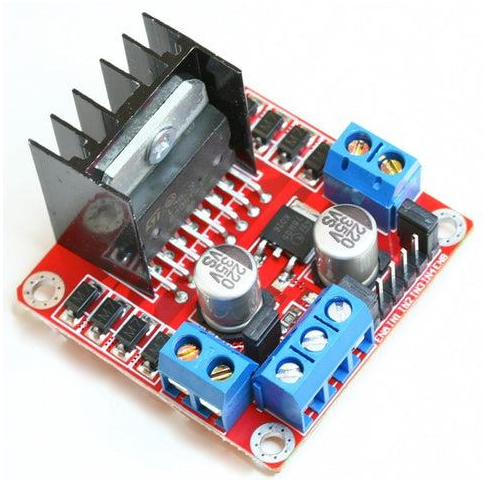


*Figure 8- DC Motor Driver*

7. **2-18650, 3.7 battery:** Used to provide 8V to the DC motor driver in order to power it, as 5V from the microcontroller is not enough.

8. **Potentiometer:** A potentiometer is a type of resistor that can adjust its resistance by moving a knob or lever. We will utilize a potentiometer to regulate wheel speed because it provides precise control over resistance, enabling us to effectively manage the power delivered to the wheels and achieve the desired speed. The module is shown below in Fig.9.

*Figure 9- Potentiometer*

9. **Photo Interrupter Module with Encoder Wheel:** A Photo interrupter module, along with the encoder wheel will be used to measure the speed of the wheel. The modules can be seen in Fig.10.


*Figure 10- Encoder wheel inserted between the U shaped sections of the Photo Interrupter Module*

10. **2- nRF24l01+ Modules:** are wireless transceiver modules that will be used as a short range communication method between the vehicles. The module is shown in Fig.11.

*Figure 11- nrf24l01+ Module*

11. **Buzzer:** A buzzer is an audio signaling device that will serve as an alert mechanism. Fig.12 displays the module.

*Figure 12- Buzzer*

12. **LCD:** is one kind of electronic display module. It is used in an extensive range of applications such as various circuits and devices which include mobile phones, calculators, computers, television sets, etc. The module is shown in Fig.13.

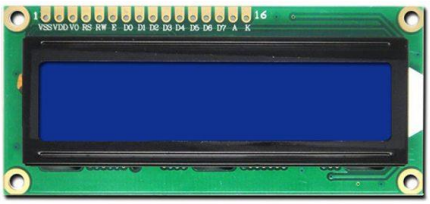



*Figure 13- LCD*

13. **Ultrasonic Sensor:** An ultrasonic sensor is an electronic device that measures the distance of a target object by emitting ultrasonic sound waves, receiving it, and calculating the time it took to reach the obstacle. The module is shown in Fig.14.

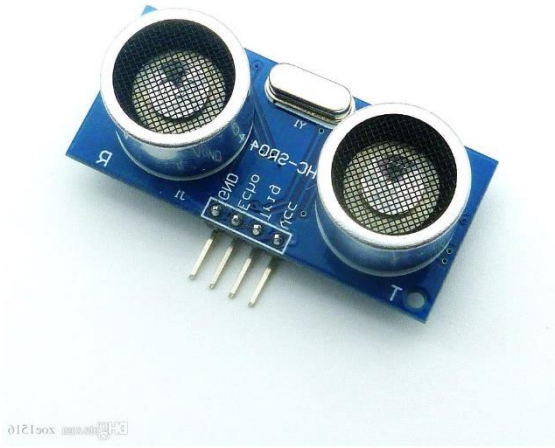



*Figure 14- Ultrasonic sensor*

### 3.1.1 Cost Model

The costs of the components used in this project are presented in Table 1.

*Table 1- Cost Model of Hardware Components*

| **Component** | **Cost** |
|---|---|
| Neurosky Mindwave EEG | 7,800 LE |
| 2 ATmega32 | 310 LE |
| 2 Smart robot car kits | 600 LE |
| 2 RF Modules | 130 LE |
| Arduino Uno | 385 LE |
| Motor Speed Sensor with Encoder Disk | 90 LE |
| 2-18650, 3.7 battery | 80 LE |

| | |
|---|---|
| GPS Module | 400 LE |
| 1 LCDs | 90 LE |
| Potentiometer | 5 LE |
| 4 LEDS | 5 LE |
| 2 Buzzers | 10 LE |
| Total Cost | 9905 LE |

## 3.2 Proposed Work

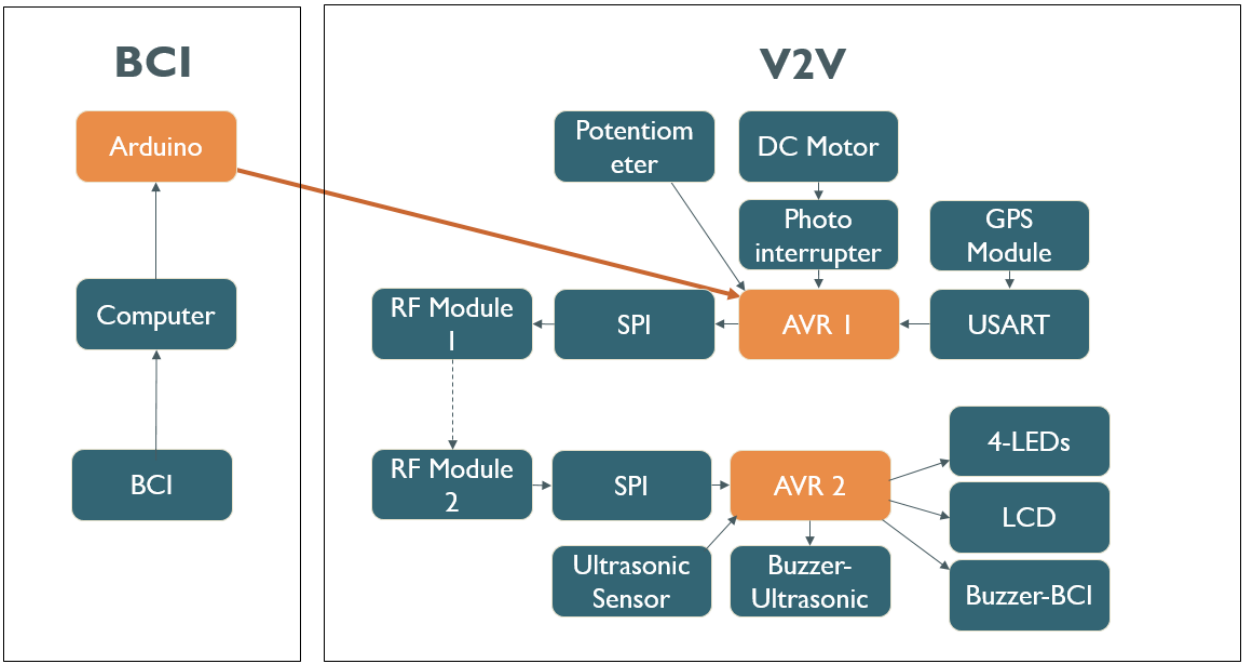



*Figure 15- Block Diagram*

In Fig. 15, we can see the full block diagram of our system.

For the V2V communication system, we will experiment on 2 cars for simplicity, each car represented by an AVR microcontroller. The drowsy driver's car is represented by AVR1, while the nearby car is represented as AVR2.

We will utilize Python V3 software on our computer to extract the EEG signals. Subsequently, we will train an AI model using drowsiness data to detect and classify levels of drowsiness/sleepiness based on the extracted EEG signals. The resulting drowsiness state is then sent to an Arduino Microcontroller, which in turn sends the data to the AVR microcontroller representing the drowsy driver (AVR1) in order to prepare it to be sent to the nearby car.

If a high level of drowsiness level is detected in the drowsy driver's car, a buzzer will be activated in the nearby car as a form of a warning. Additionally, the information regarding the drowsy driver's car will be displayed on the nearby vehicle, which will include:

- The location using a GPS module which will be displayed on an LCD
- The speed of the car using which will be expressed using 4-LEDs

The communication will be done using 2 RF modules, one in each car.

Moreover, we will employ an ultrasonic sensor module to sound a buzzer if the distance between car 1 and car 2 exceeds the minimum limit, warning the drivers.

## 3.3 Electrical BCI to Arduino Connection trials

The Neurosky Mindwave comes with a Dongle that wirelessly connects to it through RF waves. The dongle comes with Tx and Rx pins that can be directly connected to

the Arduino Uno through UART protocol. However, in order to use these pins, the connection to the USB port needs to be severed. The 2 traces to the USB port are shown in Fig 16.

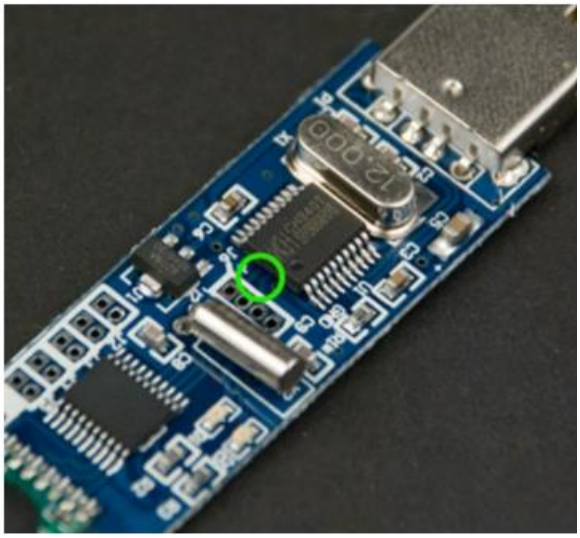


*Figure 16- TX and RX pins to the CH340 chip*

A note mentioned that no one would be able to use the USB dongle with a computer after cutting the traces. Unfortunately, since the NeuroSky Mindwave BCI was not owned by us, we could not go further into this trial. Therefore, we will be using our laptop as an intermediary between the microcontroller and the BCI device to establish a connection.

The connection between the BCI and the local host will be established using a software called TGC connector.

## 3.4 NeuroSky MindWave Communication

This section describes the various types of data values which are transmitted by the MindWave headset.

**Poor Signal Quality (POOR_SIGNAL)**

This unsigned one-byte integer value describes how poor the signal measured by the MindWave is. The value ranges between 0 and 200, where 200 specifically means that the Mindwave contacts are not touching the user's skin. Any other non-zero number is an indication of the level of noise contamination present in the most recent measurements. The refresh rate of this value is typically one second. The following is a list, in order of severity, of disturbances which may result in a poor quality signal:

- No one is wearing the MindWave headset.
- Poor contact between any of the sensors and the user's skin. This could be a result of hair blocking a sensor or the headset is not placed properly on the user's head.
- Excessive motion of the user which results in jostling of the headset.
- Excessive environmental electrostatic noise.

**eSenseTM Meters**

The meter value of the two Mindwave eSenses, Attention and Meditation, is reported on a relative eSense scale of 1 to 100. A value of 0 indicates that the MindWave is unable to calculate an eSense level with a reasonable amount of reliability. This is usually due to excessive noise as explained in the poor signal quality section. The descriptions of the other values on the eSense meter are given in table 2.

*Table 2- eSense meter description*

| eSense Range | Description | Interpretation |
|---|---|---|
| 1 to 20 | "Strongly lowered" eSense levels | These levels may indicate states of distraction, agitation, or abnormality, according to the opposite of each eSense. |
| 20 to 40 | "Reduced" eSense levels | |
| 40 to 60 | "Neutral" | Similar to "baselines" which are established in conventional EEG techniques. |
| 60 to 80 | "Slightly elevated" eSense levels | Indicate levels which are possibly higher than normal for a given person. |
| 80 to 100 | "Elevated" eSense levels | Indicate heightened levels of an eSense. |

Each interpretation represents a relatively wide range of the eSense meter because some parts of the eSense algorithm are dynamically learning which sometimes implement "slow-adaptive" algorithms to adjust to natural fluctuations and trends of each user. These algorithms account and compensate for the normal ranges of variance and fluctuations of EEG in the human brain. This is one of the reasons why the MindWave is able to be used by a wide range of individuals in a wide variety of personal and environmental conditions while still providing good accuracy and reliability.

**Attention eSense (ATTENTION)**

This data value is an unsigned, single byte which represents the current eSense Attention meter of the user. It has a range between 0 and 100. The eSense Attention meter measures the intensity of a user's level of attention or mental focus. Heightened levels of Attention may be achieved by intense concentration and directed mental activity. Actions which may lower the Attention meter include distractions, wandering thoughts, lack of focus or anxiety. The refresh rate of this value is typically one second.

**Meditation eSense (MEDITATION)**

This unsigned single-byte value represents the current eSense Meditation meter of the user. It has a range between 0 and 100. The eSense Mediation meter measures the level of a user's mental calmness or relaxation. It must be noted that Meditation is a measure of an individual's mental levels, not physical levels. Therefore, simply relaxing the muscles in the body may not immediately result in a heightened level of Meditation, but with most individuals, relaxing the body may aid the mind to relax as a consequence. Closing one's eyes turns off the mental activities in the brain which process images from the eyes, which then increases the Meditation meter levels. Actions which may lower the Mediation meter include distractions, wandering thoughts, anxiety, agitation and sensory stimuli. The refresh rate of this value is typically one second.

**Raw Wave Value (RAW Wave Value)**

This signed 16-bit integer value represents a single raw wave sample. It has a range between -32768 and 32768. This value is refreshed 512 times every second.

**EEG Power (ASIC_EEG_POWER)**

The current magnitude of the eight commonly recognized types of EEG is represented by this value. It is made up of a series of eight 3-byte unsigned integers in little-endian format. The eight EEG powers are represented in the following order:

- Delta (0.5 – 2.75 Hz)
- Theta (3.5 – 6.75 Hz)
- Low-alpha (7.5 – 9.25)
- High-alpha (10 – 11.75 Hz)
- Low-beta (13 – 16.75 Hz)
- High-beta (18 – 29.75 Hz)
- Low-gamma (31 – 39.75 Hz)
- Mid-gamma (41 – 49.75 Hz)

We will use the attention, meditation, and the eight frequency values as an input to our AI model. The model will learn the correlation between these features and drowsiness, and will later be able predict the drowsiness state of the driver. The refresh rate of this value is typically one second.

# Chapter 4: Design and Implementation of V2V System

This chapter provides a summary of modular programming and the header files utilized in the AVR codes. It also covers the peripherals available in the ATmega32 microcontroller used for the development of our project.

## 4.1 Modular Programming

Each Driver consists of 3 files, as shown in Fig.17:

- Configuration Header file (.h): hardware interface definition
- Interface Header file (.h): for function prototypes and macros
- Private Header file (.h): for defining variables using register and bit addresses
- Source file (.c): contains the function bodies.

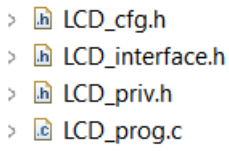



*Figure 17- Modular Driver*

## 4.2 Used Header Files

### 4.2.1 Bit Operations

The bit operations are shown in Fig.18, which provides a visual representation of how the bits are manipulated.

```
#define SET_BIT(VAR,BIT)        VAR |=  (1 << (BIT))
#define CLR_BIT(VAR,BIT)        VAR &= ~(1 << (BIT))
#define GET_BIT(VAR,BIT)        ((VAR >> BIT) & 1  )
#define TOG_BIT(VAR,BIT)        VAR ^=  (1 << (BIT))
```

*Figure 18- Bit Operations*

### 4.2.2 Standard Data Types

Fig.19 provides an illustration of the data types created to be used in the project.

```
typedef unsigned char u8;
typedef unsigned short int u16;
typedef unsigned long int u32;
typedef signed char s8;
typedef signed short int s16;
typedef signed int s32;
typedef float  f32;
```

*Figure 19- Standard Data Types*

## 4.3 Used Peripherals

### 4.3.1 DIO

ATmega32 has programmable input and output lines divided into 4 PORTs (A, B, C and D), 8 Pins each (0-7) as shown in Fig.20.

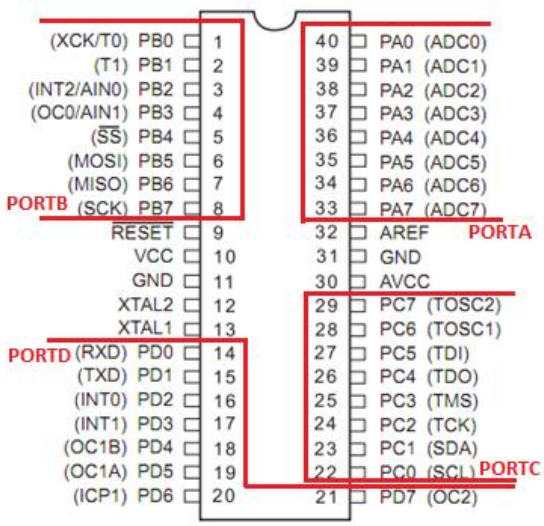



*Figure 20- DIO Ports*

Each PORT is controlled by 3 Registers:

- DDRx: Data Direction Register
- PINx: Input Register
- PORTx: Output Register

The functions and macros used in the DIO driver (refer to Fig.21) include:

```
// port MACROS
#define DIO_PORTA  0
#define DIO_PORTB  1
#define DIO_PORTC  2
#define DIO_PORTD  3

// Pin MACROS
#define DIO_PIN0  0
#define DIO_PIN1  1
#define DIO_PIN2  2
#define DIO_PIN3  3
#define DIO_PIN4  4
#define DIO_PIN5  5
#define DIO_PIN6  6
#define DIO_PIN7  7
```

```
// DIO status
#define DIO_HIGH  1
#define DIO_LOW   0

// DIO Direction
#define DIO_INPUT    0
#define DIO_OUTPUT   1

#define DIO_PORT_OUTPUT 0xFF
#define DIO_PORT_INPUT   0
#define DIO_PORT_HIGH 0xFF
#define DIO_PORT_LOW   0
```

```
void DIO_vidSetPinDir(u8 u8PortId,u8 u8PinId,u8 u8Dir);
void DIO_vidSetPinVal(u8 u8PortId,u8 u8PinId,u8 u8Val);
void DIO_vidSetPortDir(u8 u8PortId,u8 u8Dir);
void DIO_vidSetPortVal(u8 u8PortId,u8 u8Val);
u8   DIO_u8GetPinVal(u8 u8PortId,u8 u8PinId);
```

*Figure 21- DIO Functions and Macros*

### 4.3.2 UART

UART is a hardware communication protocol that uses asynchronous serial communication with configurable speed and enables two devices to exchange data. It works in full duplex mode because it has 2 separate transmission and reception lines as shown in Fig.22.

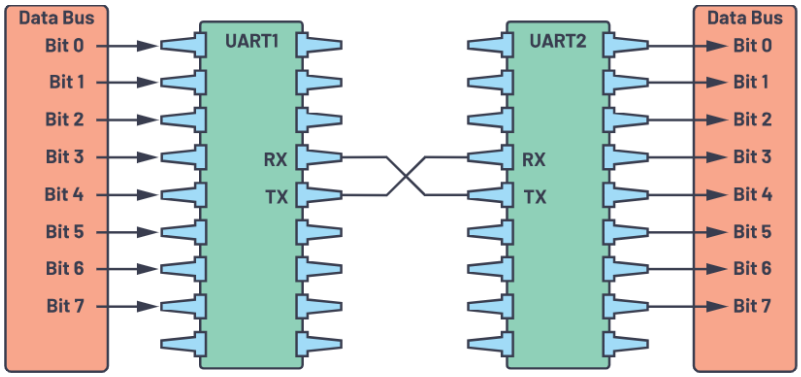



*Figure 22- UART transmission*

**Frame format:**

A packet consists of a start bit, data frame, a parity bit, and stop bits. The start and stop bits serve as a synchronization mechanism. They indicates the beginning and the end of a data byte and allows the receiver to align itself with the incoming data stream. For visual reference, please refer to Fig.23.


*Figure 23- UART Packet*

**Baud Rate:**

Baud rate is the rate at which information is transferred to a communication channel. For UART and most serial communications, the baud rate needs to be set the same on both the transmitting and receiving device to manage synchronization. Failure to do so may affect the timing of sending and receiving data that can cause discrepancies during data handling. Since UART is used for the GPS Module to communicate with the ATMEGA32, and the GPS Module sends serial data at a bit rate of 9600 bps, the baud rate is set to 9600 bps for the ATMEGA32 as well. Defining the baud rate in the GPS code is shown in Fig. 24.

```
#define USART_BAUDRATE 9600
#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)
```

*Figure 24- GPS UART Baud Rate*

**Driver:**

Two functions were implemented (refer to Fig.25):

- Usart_init(): which initializes the UART communication module by enabling the UART transmitter, receiver and the Receive Complete Interrupt. It also selects the baud rate we will be working on which is initialized above as 9600.

- Usart_getch(): The loop waits for the RXC (Receiver Complete) bit to be set, indicating that there is data available to be read. Once the RXC bit is set, the loop exits, and the function proceeds to return the value stored in the UART Data Register (UDR) which contains the received character from the UART module.

```
void usart_init()
{

UCSRB |= (1<<RXCIE) | (1 << RXEN) | (1 << TXEN);
UCSRC |= (1 << URSEL) | (1 << UCSZ0) | (1 << UCSZ1);

UBRRL = BAUD_PRESCALE;
UBRRH = (BAUD_PRESCALE >> 8);
}

unsigned int usart_getch()
{

while ((UCSRA & (1 << RXC)) == 0);
return(UDR);
}
```

*Figure 25- UART Functions*

### 4.3.3 SPI

In SPI protocol, the devices are connected in a Master – Slave relationship in a multi – point interface. In this type of interface, one device is considered the Master of the bus (usually a Microcontroller) and all the other devices (peripheral ICs or even other Microcontrollers) are considered as slaves. The master device controls the slave device, and the slave device takes the instruction from the master device.

SPI is a full-duplex, synchronous type serial communication i.e. it uses a dedicated clock signal to synchronize the Master and Slave. The master device generates the clock signal, which is used by both the master and slave devices to synchronize the transfer of data. The clock signal ensures that the data is sampled and transmitted at the correct timing on both ends.

**Configuration:**

- **MOSI:** MOSI stands for Master Output Slave Input. It is used to send data from the master to the slave.
- **MISO:** MISO stands for Master Input Slave Output. It is used to send data from the slave to the master.
- **SCK or SCLK (Serial Clock):** It is used to the clock signal.
- **SS/CS (Slave Select / Chip Select):** It is used by the master to send data by selecting a slave.

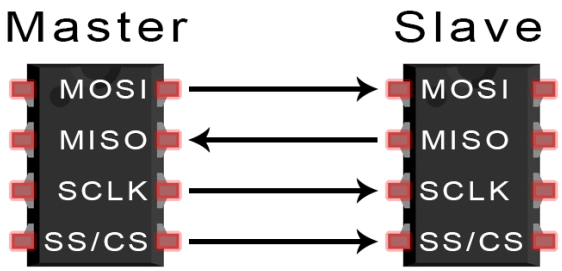



*Figure 26- SPI Module*

To begin SPI communication, the master must send the clock signal and select the slave by enabling the CS signal. During SPI communication, the data is simultaneously transmitted (shifted out serially onto the MOSI bus) and received (the data on the bus (MISO) is sampled or read in). The serial clock edge synchronizes the shifting and sampling of the data. This is illustrated in Fig.27.

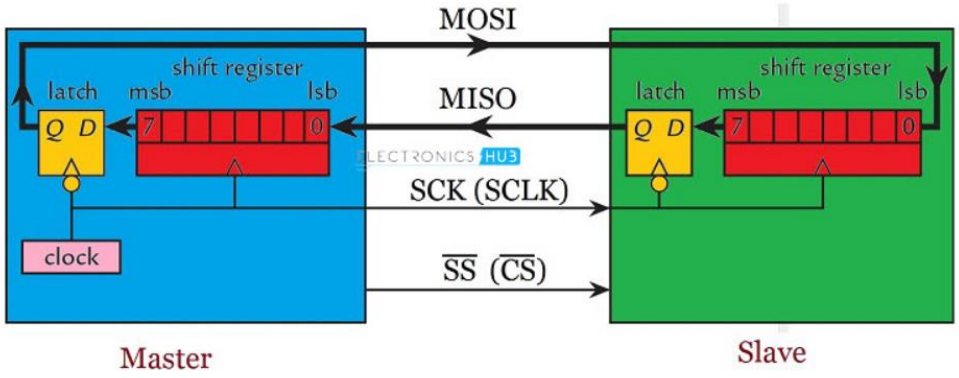



*Figure 27- SPI Data Transfer*

In our project, we will use the SPI interface to establish a communication between our microcontroller (master) and the NRF module (slave).

### 4.3.4 ADC

The ATmega32 has an analog to digital converter with a 10-bit resolution (converts a value to the range of 0→1023). The ADC is connected to a multiplexer providing 8 channel inputs at the pins of PORTA shown in Fig. 28.

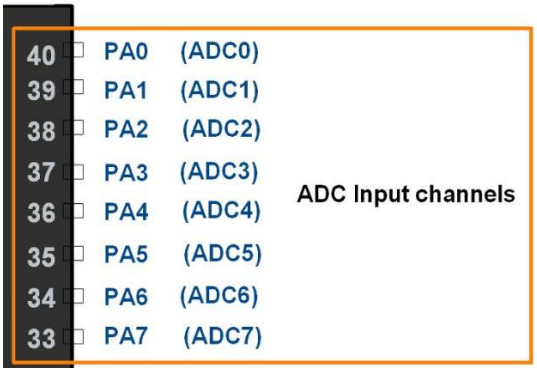



*Figure 28- ADC Port*

It converts the analog input voltage to a 10-bit digital value and is configured using the following registers:

- ADMUX: ADC Multiplexer Selection Register.
- ADCSRA: ADC Control & Status Register.

Output of the ADC is stored in ADCW which is stored in two bytes:

- ADCH: High Byte Data Register
- ADCL: Low Byte Data Register

The functions used for the ADC peripheral are illustrated in Fig.29.

```
void ADC_Init()             /* ADC Initialization function */
{
    DDRA = 0x00;            /* Make ADC port as input */
    ADCSRA = 0x87;          /* Enable ADC, with freq/128 */
    ADMUX = 0x40;           /* Vref: Avcc, ADC channel: 0 */
}

int ADC_Read(char channel)      /* ADC Read function */
{
    ADMUX = 0x40 | (channel & 0x07);  /* set input channel to read */
    ADCSRA |= (1<<ADSC);        /* Start ADC conversion */
    while (!(ADCSRA & (1<<ADIF)));  /* Wait until end of conversion */
    ADCSRA |= (1<<ADIF);        /* Clear interrupt flag */
    _delay_ms(1);           /* Wait a little bit */
    return ADCW;            /* Return ADC word */
}
```

*Figure 29- ADC Functions*

The ADC peripheral will be used to read analog voltage potentiometer values later on.

### 4.3.5 TIMERs

ATmega32 has 3 Timers that can be used for waveform generation, delays, and capturing events.

- Timer0: 8-bit
- Timer1: 16-bit
- Timer2: 8-bit

The Basic Timer Registers are:

- TCNTn: “Timer Counter Register” which contains the value of the time elapsed.
- TCCRn: “Timer Counter Control Register” which enables and disables the counter while setting the prescalar value of the timer used.

Timer 0 and Timer 2 will be used to calculate the revolutions per second of the wheel, while timer 1 will be used to calculate the distance of the ultrasonic module.

# Chapter 5: System Implementation of V2V System

Chapter 5: System Implementation of V2V System focuses on the implementation of various modules in the V2V system. It covers the LCD display, GPS module for position tracking, ultrasonic module for proximity detection, speed measuring system using a photo interrupter module, DC motor driver for speed control, and the NRF24l01+ module for direct communication between vehicles. The chapter provides detailed explanations of the components, their specifications, code snippets, and results obtained during testing.

## 5.1 LCD for Display

In LCD 16×2, the term LCD stands for Liquid Crystal Display that is used to display data. As the name suggests, it includes 16 Columns & 2 Rows so it can display 32 characters (16×2=32) in total, and every character will be made with 5×8 (40) Pixel Dots.

The LCD pinout configuration shown in Fig.30 is explained below:

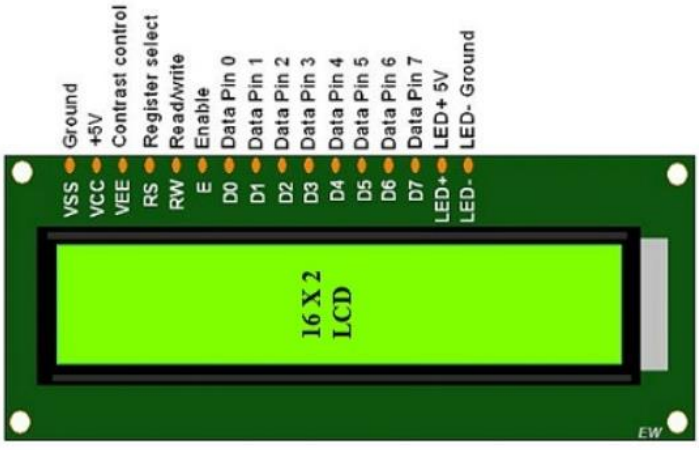



*Figure 30- LCD Display*

- Pin1 (Ground): This pin connects the ground terminal.
- Pin2 (+5 Volt): This pin provides a +5V supply to the LCD
- Pin3 (VE): This pin selects the contrast of the LCD.
- Pin4 (Register Select): Selects command register when low, and data register when high. The main function of the command register is to perform instructions illustrated on LCD. That assists in data clearing, changes the cursor location, and controlling the display. Once we transmit data to LCD, it shifts to the data register to process the data.
- Pin5 (Read & Write): Low to write to the register; High to read from the register.
- Pin6 (Enable): Sends data to data pins when a high to low pulse is given.
- Pin7-Pin14 (Data Pin0-Data Pin7): The data pins connected through the microcontroller for data transmission.
- Pin15 (LED Positive): This is a positive terminal of the backlight LED of the display & it is connected to +5V to activate the LED backlight.
- Pin16 (LED Negative): This is a negative terminal of a backlight LED of the display & it is connected to the GND terminal to activate the LED backlight.

### 5.1.1 Driver

The LCD functions used in the project are shown in Fig.31:

```
void LCD_vidInit(void);
void LCD_vidSendCommand(u8 u8Cmd);
void LCD_vidSendChar(u8 u8Char);
void LCD_vidWriteStr(u8* pu8Str);
void LCD_vidGoToXY(u8 x, u8 y);
void LCD_Custom_Char (u8 loc, u8 msg[]);
void LCD_PRINT_INT(u16 x);
```

*Figure 31- LCD Functions*

The most common LCD used commands are:

1) LCD_vidSendCommand(0x80) which sets the cursor at the beginning of the first row.
2) LCD_vidSendCommand(0xC0) which sets the cursor at the beginning of the second row.
3) LCD_vidSendCommand(LCD_CLR) or LCD_vidSendCommand(0x01) which clears the LCD display.

## 5.2 GPS Module

GPS modules are equipped with compact processors and antennas designed to receive data from satellites. The module's antenna can detect signals from up to four satellites, and the processor can then precisely calculate the module's position and time.

The Ublox NEO-6m GPS Module has the following specifications:

- It operates at a supply voltage of 3.3V, but it can tolerate up to 5V due to the presence of an internal voltage regulator.

- The module requires a minimum power of 162 dBm for maintaining a satellite connection, which is known as the tracking sensitivity.

- It offers a position update rate of 5Hz, meaning that the module updates its data five times per second.

The Ublox NEO-6m GPS Module employs the NMEA-0183 standard protocol for transmitting data via a serial port. NMEA, which stands for National Marine Electronics Association, is a widely adopted data format utilized by GPS manufacturers. NMEA utilizes straightforward ASCII characters, and the data transmitted in NMEA format includes PVT (position, velocity, time) information that is computed by the GPS receiver.

**NMEA Message Format**

The NMEA format consists of sentences, each containing a specific set of data fields. The sentence format starts with the symbol ($) and is followed by GP, which is the prefix for GPS receivers. After that, a 3-letter sequence is used to indicate the data type that defines the content of the sentence.

An example of some incoming GPS messages and the description of the data types are shown in Fig.32 and Fig.33.

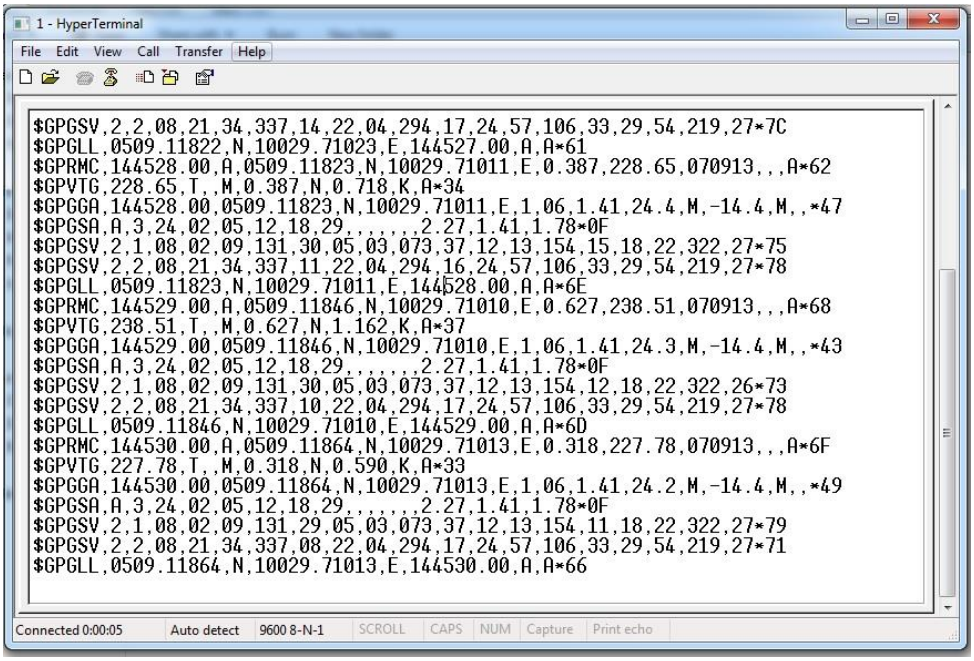



*Figure 32- GPS Messages Example*

| Message | Description |
|---|---|
| GGA | Time, position and fix type data |
| GLL | Latitude, longitude, UTC time of position fix and status |
| GSA | GPS receiver operating mode, satellites used in the position solution, and DOP values |
| GSV | Number of GPS satellites in view satellite ID numbers, elevation, azimuth, & SNR values |
| MSS | Signal-to-noise ratio, signal strength, frequency, and bit rate from a radio-beacon receiver |
| RMC | Time, date, position, course and speed data |
| VTG | Course and speed information relative to the ground |
| ZDA | PPS timing message (synchronized to PPS) |
| 150 | OK to send message |
| 151 | GPS Data and Extended Ephemeris Mask |
| 152 | Extended Ephemeris Integrity |
| 154 | Extended Ephemeris ACK |

*Figure 33- NMEA Data Types*

Our interest is in the GGA message as it contains the position of the GPS module. Here's an example and a breakdown of a GGA sentence:

$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47

- $GPGGA: The "$" symbol denotes the start of the sentence, followed by the talker ID "GP" (which represents GPS) and the sentence type "GGA" (Global Positioning System Fix Data).
- 123519: The UTC (Coordinated Universal Time) timestamp indicating the time of the fix in hours, minutes, and seconds (HHMMSS format).
- 4807.038: The latitude of the fix in degrees and decimal minutes.
- N: The indicator for North latitude.
- 01131.000: The longitude of the fix in degrees and decimal minutes.
- E: The indicator for East longitude.

The last four lines are our target as we want to acquire the latitude and the longitude of the car. We can do this by parsing the incoming GPS data. We can see how this is done in the flowchart shown in Fig.35 and the code shown in Fig.36.

## Flowchart


*Figure 34- GPS Flow Chart*

# Code

```
do
{
    u8 value;
    value = usart_getch();

    if (value == '$')
    {
        value = usart_getch();

        if (value == 'G')
        {
            value = usart_getch();

            if (value == 'P')
            {
                value = usart_getch();

                if (value == 'G')
                {
                    value = usart_getch();

                    if (value == 'G')
                    {
                        value = usart_getch();

                        if (value == 'A')
                        {
                            gpgga_found = 1; // Set the flag when "$GPGGA" is found
                        }
                    }
                }
            }
        }
    }



  if (gpgga_found)
  {
      value=usart_getch();
                  if(value==',')
                  {
                   value=usart_getch();
                   while(value!=',')
                   {
                    value=usart_getch();
                   }
                   lati_value[0]=usart_getch();
                   value=lati_value[0];
                   for(i=1;value!=',';i++)
                   {
                    lati_value[i]=usart_getch();
                    value=lati_value[i];
                   }
                   *lati_dir=usart_getch();
                   value=usart_getch();
                  /* while(value!=',')
                   {
                    value=usart_getch();
                   }*/
                   longi_value[0]=usart_getch();
                   value=longi_value[0];
                   for(i=1;value!=',';i++)
                   {
                    longi_value[i]=usart_getch();
                    value=longi_value[i];
                   }
                   *longi_dir=usart_getch();


                   LCD_vidSendCommand(0x80);
                   LCD_vidWriteStr(lati_value);
                   LCD_vidSendChar(*lati_dir);

                   LCD_vidSendCommand(0xC0);
                   LCD_vidWriteStr(longi_value);
                   LCD_vidSendChar(*longi_dir);
                  }

  }

 } while (!gpgga_found);
}
```

*Figure 35- GPS Code*

### 5.2.1 Results

As shown in Fig.36, the GPS module was tested, and the vehicle coordinates (latitude and longitude) were extracted.


*Figure 36- GPS Results*

To check if the results were correct, we changed the format to decimal degrees which is used by google maps to detect a location. The calculation was done using an online calculator as shown in Fig.37.

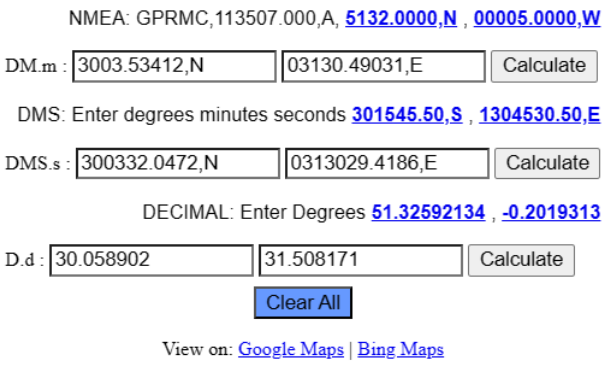



*Figure 37- NMEA to Decimal Degrees Conversion*

The location on google maps was in fact the location of the GPS Module as shown in Fig.38.

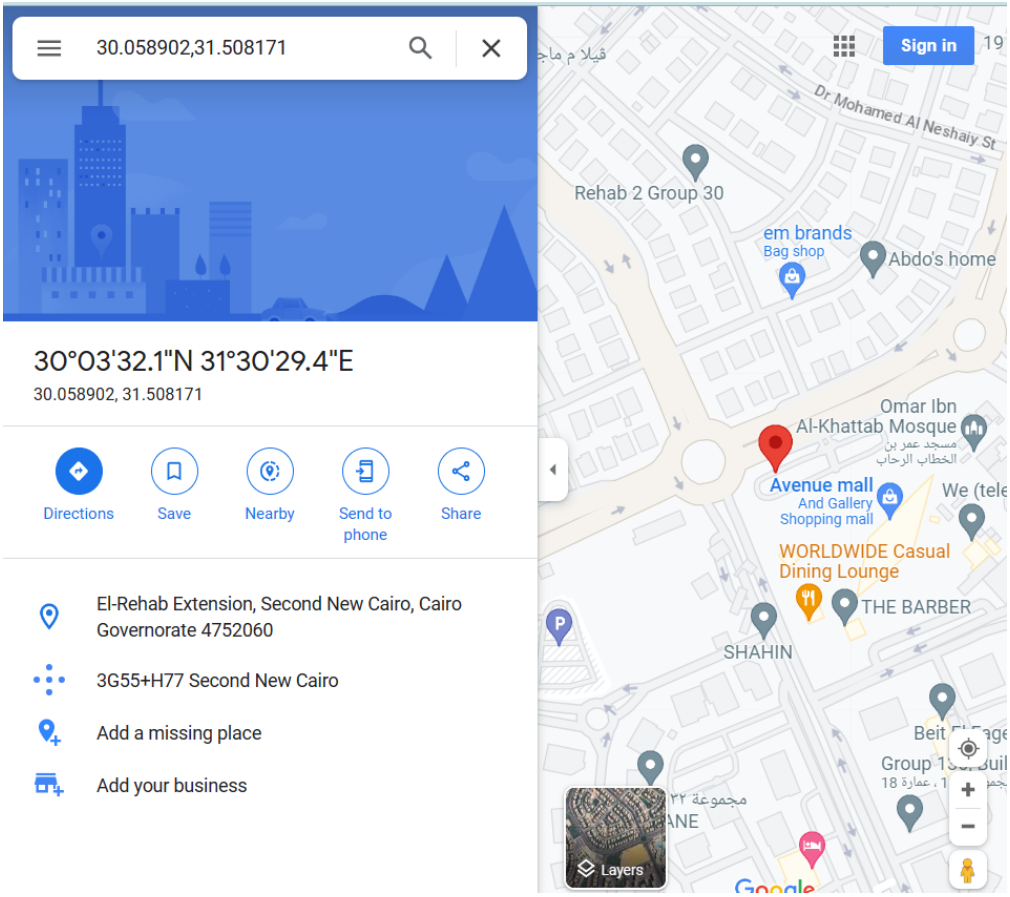



*Figure 38- Google Maps GPS Location*

## 5.3 Ultrasonic Module

The ultrasonic sensor utilizes a transducer to transmit and receive ultrasonic waves that provide information about the proximity of an object. A transducer is a device that converts changes in a physical quantity into an electrical signal, and vice versa. The ultrasonic sensor is comprised of two essential elements: the transmitter, which utilizes piezoelectric crystals to emit sound waves, and the receiver, which captures the sound waves after they have traveled to and from the target object.

The module operates by emitting a 40KHz ultrasonic pulse, which acts as a signal. When this pulse encounters an obstacle in its path and reflects back to the receiver, it indicates the presence of an obstacle. This process is visually explained in Fig.39.

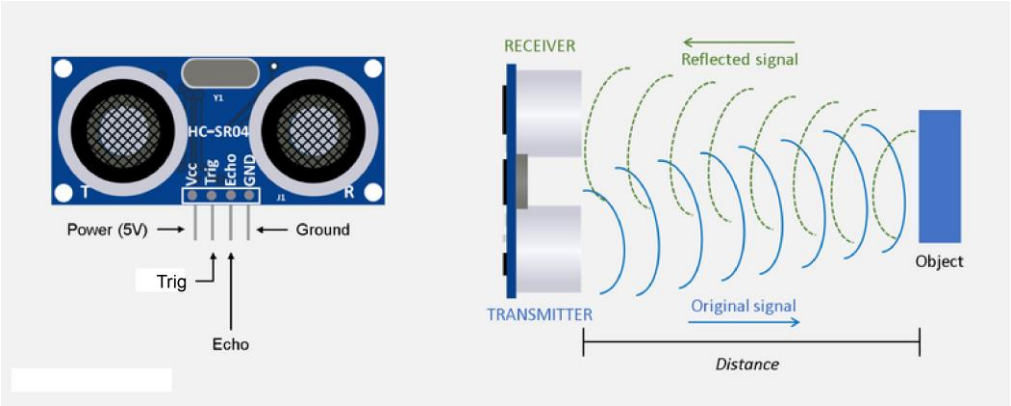



*Figure 39- Ultrasonic Module Explanation*

HC-SR04 Specs:

- Power Supply: 5V
- Range: 2cm- 400cm
- Accuracy: 3mm

The time to the obstacle is measured by dividing the time of the original signal output from the transmitter to the receiver by two. We then multiply the time taken to the obstacle by the speed of sound to get the distance to the obstacle. The same approach can be employed by utilizing microseconds to measure the speed and obtaining distance as a unit of centimeters. The formula becomes:

(1) Formula: Time of wave from TX to RX (uS) / 58 = distance (centimeters)

**Code:**

The Ultrasonic Driver uses DIO and TIMER1 where:

- Trigger Pin: PORTD-PIN2 (Set as OUTPUT Pin)
- Echo Pin: PORTB-PIN0 (Set as INPUT Pin)
- TIMER1: Set to count in Normal Mode with prescalar=8 to count in 1us ticks.

(2) Tick Time= Prescalar/ Frequency of CPU = 8/8MHz= 1us

- Buzzer: PORTD-PIN7

The timing diagram of the ultrasonic module works as follows:
We apply a 10µsec pulse onto the trigger pin which outputs 8 pulses of 40 KHz sound waves. The Echo Pin is then set to HIGH and stays HIGH until the transmitted pulses bounces back. A visual representation is shown in Fig.40.

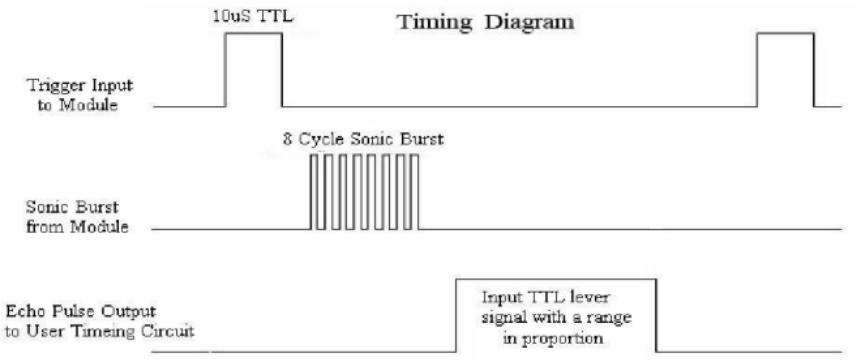



*Figure 40- Ultrasonic Timing Diagram*

The TIMER1 counter starts counting when the Echo pin is set to HIGH until it goes back to LOW. This gives us the time (in microseconds) it takes for the pulse to travel from the transmitter to the receiver. We then use this time to get the distance in centimeters. In the code, the ultrasonic was calibrated to get accurate readings, which is why we divided the pulse by 29 instead of 58. The code implementation for the process mentioned above is shown in Fig.41.

```
while(1) {

    DIO_vidSetPinVal(DIO_PORTD, DIO_PIN2, DIO_HIGH);
        _delay_us(10);
        DIO_vidSetPinVal(DIO_PORTD, DIO_PIN2, DIO_LOW);

        while(DIO_u8GetPinVal(DIO_PORTB,DIO_PIN0)==0);
            TCCR1B|=(1<<CS11);//enabling counter, prescaler= 8 to count in 1us ticks

        while(DIO_u8GetPinVal(DIO_PORTB,DIO_PIN0)==1);
            TCCR1B=0;//disabling counter
            pulse=TCNT1;//count memory is updated to integer
            TCNT1=0;//resetting the counter memory

        distance = pulse/29;

        //buzzer
        if(distance<15)
        {
            DIO_vidSetPinVal(DIO_PORTD, DIO_PIN7, DIO_HIGH);
        }
        else
        {
            DIO_vidSetPinVal(DIO_PORTD, DIO_PIN7, DIO_LOW);
        }
```

*Figure 41- Ultrasonic Code*

## Flowchart:

In our project, we are utilizing the ultrasonic sensor to detect the distance between the car of a drowsy driver and the surrounding cars. The purpose is to ensure that a safe distance is maintained. If the distance falls below a specified minimum range, a buzzer is activated to alert the drowsy driver of the potential danger. This concept is visually explained in Fig.42.

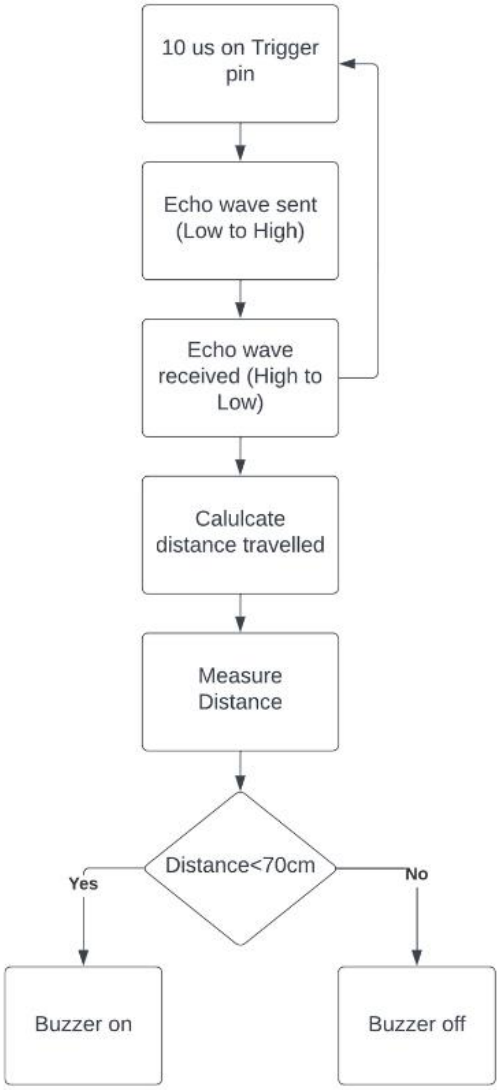



*Figure 42- Ultrasonic Flow Chart*

## 5.4 Speed Measuring System

### 5.4.1 Photo Interrupter Module

This module consists of light emitting elements & light receiving elements. It works by detecting light blockage when a target object comes between both elements. A detailed visual of how this module works is shown in Fig.43.

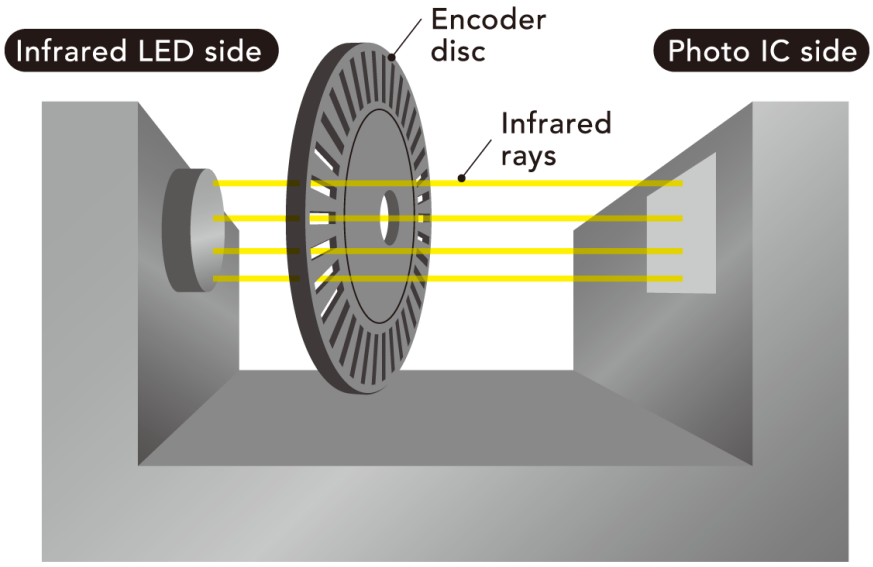



*Figure 43- Optical Sensing with Photo Interrupter Module*

The encoder wheel will be fixed to the DC motor gear on the other side of the wheel and will be inserted between the U-shaped sections of the Photo Interrupter Module as shown in Fig.44.

*Figure 44- Encoder wheel fixed to DC motor gear*

The operation of the encoder involves an IR sensor that detects whether the light is passing through or blocked as the wheel spins.

When the motor gear rotates, it causes the wheel encoder to rotate as well. As the encoder wheel turns, the IR sensor detects the transitions between the holes and no-holes sections of the wheel. Each transition corresponds to one hole being passed. To calculate the speed in RPS, we count the number of holes passed by the IR sensor in one second, and divide it by the number of holes in the encoder wheel. Since the wheel has 20 holes, we divide the hole count by 20. This value represents the speed of rotation for the wheel and indirectly for the connected motor gear.

The equation of RPS is:

(3) RPS = Hole Count/20.

In our system, the peripheral INT0 (Interrupt Zero) is responsible for counting the number of holes on the encoder wheel and storing the count in a variable called "counter," as illustrated in Fig.45. The photo interrupter module is equipped with a digital pin that is connected to the INT0 pin of the microcontroller. This pin outputs '0' when light is detected and '1' when light is not detected. Therefore, when a falling edge occurs, it indicates that a hole has been passed. We have configured the INT0 peripheral to trigger the Interrupt Service Routine (ISR) whenever a falling edge is detected, and the “counter” variable is incremented accordingly.

```
ISR(INT0_vect)
{
    counter++;   //incremenet number of holes
}
```

*Figure 45- Photo Interrupter Counter ISR Code*

We then used the peripheral Timer0 in order to count one second before proceeding to calculate the revolutions per second.

The "tick time" indicates the duration of each count performed by the timer, and is calculated using the formula below:

(4) Tick Time= Prescalar/ Frequency of CPU = 8/8MHz= 1us

The total number of ticks for an 8-bit timer is calculated as 2 raised to the power of 'n', where 'n' represents the number of bits. In this case, the total number of ticks is 256, indicating that the timer can count up to 256 before it overflows and restarts. This information helps us determine the range and precision of the timer for our application.

(5) Timer (8 bits) total number of ticks= 2^n = 256

This results in an overflow time of 256 microseconds. Understanding the overflow time allows us to effectively manage and handle events that occur when the timer reaches its maximum value.

(6) Overflow time= Number of ticks * Overflow Counts = 256 * 1us =256 us

In our application, we have enabled the timer interrupt to trigger after each overflow event. So every 256us, we enter the ISR (Interrupt Service Routine) shown in Fig.46. In the ISR, a variable called overflow_count, which represents the overflow counts needed to reach one second, is incremented by one. Our wanted delay=1s=1,000,000us. Overflow counts needed to reach 1 second is = 1,000,000us/256us = 3906.25. Therefore, the condition for RPS calculation will be when the variable overflow_count reaches 3906 counts.

Since the 0.25 is a decimal number, we will initialize our counter with the corresponding number at the beginning. The counts needed for 0.25 overflow counts is = 0.25*256us= 64. Therefore, we will preload out counter with 256-64=192 at the beginning of the code. (TCNT0=192).

When the condition of reaching one second is achieved, the RPS calculation is done by dividing the counter variable by 20.

In conclusion, the RPS is calculated and updated every second.

```
ISR(INT0_vect)
{
    counter++;   //incremenet number of holes

}

ISR(TIMER0_OVF_vect)
{
    ov_count++;


    if(ov_count==3906)   //count one second
    {
        TCCR0=0;  //stop counter
        rps= counter/20;   //rps calculation

        if(counter!=0)
        {
        LCD_vidSendCommand(LCD_CLR);
        LCD_vidSendCommand(0x80);
        LCD_vidWriteStr(str);
        LCD_PRINT_INT(counter);   //RPS
        }
        else
        {
            LCD_vidSendCommand(LCD_CLR);
            LCD_vidSendCommand(0x80);
            LCD_vidWriteStr(str2);  //No Wheel
        }

        counter=0;   //zero counter
        TCNT0=192;   //load number
        ov_count=0;  //zero overflow counter
        TCCR0|=(1<<1);   // 8 prescaler , start timer

    }

}
```

*Figure 46- Photo Interrupter Timer ISR Code*

### 5.4.2 DC Motor Driver

Duty cycle is the percentage of time a digital signal is in the "on" state.
As we can see in the figure below, a short duration pulse reduces the available voltage given to the motor which makes the wheel spin more slowly. However, a pulse with a long duration gives more voltage to the motor, increasing the speed of the wheel.
This phenomenon occurs when a high pulse width is applied to the motor, resulting in the motor being powered for a longer duration during each cycle of the PWM signal. Consequently, this leads to a higher average power being delivered to the motor over time. As the speed of the motor is directly proportional to the average power applied to it, increasing the pulse width results in a greater amount of power being delivered to the motor, thereby causing it to rotate faster. This is visually explained in Fig.47.

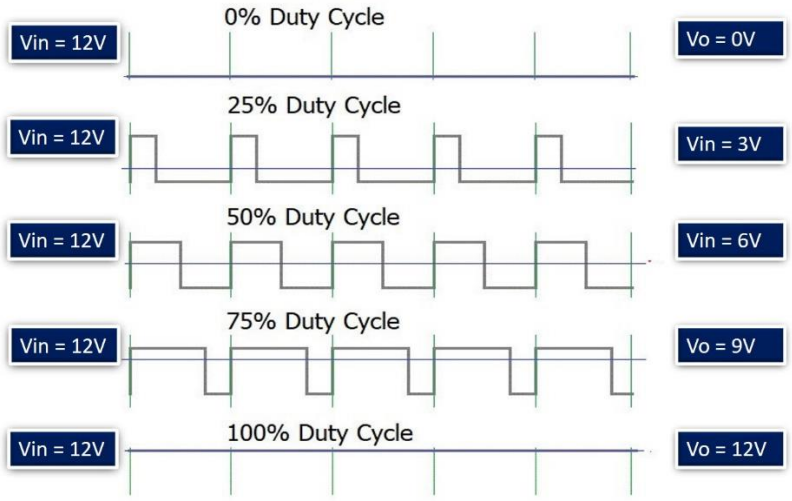



*Figure 47- Duty Cycle Relation to Voltage Output*

To enable the user to control the speed of the motor within a range, we incorporated a potentiometer, which is an electronic component used to vary resistance. The microcontroller utilizes the value from the potentiometer to adjust the duty cycle for PWM control. The potentiometer receives an input voltage ranging from 0 to 5 V.

This voltage is then converted to a digital value using an analog-to-digital converter, which provides a range of 0 to 1024. We want to input this value onto the Timer2 peripheral, but since the range of the timer is from 0 to 255, we divide the digital value by 4.

After the conversion of the potentiometer value to a number between 0-255, the microcontroller compares this value with the timer line ranging from 0-255. The time before the potentiometer value intersects with the timer line, the output state of the PWM signal is set to high. This allows the motor to receive a variable amount of power based on the position of the potentiometer. This concept is further illustrated in Fig.48, providing a clearer visual representation of the relationship between the potentiometer value, timer line, and the resulting PWM signal.

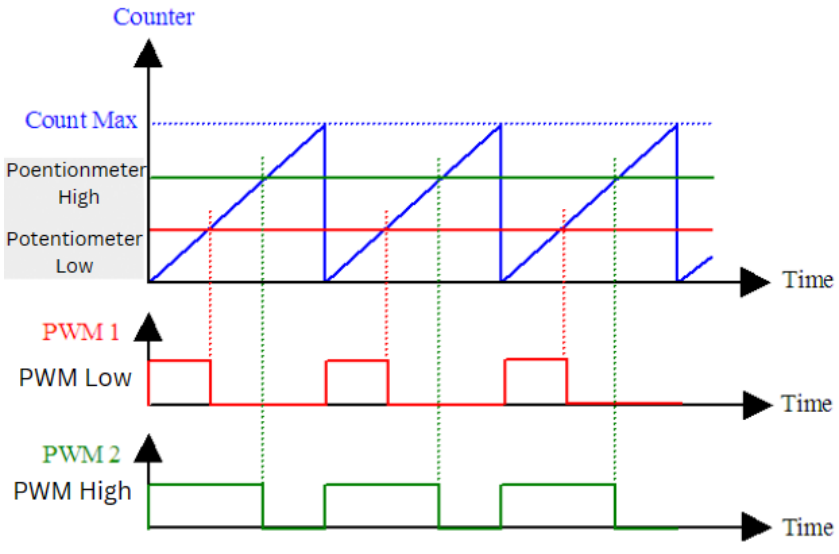



*Figure 48- Generating PWM Signal with Timer*

A DC motor driver is used to supply 12V to power a DC motor that is responsible for controlling the direction and speed of our wheel. The motor features an input enable pin, which is utilized to adjust its speed by receiving the PWM signal.

Additionally, the motor has In1 and In2 pins that serve the purpose of controlling the direction of rotation.

- "01" -> clockwise.
- "10" -> anticlockwise.
- "11"-> stop.
- "00" ->stop  11"-> stop.

The code for the DC motor driver is shown in Fig.49. It shows

```
void main()
{
        DDRC = 0xFF;            /* Make PORTC as output Port */
        ADC_Init();         /* Initialize ADC */
        TCNT0 = 0;          /* Set timer0 count zero */
        TCCR0 = (1<<WGM00)|(1<<WGM01)|(1<<COM01)|(1<<CS00)|(1<<CS01);/* Set Fast PWM with Fosc/64 Timer0 clock */
        /*In fast PWM mode, Clear OC0 on compare match, set OC0 at TOP*/
        while(1)
        {
            PORTC=0b00000010; // for rotate clock wise
            OCR0 = (ADC_Read(0)/4); /* Read ADC and map it into 0-255 to write in OCR0 register */
            /* Based on potentiometer value, pwm wave will change*/
        }
}
```

*Figure 49- DC Motor Driver Code*

## 5.5 NRF24l01+ Module

### 5.5.1 NRF vs WI-FI vs GSM

Table 3 provides a detailed comparison between the NRF, Wi-Fi, and GSM technologies.

*Table 3- NRF vs Wi-Fi vs GSM*

| **Technology** | **NRF** | **Wi-Fi** | **GSM** |
|---|---|---|---|
| **Range** | Short-range | Medium-range | Long-range |
| **Data Rate** | Moderate | High | High |
| **Infrastructure** | No infrastructure required | Requires access points or routers | Requires cellular infrastructure |
| **Interference** | Susceptible to interference in shared frequency band | Can experience interference in crowded Wi-Fi environments | Licensed frequency bands offer more reliable communication |
| **Cost** | Affordable | Affordable | Ongoing costs and subscriptions |
| **Auto ACK** | Yes | Yes | Yes |

In our project, we have chosen to use NRF for direct access point communication. This decision was based on several factors.

Firstly, we can adjust the frequency range of NRF to minimize the risk of interference, ensuring reliable communication within our specific environment. Secondly, since we are transmitting only essential information such as location, speed, and drowsiness state, we do not require a high data rate. NRF's moderate data rate is sufficient for our data transmission needs. Thirdly, as we are sending information to nearby cars within a short range, the long-range capabilities of GSM are not necessary.

By leveraging NRF's capabilities, we can establish direct communication between vehicles, ensuring that drowsiness alerts are sent promptly and reliably. This approach eliminates the dependence on access points or routers and allows for continuous communication even in the absence of an operational access point. This is crucial for our objective of real-time drowsiness detection and timely response, providing an efficient and effective solution for V2V (Vehicle-to-Vehicle) communication in our project.

### 5.5.2 NRF Functionality

The nrf24l01+ module uses SPI communication and operates at 2.4GHz. It can achieve a range up to 100 meters. It provides sufficient coverage for vehicles in close proximity, such as on the same road or within a small area. The module contains 125 selectable frequency channels. The bands are divided from 2400 to 2525 MHz with a channel spacing of 1MHz. The formula to choose to RF channel is:

(7) 2400+CH(Selected)

In our code, we chose our RF channel to be 2440MHz. This can be shown in Fig.50.

```
#define CONFIG_NRF_RF_CHANNEL    40
```

*Figure 50- Defining RF channel*

The NRF24L01+ module employs frequency-shift keying (FSK) modulation with three available data rates: 250 kbps, 1Mbps, and 2 Mbps. Since we are sending a few bytes of data per second, we chose the 250kbps option. With a lower data rate, the signal can propagate further, reaching nearby cars even if they are at an extended range from the transmitting vehicle. Moreover, the module allows for adjusting the output power, which can be set to 0 dBm, -6 dBm, -12 dBm, or -18 dBm. For our application, we selected an output power of 0 dBm to maximize the range of communication. By utilizing this higher power level, the signal strength is enhanced, facilitating better propagation and reception. These options are adjusted in the NRF driver and are shown in Fig.51.

```
#define CONFIG_NRF_DATA_RATE     NRF_RATE_250KBPS
#define CONFIG_NRF_TX_PWR        NRF_PWR_0DBM
```

*Figure 51- Defining RF Data Rate and Power Output*

The NRF24L01+ module features a Multiceiver network, allowing a receiver to simultaneously receive data from up to six transmitters. The receiver is equipped with six parallel data pipes, each assigned a unique address. Fig.52 shows a visual representation of this concept.

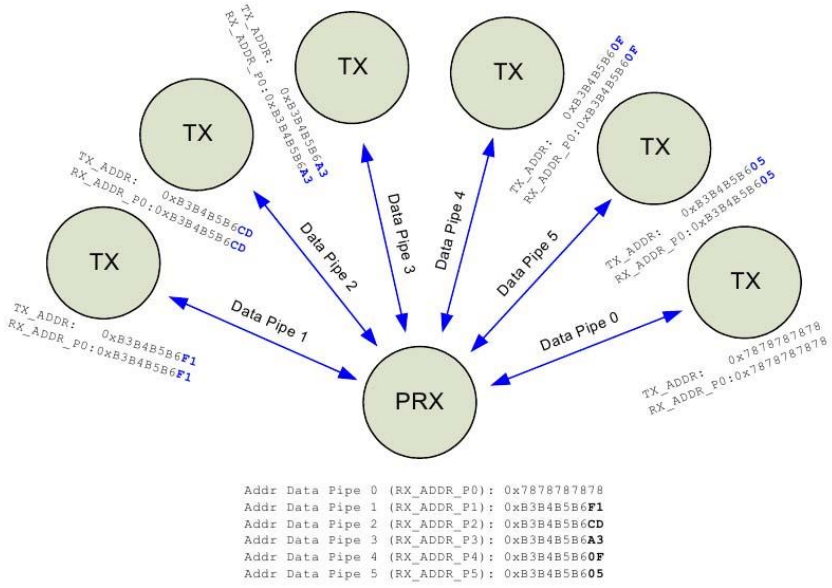



*Figure 52-- NRF24L01+ Data Pipes*

Additionally, an Auto ACK feature can be enabled to ensure reliable communication between the transmitter and receiver. Here's how the Auto ACK feature operates:

- Transmission: When the transmitter sends a data packet to the receiver, the receiver checks the packet for errors and successful reception. If the packet is error-free and received successfully, the receiver automatically sends an ACK packet back to the transmitter.

- Retransmission: If the transmitter does not receive an ACK within a specified time period or receives a negative acknowledgment (NACK), it assumes that the packet was not successfully delivered and initiates retransmission. In our code, we have set the maximum number of retransmissions to be 15 as shown in Fig.53. If the number of retransmissions exceeds 15, the packet is dropped.

```
#define CONFIG_NRF_TX_RETRANSMITS    15
```

*Figure 53- Defining RF retransmit number*

By leveraging the ACK feature provided by the NRF24L01+ module, we can establish a robust and reliable wireless communication system, which is particularly crucial for our application due to the significance of the transmitted data.

# Chapter 6: Drowsiness Detection System Implementation

In Chapter 6, we will explore the Brain-Computer Interface (BCI) systems and how they're enhanced with artificial intelligence. This combination opens up exciting possibilities for personalized experiences, real-time monitoring, and detecting drowsiness using EEG signals

## 6.1 AI Overview

### 6.1.1 Artificial Intelligence

The replication of human intellectual functions by machines, particularly computer systems, is known as artificial intelligence. Expert systems, natural language processing, speech recognition, and machine vision are some examples of specific AI applications.

Large amounts of labelled training data are ingested by AI systems, and they subsequently examine the data for correlations and patterns before employing these patterns to predict future states. As a result of evaluating millions of instances, an image recognition tool may learn to recognize and describe things in photographs, much as a chatbot that is given examples of text can learn to produce lifelike dialogues with humans.

The integration of artificial intelligence into our project serves the following purposes:

1) Personalization and adaptability: AI models can be trained on individual-specific EEG data, enabling personalized drowsiness detection for each person. These models can adapt to the unique variations and characteristics of an individual's baseline EEG signals. By accounting for individual differences, the accuracy of the drowsiness detection system can be enhanced, as it takes into consideration the specific EEG patterns and responses of each person.

2) Scalability: AI algorithms possess the ability to handle large volumes of data and can be trained on diverse datasets collected from different individuals or populations. This scalability allows for the development of robust and generalizable drowsiness detection models that can be applied to multiple individuals. By incorporating data from various sources, the AI model becomes more versatile and capable of accommodating variations in EEG signals across different people, thereby increasing its reliability and applicability in real-world scenarios.

3) Real-time monitoring: AI algorithms can be trained to process EEG data in real-time, enabling continuous monitoring of drowsiness levels. This is will useful in our applications such as we need an immediate alert to be sent to nearby cars in case of drowsiness detection.

4) Nonlinear relationships: The relationship between EEG signals and drowsiness is often nonlinear and difficult to model using traditional statistical methods. AI algorithms, particularly deep learning models, are capable of learning complex nonlinear relationships and can capture intricate patterns in the EEG data that might be indicative of drowsiness.

### 6.1.2 Machine Learning vs Deep Learning

Machine learning involves training models to make predictions or perform tasks based on data. In traditional machine learning, we need to manually engineer relevant features from the data before feeding it into the model. These engineered features capture the important aspects of the data that help the model make accurate predictions.

On the other hand, deep learning models can learn and discover relevant features from the data themselves, eliminating the need for explicit feature engineering. This will be especially valuable in EEG-based drowsiness detection, as identifying informative features from the data can be challenging due to its high dimensionality. Environmental factors such as noise, lighting conditions, temperature variations, and other external stimuli can influence EEG signals and potentially mask or distort the specific patterns indicative of drowsiness. This would make it difficult if not impossible to identify drowsiness without using artificial intelligence. The difference is more clearly illustrated in Fig.54.

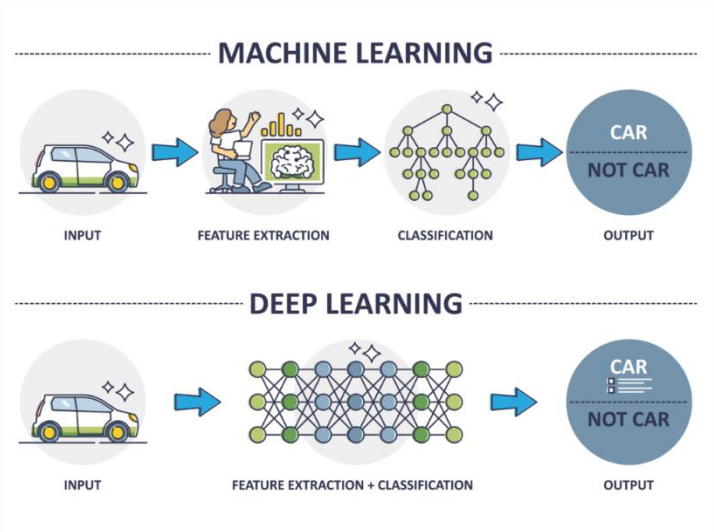



*Figure 54- Machine Learning vs Deep Learning*

### 6.1.3 Deep Neural Networks

Neurons, which are the building blocks of the human brain, communicate with one another via electrical impulses to help people interpret information.

Similar to this, neural networks are a type of artificial intelligence model inspired by the human brain. They are made up of interconnected nodes called neurons, which process and transmit information. These networks can be trained on data to learn patterns, make predictions, and solve problems. Neural networks are used in various applications such as image and speech recognition, recommendation systems, and even self-driving cars. They are powerful tools for handling complex tasks by leveraging their ability to learn from data and extract meaningful insights. A visual representation of a neural network is shown in Fig.55.

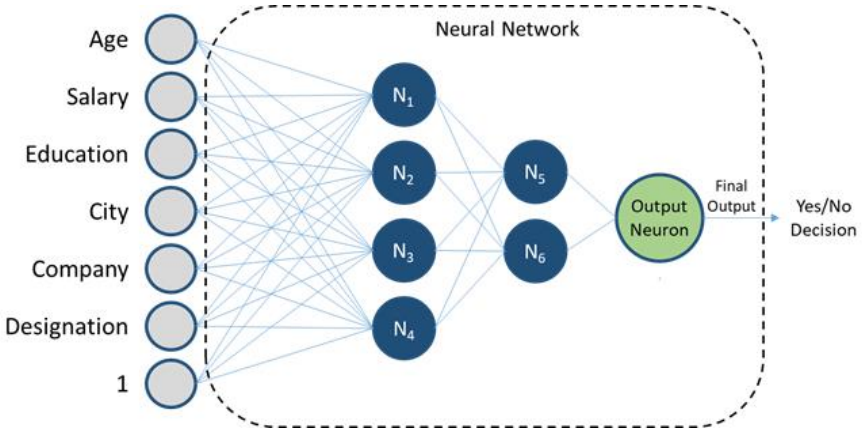



*Figure 55- Neural Network*

Neurons are coupled in three layers of a simple neural network as shown in Fig.56:

1. Input Layer

   The input layer is where data from the outside world enters an artificial neural network. Data is processed, analyzed, or categorized by input nodes before being sent to the following layer.

2. Hidden Layer

   One or more Hidden Layers are intermediate layers between the input and output layer. They process the data by applying complex non-linear functions to them. These layers are the key component that enables a neural network to learn complex tasks and achieve excellent performance.

3. Output Layer

   The output layer presents the overall outcome of the artificial neural network's data processing. It may include one or more nodes. In our system, we will use a binary (yes/no) system for drowsiness detection.

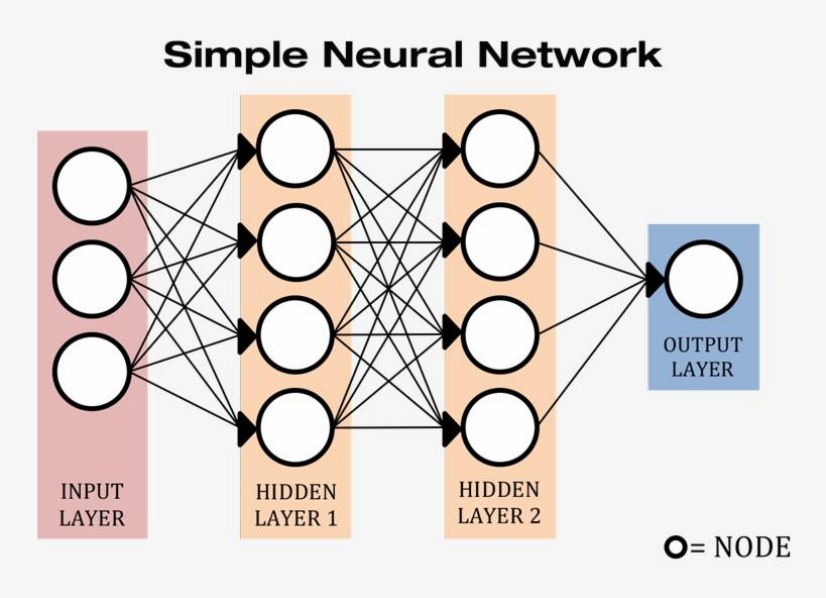



*Figure 56- Neural Network Layers*

In a neural network, multiple paths can be considered for translating the input node to the right output node. The neural network employs a feedback loop that functions as follows to determine the optimal path:

1. Each node along the path makes a prediction about the next node in line.

2. The network checks if the prediction was accurate. Pathways that lead to more accurate predictions are assigned higher weight values, while those resulting in inaccurate predictions are assigned lower weight values.

3. The network uses the backpropagation algorithm to propagate the error back through the network. This algorithm is illustrated in Fig.57. It calculates the gradient of the error with respect to the weights of each node, allowing the network to understand how much each node contributes to the overall error.

4. The network adjusts the weights and biases of each node using gradient descent. It updates them in the opposite direction of the gradient, aiming to minimize the error and improve the accuracy of predictions.

5. The nodes repeat Steps 1 to 4, making new predictions for the next data point using the updated weights and biases.

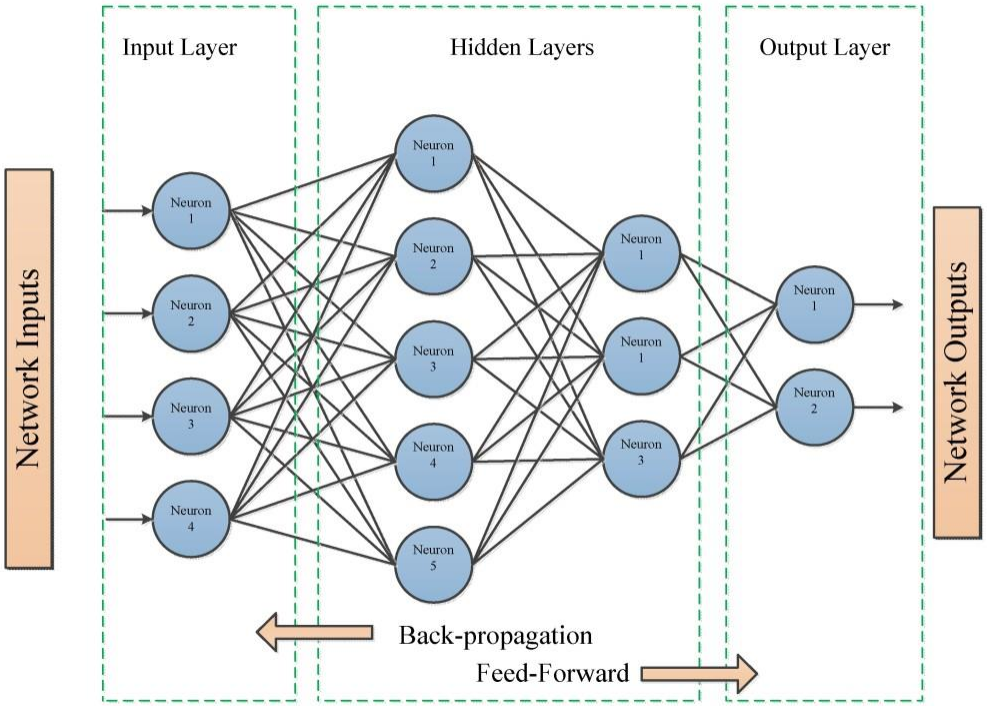



*Figure 57- Neural Network Back-Propagation*

By leveraging multiple layers, deep neural networks can capture complex patterns and relationships in the data, from low-level details to high-level abstractions. This makes deep learning particularly powerful in tasks that involve large amounts of data and complex patterns. Because EEG data is complex and requires advanced analysis, we will use a deep neural network called RNN to train it. RNNs are designed for sequential data like EEG, making them suitable for extracting patterns and features related to drowsiness.

### 6.1.4 Learning Rate

The learning rate is comparable to the model's "step size" during learning. A higher rate of learning results in larger steps, which can accelerate learning but also increase the likelihood of going beyond the ideal course of action. A slower but more accurate learning process might be achieved by taking smaller steps and learning at a slower rate. An eloquent illustration of this can be seen in Fig.58.

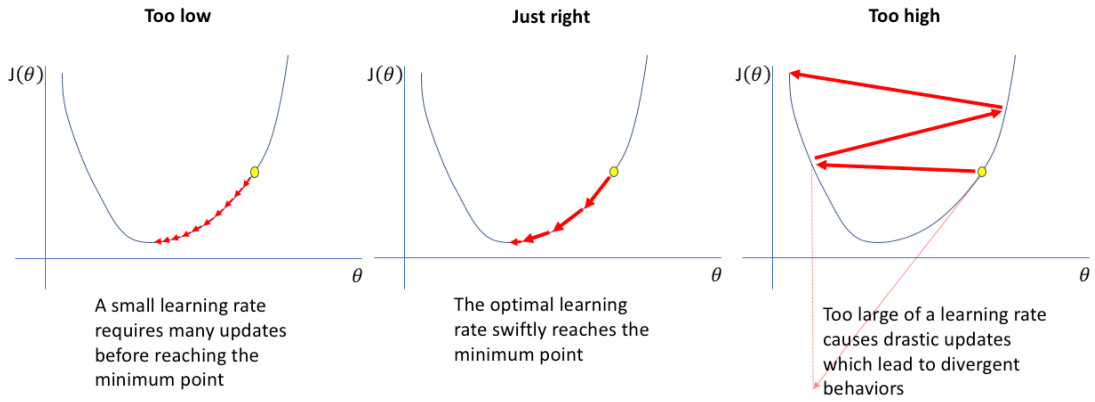



*Figure 58- Learning Rate*

### 6.1.5 Training Data vs Validation Data

To improve the model's performance during training, we optimize it using the training data. However, it's important to evaluate how well the model performs on new and untested data to avoid overfitting and ensure its effectiveness. This untested data, which we call validation data, consists of 15% of the original dataset that we use to validate and test our model.

## 6.2 Overview of the AI Code

Detecting drowsiness using EEG data from the NeuroSky MindWave device is a challenging task. To make it easier, a special type of model called bidirectional LSTM is used. This model looks at the EEG data in both directions, considering past and future information. It helps to find patterns and understand how the brain activity changes over time, which is important for detecting drowsiness. The bidirectional LSTM model is good at handling noisy data and can capture important features related to drowsiness. In this section, we will explain why bidirectional LSTM is helpful and how it improves the accuracy of drowsiness detection using EEG data from the MindWave device:

1. Temporal Dependencies and Pattern Recognition: By processing the EEG data in both forward and backward directions, the model can effectively capture the sequential nature of the data and learn the patterns associated with drowsiness. For example: a pattern of gradually decreasing delta and theta waves over time might indicate increasing drowsiness.

2. Capturing Past and Future Context: Bidirectional LSTM enables the model to access past and future context around each time step. By considering both the previous and subsequent patterns in the EEG data, the model can make more informed decisions about the current level of drowsiness. For instance, a sudden increase in beta waves may suggest a brief period of alertness, but if followed by a continuous decrease in beta waves, it could indicate drowsiness.

3. Robustness to Noise: The model can learn to focus on relevant patterns and reduce the influence of noisy or irrelevant fluctuations in the EEG data, leading to more robust drowsiness detection. Noise sources can include muscle artifacts or environmental interference.

## 6.3 Collected Dataset

Multiple subsets of a dataset were collected using the NeuroSky MindWave device. The dataset is divided into two sections, “awake” and “drowsy” state. Each data file within the dataset collected EEG data for 10 minutes. The EEG data consisted of 10 features: attention, meditation, delta, theta, low-beta, high-beta, low-alpha, high-alpha, low-gamma and high-gamma. The dataset contains data from 5 subjects, with a total of 160 minutes. Half of the data, 80 minutes, represents drowsiness, while the other 80 minutes represents the awake state.

The activities done to induce attentiveness included solving math problems, or playing mental math games using a phone. On the other hand, the activities done to induce drowsiness included watching a boring video, listening to white noise, or resting while closing your eyes. These activities were conducted in the late evening, near bedtime, or when an individual felt fatigued to ensure the accuracy of the dataset. The inclusion of diverse activities aimed to create variation within the dataset and facilitate easier learning for the model.

The dataset was gathered using the Neurosky Mindwave BCI by the Python software. The code collected ten features for a duration of ten minutes, including their timestamps. The data was then saved in an Excel file. Fig.59 shows the code for writing the data to a CSV file.

```
# Define a function to write data to the CSV file
def write_to_csv(attention, meditation, delta_power, theta_power, alpha_low_power,
                 alpha_high_power, beta_low_power, beta_high_power, gamma_low_power, gamma_high_power,blink):

    current_time = time.time()
    timestamp = time.strftime('%H:%M:%S', time.localtime(current_time))

    row = [timestamp, attention, meditation, delta_power, theta_power, alpha_low_power,
           alpha_high_power, beta_low_power, beta_high_power, gamma_low_power, gamma_high_power,blink]

    with open('eeg_data.csv', mode='a', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(row)
```

*Figure 59- Writing data to csv file*

## 6.4 Data Preprocessing and Feature Engineering

Firstly, data cleaning was done by removing rows with missing values and selecting attention and meditation values greater than 0.

We then normalized all 10 features to be from -1 to 1. This is done because features have significantly different scales, and algorithms may give more weight to features with larger values, leading to biased results. By normalizing the values, all features are on a level playing field, and their impact on the learning process is more balanced. This finalized normalized feature values are shown in Fig.60.

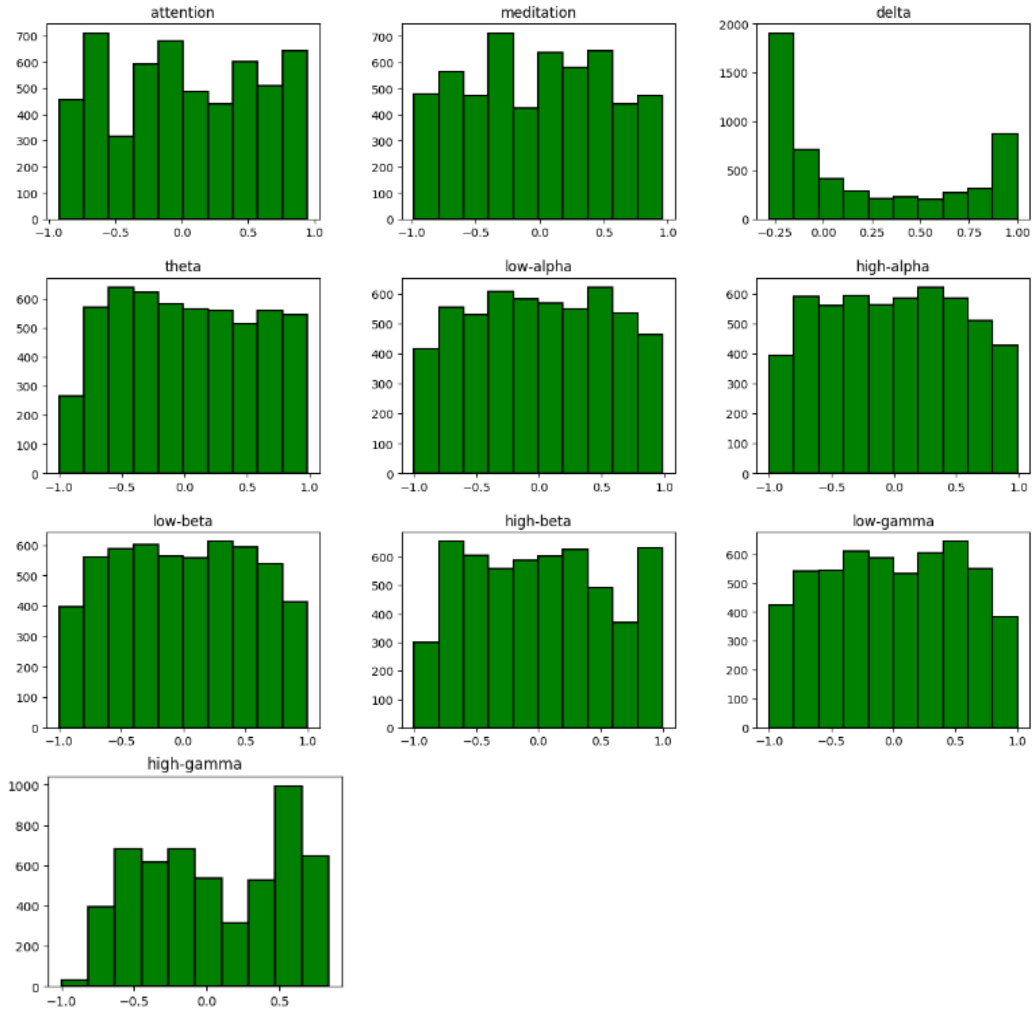



*Figure 60- AI features after normalization*

After that, we split our data into train and test sets, where 85% of our data became train data, and 15% became test data. Our labelled output data is an array of 0's and 1's, where 0 represents a drowsiness state, and 1 represents an awake state.

In the code, the original shape of the input data is (n_samples, 10), where n_samples represents the number of samples and 10 represents the number of features.
By reshaping the input data using np.array (x_train).reshape (-1, 2, 10) and np.array (x_test).reshape (-1, 2, 10), the new shape becomes (n_samples, 2, 10).

This new shape transformation allows us to represent each sample in the data as a sequence with two consecutive time steps. Each time step corresponds to a specific moment in time and contains 10 different features. By doing this, we can capture the temporal patterns and dependencies of drowsiness. The code snippet shown in Fig.61 demonstrates how this transformation is implemented.

```
if x_train.shape[0]-y_train.shape[0] != 0:
  y_train=y_train[:x_train.shape[0]-y_train.shape[0]]
y_train.shape
```

```
(4610,)
```

```
x_train.shape, x_test.shape,y_train.shape,y_test.shape
```

```
((4615, 10), (815, 10), (4615,), (815,))
```

```
x_train = np.array(x_train).reshape(-1,2,10)
x_train.shape
```

```
(2305, 2, 10)
```

```
x_test = np.array(x_test).reshape(-1,2,10)
x_test.shape
```

```
(405, 2, 10)
```

```
y_train = np.mean(np.array(y_train).reshape(-1, 2), axis=1).astype(int)
y_train.shape
```

```
(2305,)
```

```
y_test = np.mean(np.array(y_test).reshape(-1, 2), axis=1).astype(int)
y_test.shape
```

```
(405,)
```

```
x_train.shape, x_test.shape,y_train.shape,y_test.shape
```

```
((2305, 2, 10), (405, 2, 10), (2305,), (405,))
```

*Figure 61- AI preprocessing code*

We selected two time steps because the NeuroSky MindWave device has a slow frequency of 1Hz. To get a drowsiness detection result, a minimum waiting period of two time steps/seconds is required. In order for our system to respond quickly, we opted for the minimum number of time steps. If we were to use a different BCI device with a higher frequency range of 128Hz to 512Hz, we could utilize more time steps in order to increase our model's accuracy by capturing long-term dependencies. This would enable faster detection of the drowsiness state when testing real-time data.

## 6.5 Model Architecture

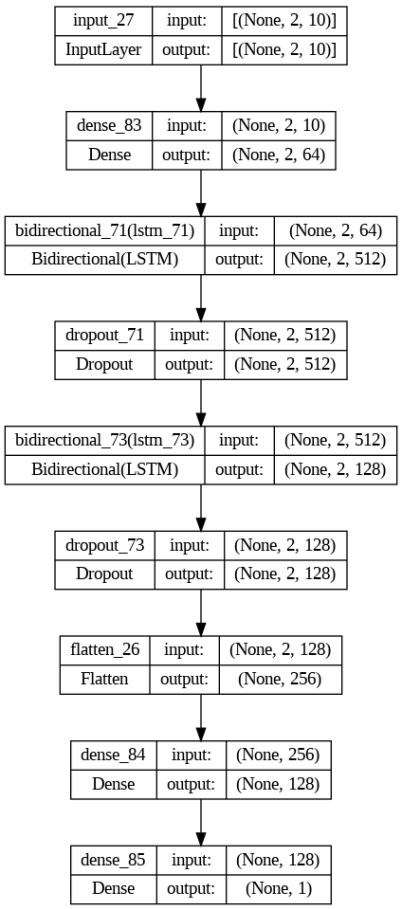



*Figure 62- AI Model Architecture*

The model architecture shown in Fig.62 consists of multiple layers. The first dense layer captures relevant features and introduces non linearity by uses the ReLu activation function. It also uses an L2 regularization technique in order to prevent overfitting.

This model is a stacked bidirectional LSTM model. The purpose of stacking LSTM layers is to capture increasingly abstract and high-level representations of the input data. The stacked LSTM architecture consists of three levels of LSTM layers.

The first level, an LSTM layer with 256 units, serves as the initial layer. It receives the input data and is responsible for learning low-level temporal patterns and features. With its high capacity, this layer is capable of capturing complex patterns and dependencies in the data.

The second level of the stacked LSTM architecture is an LSTM layer with 128 units. It takes the output of the first layer as its input and focuses on learning more abstract representations. By reducing its capacity compared to the previous layer, it aims to capture higher-level patterns and dependencies in the data.

The third and final level of the stacked LSTM architecture is an LSTM layer with 64 units. It takes the output of the second layer as its input and is responsible for learning the most abstract and high-level representations. With a further reduction in capacity, this layer aims to extract the most essential and relevant information from the previous layers, emphasizing the core features and patterns in the data.

After each the LSTM layer, Dropout layers are applied to randomly drop out a fraction of the LSTM units' outputs, forcing the subsequent layers to learn from the remaining units and preventing over-reliance on specific LSTM connections. This helps in reducing overfitting and improving the model's ability to generalize well to unseen data.

After the LSTM and Dropout layers, the Flatten layer is used to convert the sequential output of the LSTM layers into a one-dimensional vector suitable for the final classification or regression task performed by the output layer.

The next dense layer further processes and extracts relevant information from the flattened input. It learns and captures more complex patterns and relationships in the data, ultimately contributing to the model's ability to make accurate predictions.

Finally, the output layer is responsible for predicting whether a given input corresponds to drowsiness or not. Since it is a binary classification task (drowsy or not drowsy), the output layer consists of a single neuron. To obtain a probability value between 0 and 1, the sigmoid activation function is applied in the output layer. This allows the model to compress the output values and interpret them as the likelihood of drowsiness, where 0 represents high drowsiness probability and 1 represents low drowsiness probability. Once trained, the model can take real time data from the Neurosky Mindwave device and predict a probability value between 0 and 1, indicating the likelihood of drowsiness.

## 6.6 Model Training

```
def train_model(model,x_train, y_train,x_test,y_test, save_to, epoch = 2):

        opt_adam = keras.optimizers.Adam(learning_rate=0.001)

        es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)
        mc = ModelCheckpoint(save_to + '_best_model.h5', monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)
        lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 0.001 * np.exp(-epoch / 10.))

        model.compile(optimizer=opt_adam,
                  loss=['binary_crossentropy'],
                  metrics=['accuracy'])

        history = model.fit(x_train,y_train,
                        batch_size=20,
                        epochs=epoch,
                        validation_data=(x_test,y_test),
                        callbacks=[es,mc,lr_schedule])

        saved_model = load_model(save_to + '_best_model.h5')

        return model,history
```

*Figure 63- Training AI Model Code*

The function shown in Fig. 63 is used to train a model using the provided training and testing data. It applies various callbacks and settings to enhance the training process.

### 6.6.1 Callbacks

- EarlyStopping: This callback monitors the validation loss (val_loss) during training. If the loss does not improve for 10 consecutive epochs, training is stopped early. It helps prevent overfitting and saves computational resources by stopping the training when the model's performance is no longer improving.

- ModelCheckpoint: This callback saves the best model based on the validation accuracy (val_accuracy) during training. The best model is saved as "_best_model.h5"". This model will be later used to predict the drowsiness state of the real-time data coming from the Neurosky Mindwave.

- LearningRateScheduler: This callback adjusts the learning rate during training. It decreases the learning rate exponentially with each epoch using a lambda function.

### 6.6.2 Model Compilation

The model is compiled with the following settings:

- Optimizer: Adam optimizer from Keras with a learning rate of 0.001.
- Loss Function: Binary cross-entropy, suitable for binary classification problems. It measures the dissimilarity between the predicted probability distribution and the true distribution of the target variable.
- Metrics: Accuracy, used to evaluate the model's performance.

### 6.6.3 Training

The model is trained and validated using the fit() method at 20 samples at a time.

## 6.7 Evaluations and Results

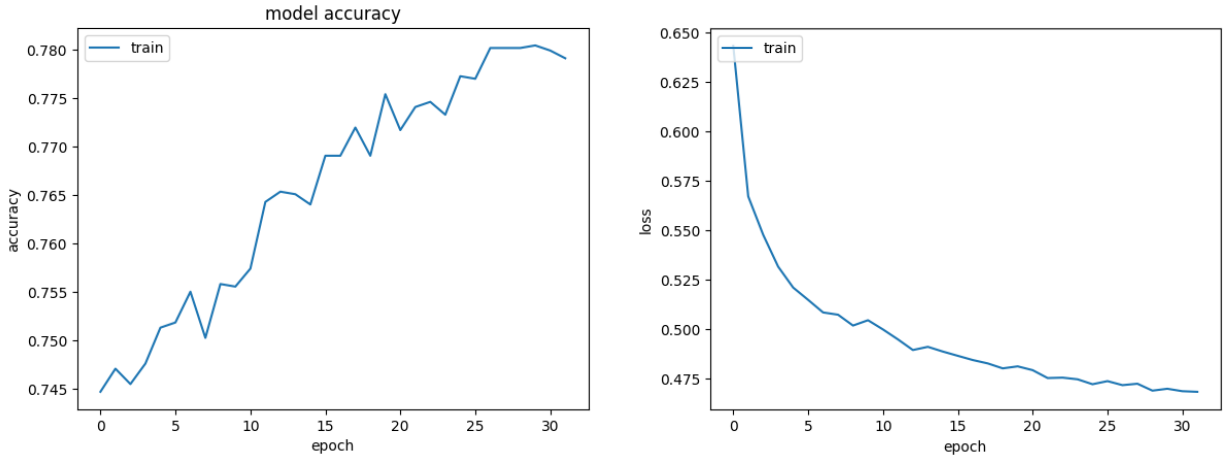



*Figure 64- AI Model Results (Accuracy and Loss)*

```
max_accuracy = max(history.history['accuracy'])
max_accuracy = round(max_accuracy*100, 3)
max_val_accuracy = max(history.history['val_accuracy'])
max_val_accuracy = round(max_val_accuracy*100, 3)
print("Maximum accuracy:", max_accuracy,"%")
print("Maximum validation accuracy:", max_val_accuracy,"%")
```

```
Maximum accuracy: 78.042 %
Maximum validation accuracy: 75.94 %
```

*Figure 65- AI Model results (Maximum Accuracy and Validation Accuracy)*

In the given results shown in Fig.64 and Fig.65, the training accuracy reached 78.042%, which is relatively high and suggests that the model is performing well on the training data. Moreover, the validation accuracy reached a maximum value of 75.94%. This suggests that the model performs well on the validation data, correctly predicting the target variable.

## 6.8 External Libraries and Dependencies

1. TensorFlow
2. Scikit-learn
3. NumPy
4. SciPy
5. Pandas
6. Matplotlib
7. Keras

# Chapter 7: Integration of V2V and BCI Systems

We integrate our components to obtain real-time location and speed information when the driver is drowsy. The information is transmitted through an RF module from the first car (drowsy driver's car) to the second car (nearby car).

The GPS, speed, and drowsiness state information will be sent asynchronously using a packet format. The purpose of using a packet format is to organize the data in a structured manner so that it can be transmitted and easily and correctly parsed by the receiving device or system. The packet format includes a header that specifies the start of the packet and provides information about the data contained within it.

The prototype that we built consists of two cars, one representing the drowsy driver, and the other representing the nearby car. The car in Fig.66 will act as the drowsy driver's car, while the car in Fig.67 will act as the nearby car. The first car is equipped with the GPS module, the speed detection modules, and the RF module which will be transmitting both of these information. It also includes an Arduino Uno which will be connected to the PC in order to receive the drowsiness state and transmit it through the RF Module. The second car is equipped with an RF module which will act as a receiver for all of these information. Furthermore, it contains an LCD to display the GPS location and 4 LEDs to express the speed of the first car. It also contains an ultrasonic sensor positioned at the front in order to detect any cars in close range.

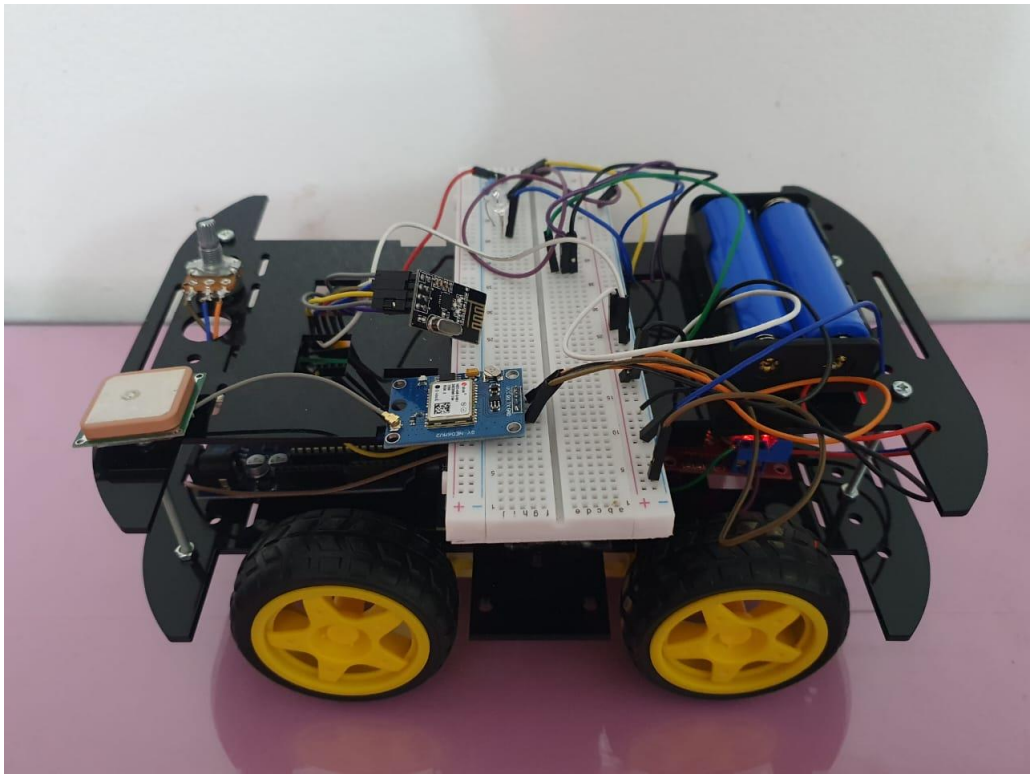


*Figure 66- Drowsy Driver's Car Prototype*

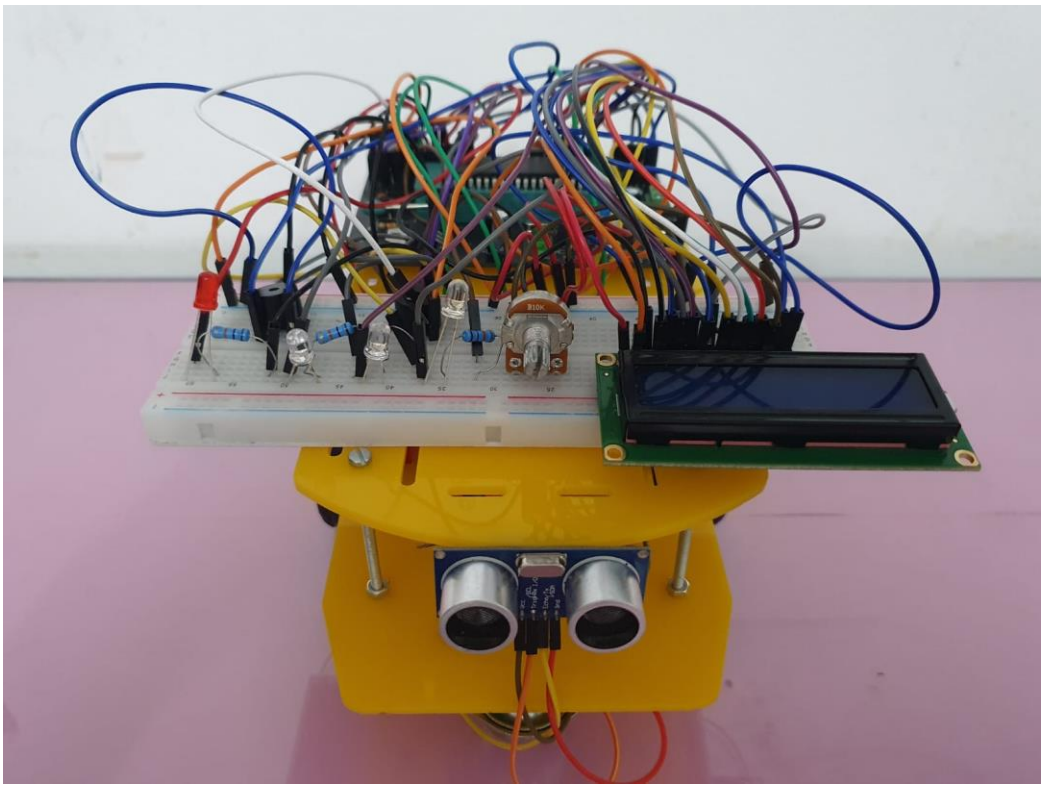


*Figure 67- Nearby Car Prototype*

## 7.1 Sending GPS Location

The GPS location is acquired in real time and displayed on an LCD screen for easy visibility. Any change in the GPS location of the first car is automatically updated on the LCD of the second car.

### Transmitter Code

```
while(1) {
    /* we separated the uart and spi protocols because they interfere with each other*/
    usart_init();
    get_latlong(lati_value,&lati_dir,longi_value,&longi_dir);
    usart_disable();
    _delay_ms(100);

    sei();

    nrf_init(NRF_MODE_PTX, addr);  //Set nRF to transmitter mode
    sprintf(packet,"LAV%s%c",lati_value,lati_dir);

    if(nrf_transmit_packet(packet, sizeof(packet)) != 0) {  //If packet not successfully transmitted
                    _delay_ms(50);
                    RED_LED_ON();    //Blink LED
                    _delay_ms(50);
                    RED_LED_OFF();   //Blink LED
                }

    _delay_ms(100);

    sprintf(packet,"LOV%s%c",longi_value,longi_dir);
    if(nrf_transmit_packet(packet, sizeof(packet)) != 0) {  //If packet not successfully transmitted
                        _delay_ms(50);
                        RED_LED_ON();    //Blink LED
                        _delay_ms(50);
                        RED_LED_OFF();   //Blink LED
                    }

    _delay_ms(100);
```

*Figure 68- Transmitting GPS Code*

In the code shown in Fig.68, the longitude is sent in a packet with a header “LAV” followed by the latitude value and direction. The same thing is done for the longitude with the header “LOV”.

Note: During Integration, we noticed that the UART protocol used by the GPS module interferes with the SPI protocol used by the NRF module. Therefore, we separated the protocols to prevent them from working simultaneously.

## Receiver Code

```
nrf_receive_packet(rx_buf, &length);    //Receive string in rx_buf
```

*Figure 69- NRF24L01+ Receiving Data*

From Fig.69, we can see that the receiver continuously receives packets in a while (1) loop and stores it in a buffer. The information in the packet is then parsed to extract relevant information.

```
if(rx_buf[0]=='L'&&rx_buf[1]=='A'&&rx_buf[2]=='V')
{
    for(i=0;i<=11;i++)
    {
        lati_value[i]=rx_buf[i+3];
    }
}

if(rx_buf[0]=='L'&&rx_buf[1]=='O'&&rx_buf[2]=='V')
        {
            for(i=0;i<=12;i++)
            {
                longi_value[i]=rx_buf[i+3];
            }
        }


            LCD_vidSendCommand(0x80);
            LCD_vidWriteStr(lati_value);

            LCD_vidSendCommand(0xC0);
            LCD_vidWriteStr(longi_value);
```

*Figure 70- Displaying GPS Code*

In the code shown in Fig.70 we check if the received buffer contains the header “LAV” or “LOV” which contain the latitude and longitude location. If found, the information is extracted and stored in an array. Afterwards, the information is displayed on the LCD screen.

## 7.2 Sending Speed Information

We incorporated a potentiometer to control and vary the speed of the first car. The speed value is then transmitted to the second car using the RF module. To visually represent the revolutions per second (RPS), we utilized four different LEDs with the following indications:

- 0 RPS: The first LED is illuminated.
- 1 RPS: The second LED is illuminated.
- 2 RPS: The third LED is illuminated.
- 3 RPS: The fourth LED is illuminated.

At each state, only a single LED is illuminated, while the others are turned off.

### Transmitter Code

```
sprintf((char *)tx_buf, "rps%u", rps); // format the integer and store in buffer

if(nrf_transmit_packet(tx_buf, sizeof(tx_buf)) != 0) {  //If packet not successfully transmitted
        _delay_ms(50);
        RED_LED_ON();    //Blink LED
        _delay_ms(50);
        RED_LED_OFF();   //Blink LED
    }

_delay_ms(100);
```

*Figure 71- Transmitting Speed Code*

In the code shown in Fig.71, the RPS is sent in a packet with a header “rps” followed by the rps value.

**Receiver Code**

```
if(rx_buf[0]=='r'&&rx_buf[1]=='p'&&rx_buf[2]=='s')
{
    if(rx_buf[3]=='0')
    {
        DIO_vidSetPinVal(DIO_PORTA, DIO_PIN0,DIO_HIGH);
        DIO_vidSetPinVal(DIO_PORTA, DIO_PIN1,DIO_LOW);
        DIO_vidSetPinVal(DIO_PORTA, DIO_PIN2,DIO_LOW);
        DIO_vidSetPinVal(DIO_PORTA, DIO_PIN3,DIO_LOW);
    }
    else if(rx_buf[3]=='1')
    {
        DIO_vidSetPinVal(DIO_PORTA, DIO_PIN0,DIO_LOW);
        DIO_vidSetPinVal(DIO_PORTA, DIO_PIN1,DIO_HIGH);
        DIO_vidSetPinVal(DIO_PORTA, DIO_PIN2,DIO_LOW);
        DIO_vidSetPinVal(DIO_PORTA, DIO_PIN3,DIO_LOW);
    }
    else if(rx_buf[3]=='2')
    {
        DIO_vidSetPinVal(DIO_PORTA, DIO_PIN0,DIO_LOW);
        DIO_vidSetPinVal(DIO_PORTA, DIO_PIN1,DIO_LOW);
        DIO_vidSetPinVal(DIO_PORTA, DIO_PIN2,DIO_HIGH);
        DIO_vidSetPinVal(DIO_PORTA, DIO_PIN3,DIO_LOW);
    }
    else if(rx_buf[3]=='3')
            {
                DIO_vidSetPinVal(DIO_PORTA, DIO_PIN0,DIO_LOW);
                DIO_vidSetPinVal(DIO_PORTA, DIO_PIN1,DIO_LOW);
                DIO_vidSetPinVal(DIO_PORTA, DIO_PIN2,DIO_LOW);
                DIO_vidSetPinVal(DIO_PORTA, DIO_PIN3,DIO_HIGH);
            }
}
```

*Figure 72- Receiving Speed Code*

In the code shown in Fig.72, we check if the received buffer contains the header “rps”. If found, we then check for the value of the revolutions per second, and adjust the LEDs based on the speed.

- The first led is configured to take its input from PORTA, PIN0
- The second led is configured to take its input from PORTA, PIN1
- The third led is configured to take its input from PORTA, PIN2
- The fourth led is configured to take its input from PORTA, PIN3

## 7.3 Sending Drowsiness State

If the headset detects drowsiness in the driver, a signal is sent to the nearby car, alerting them using a buzzer.

### 7.3.1 Collecting Real-Time Data from the BCI

```
# Initialize an empty buffer to store the data
buffer = []
ser = serial.Serial('COM9', 4800)  # Change the port name and baud rate to match your setup
model = tf.keras.models.load_model(r'C:\Users\yasmi\OneDrive\Desktop\_best_model.h5')

attention = 0
meditation = 0
delta_power = 0
theta_power = 0
alpha_low_power=0
alpha_high_power=0
beta_low_power=0
beta_high_power=0
gamma_low_power=0
gamma_high_power=0
notworn=0
signal_quality=0

connection = tel.Telnet('127.0.0.1', 13854)  # connect to TGC local host
connection.write(b'{"eSense": true, "format": "Json"}')  # activate eSense value

freq_bands = {'delta': (0.5, 4),
              'theta': (4, 8),
              'low_alpha': (8, 10),
              'high_alpha': (10, 12),
              'low_beta': (12, 16),
              'high_beta': (16, 20),
              'low_gamma': (20, 30),
              'high_gamma': (30, 40)}
```

*Figure 73- Feeding Real Time EEG Signals Initialization Code*

At the beginning of the code shown in Fig.73, we created an array called “buffer” to store two time steps of features before inputting it into the AI model. We then established a serial communication with COM Port 9 with a baud rate of 4800 which will be used to send the drowsiness state to the Arduino Uno.

As well as, we loaded the model weights that were obtained from training our dataset which will be used to make a prediction on real time data. Global variables were also created to store all 10 features, including attention, meditation, delta power, theta power, alpha low power, alpha high power, beta low power, beta high power, gamma low power, and gamma high power. Additionally, a variable named "notworn" was created to indicate whether the person is not wearing the headset. These global variables will be used in the code to track and manipulate the relevant data.

Furthermore, we established a Telnet connection between a computer and a NeuroSky MindWave EEG headset using the TGC software. The at IP address 127.0.0.1 which represents the local machine was used to make the connection. Moreover, the port number 13854 is the port used by the Think Gear Connector software to listen for incoming connections from applications that want to access the EEG data collected by the NeuroSky MindWave headset.

We then activated the eSense values which contains the attention and meditation levels. We also defined the frequency levels that would need to be extracted from the incoming BCI data.

```
# Define a function to get the data
def get_data(connection):
    global attention, meditation, delta_power, theta_power, alpha_low_power,alpha_high_power
    global beta_low_power, beta_high_power, gamma_low_power, gamma_high_power, notworn, signal_quality
    line = connection.read_until(b'\r')
    while True:
        line = connection.read_until(b'\r')
        raw_str = (str(line).rstrip("\\r'").lstrip("b'"))
        data = json.loads(str(raw_str))

        if "eSense" in data.keys():  #if eSense found
            attention=int(data['eSense']["attention"])  #get attention
            meditation=int(data['eSense']['meditation'])  #get meditation
        if "eegPower" in data.keys():
            eeg_power = data['eegPower']
            delta_power = eeg_power['delta']
            theta_power = eeg_power['theta']
            alpha_low_power = eeg_power['lowAlpha']
            alpha_high_power = eeg_power['highAlpha']
            beta_low_power = eeg_power['lowBeta']
            beta_high_power = eeg_power['highBeta']
            gamma_low_power = eeg_power['lowGamma']
            gamma_high_power = eeg_power['highGamma']
        if "poorSignalLevel" in data.keys():
            signal_quality = int(data['poorSignalLevel'])
            if signal_quality == 200:
                notworn=1
                #print("Signal quality: Electrodes not contacting the skin")
            else:
                notworn=0
                #print("Signal quality: Excessive noise or interference or poor signal")
```

*Figure 74- Feeding Real Time EEG Signals Extracting Features Code*

The second part of the code shown in Fig.74 reads and extracts the data from the BCI. The function “get_data” starts by reading a line of data from the EEG headset using the read_until() method of the Telnet connection object. The argument b'\r' specifies the delimiter to use for reading data. In this case, the function is set up to read one line of data at a time.

The function then enters an infinite loop, reading lines of data from the EEG headset and processing them as they come in. Inside the loop, the raw data is cleaned up using the rstrip() and lstrip() methods, which remove any unnecessary characters from the string. This is necessary because the data is transmitted in a binary format and needs to be converted to a string for processing. The cleaned up data is then converted to a Python dictionary using the json.loads() method, which parses the JSON-formatted data into a dictionary that can be accessed using key-value pairs.

After that, the function checks if the 'eSense' key is present in the data dictionary using the if "eSense" in data.keys(): statement. If the 'eSense' key is present, the attention and meditation values are extracted from the dictionary and stored in the attention and meditation variables respectively. The function does the same thing by checking if ‘eegPower’ is present in the data dictionary, if it is present, all 8 frequencies are extracted into their variables.

As mentioned earlier in chapter 3, the 'poorSignalLevel' key can be utilized to determine if the person is not wearing the headset. We will leverage this feature because even if the person is not wearing the headset, the BCI still produces noisy data. It is crucial to prevent the model from classifying drowsiness in such cases to maintain accuracy and reliability. If the person is not wearing the headset, the global variable “notworn” is set to 1. This variable will serve as a condition later in the code.

```
input_data = np.array([[attention, meditation, delta_power, theta_power, alpha_low_power,
                        alpha_high_power, beta_low_power, beta_high_power, gamma_low_power,
                        gamma_high_power]])


input_data = pd.DataFrame(input_data, columns=['attention', 'meditation', 'delta_power', 'theta_power',
                                               'alpha_low_power', 'alpha_high_power', 'beta_low_power',
                                               'beta_high_power', 'gamma_low_power', 'gamma_high_power'])

input_data.iloc[0, 0:10] = np.log2(input_data.iloc[0, 0:10] + 0.0001)  # log normalization for the first row

md = np.median(input_data.iloc[0, 0:10])
p75 = np.percentile(input_data.iloc[0, 0:10], 75)
p25 = np.percentile(input_data.iloc[0, 0:10], 25)

input_data.iloc[0, 0:10] = np.tanh((input_data.iloc[0, 0:10] - md) / (p75 - p25))

input_data = input_data.values  # convert back to NumPy array

buffer.append(input_data)

if len(buffer) == 2:
    input_data = buffer
    buffer=[]
    input_data = input_data.reshape(1, 2, 10)

    prediction = model.predict(input_data)
    print(prediction)
```

*Figure 75-Feeding Real Time EEG Signals Preprocessing and Prediction Code*

Fig.75 shows all 10 features getting extracted and stored in an array called "input_data". After that, all the preprocessing steps that were performed on the dataset will also be applied to this data. The input data will then be appended to an array called "buffer" until it gets reaches two rows/time steps of data. If this condition is satisfied, the "buffer" array is cleared to be prepared to accept the next two time steps of data. The data is then reshaped to match the format used during training of the LSTM model, which is one sample, two time steps and ten features (1, 2, 10). This reshaped data is then fed into the model, which predicts a value between 0 and 1 indicating the level of wakefulness of the person.

```
threshold = 0.2

if notworn==1:
    print("Headset Not Worn")
    data = '2'  # string and add a newline character
    ser.write(data.encode('utf-8'))  # Encode the string using UTF-8 and send it


elif prediction <= threshold:
    print("Drowsy")
    data = '1'  # string and add a newline character
    ser.write(data.encode('utf-8'))  # Encode the string using UTF-8 and send it


elif prediction > threshold:
    print("Not Drowsy")
    data = '2'  # string and add a newline character
    ser.write(data.encode('utf-8'))  # Encode the string using UTF-8 and send it
```

*Figure 76- Python-Arduino Serial Transfer of Drowsiness State*

The code in Fig.76 shows the prediction output being compared with a predetermined threshold. If it is <= than the threshold, the person is classified as being drowsy, if it is > than the threshold, the person is classified as being awake. We will choose the threshold to be 20%. This means that if the model predicts that the percentage of wakefulness of a person is less than 20%, it will warn nearby cars. We chose a low threshold to reduce the false positive results, which is predicting sleepiness when the person is in an alert state.

However, if the variable “notworn”=1, which indicates that the person is not wearing the headset, the person is instantly classified as being not drowsy.
The data is then serially sent to an Arduino Uno using a COM port. This is done to pass the data to the AVR microcontroller in order to be transferred to the other controller using the RF module.

If the person is drowsy, the value "1" is transmitted to the Arduino Uno microcontroller. Otherwise, if the person is not drowsy, the value "2" is sent.

**Imported Libraries**

- telnetlib as tel
- tensorflow.compat.v1 as tf
- numpy as np
- pandas as pd
- serial

### 7.3.2 Arduino Transmitter Code

```
void setup() {
  Serial.begin(4800); // initialize serial communication at 9600 baud
  pinMode(13, OUTPUT);
}

void loop() {
  if (Serial.available()) { // check if data is available to read
    int value = Serial.read(); // read the integer value from the serial port

    if (value =='2') {
      digitalWrite(13, HIGH);
      delayMicroseconds(1000);
    }
    else if(value=='1')
      digitalWrite(13, LOW);
      delayMicroseconds(1000);
    }
  }
```

*Figure 77- Python-Arduino Serial Receiving of Drowsiness State*

This Arduino code shown in Fig.77 serially receives the drowsiness state through python. If the driver is drowsy, PIN13 in the Arduino Uno is set to LOW. Consequently, if the driver is not drowsy, PIN13 is set to HIGH.
To send the status of the driver from the Arduino to the AVR microcontroller representing the drowsy driver, PIN13 in the Arduino Uno is connected to PORTB, PIN1 in the AVR microcontroller, and the grounds of both the microcontrollers are made common.

### 7.3.3 AVR Transmitter Code

```
u8 x;
x=DIO_u8GetPinVal(DIO_PORTB,DIO_PIN1);

sprintf((char *)tx_buf, "St:%u", x); // format the integer and store in buffer
if(nrf_transmit_packet(tx_buf, sizeof(tx_buf)) != 0) {  //If packet not successfully transmitted
                _delay_ms(50);
                RED_LED_ON();    //Blink LED
                _delay_ms(50);
                RED_LED_OFF();   //Blink LED
            }

_delay_ms(100);

SPI_disable();
```

*Figure 78- AVR Transmitter Code for Drowsiness State*

Fig.78 shows the AVR code used to transmit the drowsiness state from the drowsy driver's car to the nearby car. A variable called "x" is used to read the status of the drowsiness from PORTB, PIN1. This value is then sent in a packet with the header "St:" followed by the value read.

### 7.3.4 AVR Receiver Code

```
if(rx_buf[0]=='S'&&rx_buf[1]=='t'&&rx_buf[2]==':')
                {
                    if(rx_buf[3]=='0')
                    {
                        DIO_vidSetPinVal(DIO_PORTB,DIO_PIN3,DIO_HIGH);
                    }


                    else if(rx_buf[3]=='1')
                    {
                        DIO_vidSetPinVal(DIO_PORTB,DIO_PIN3,DIO_LOW);
                    }
                }
```

*Figure 79- AVR Receiving Code for Drowsiness State*

Fig.79 shows the code used to receive the drowsiness state from the nearby car. The packet is extracted by identifying the header "St:". If the payload value is '0', indicating that the driver is drowsy, the buzzer is activated. Otherwise, if the payload value is '1', indicating that the driver is awake, the buzzer is turned off. The buzzer is connected to PORTB, PIN3.

## 7.4 Ultrasonic Module

In addition, we incorporated an ultrasonic sensor in the second car to measure distances between vehicles. If the sensor detects that the distance falls below a predetermined threshold, indicating that the cars are too close to each other, a buzzer is activated as an alert mechanism specifically designed for the drowsy driver. This alert serves as a timely reminder to maintain a safe distance and avoid potential collisions or accidents**.**

# Chapter 8: Results

The chapter discusses the results obtained for GPS tracking, speed calculation, drowsiness detection, and ultrasonic module testing. These results are all obtained at the same time.

## 8.1 GPS Results

We conducted a test to observe real-time GPS module performance. The two cars were positioned next to each other, and then we moved the car equipped with the GPS module away from the other car. We were able to see the immediate change in the GPS location on the LCD display as seen in Fig.80 and Fig.81.

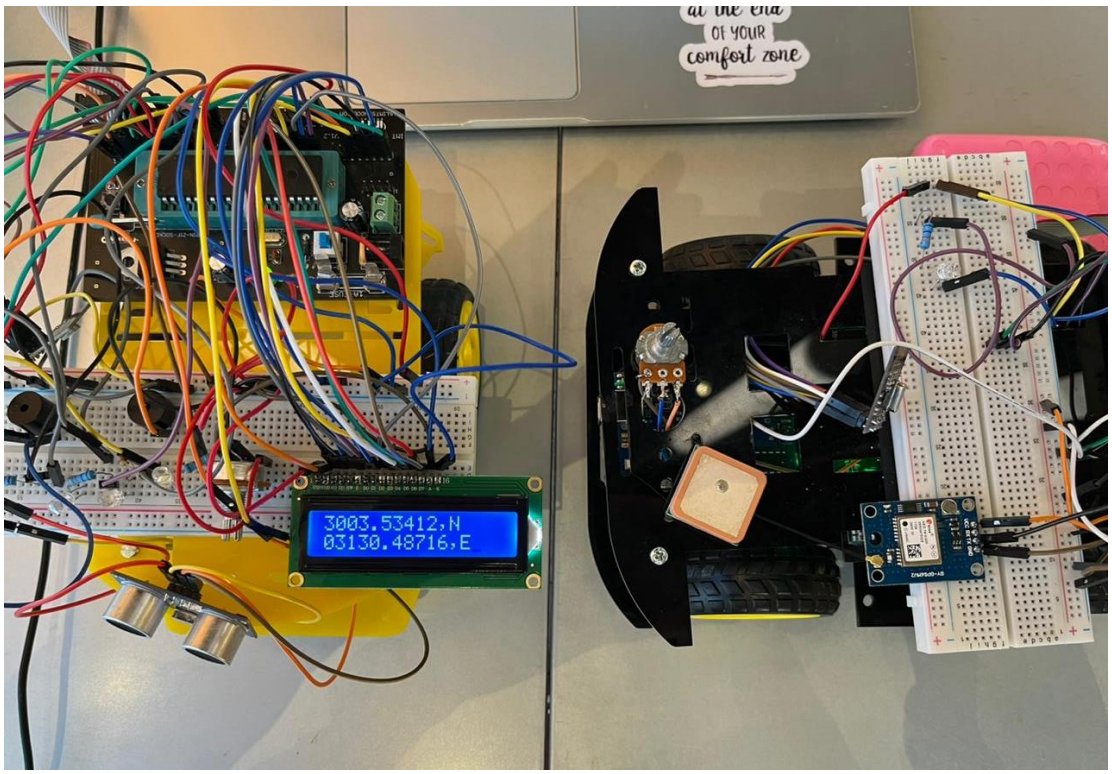



*Figure 80- GPS Results 1*

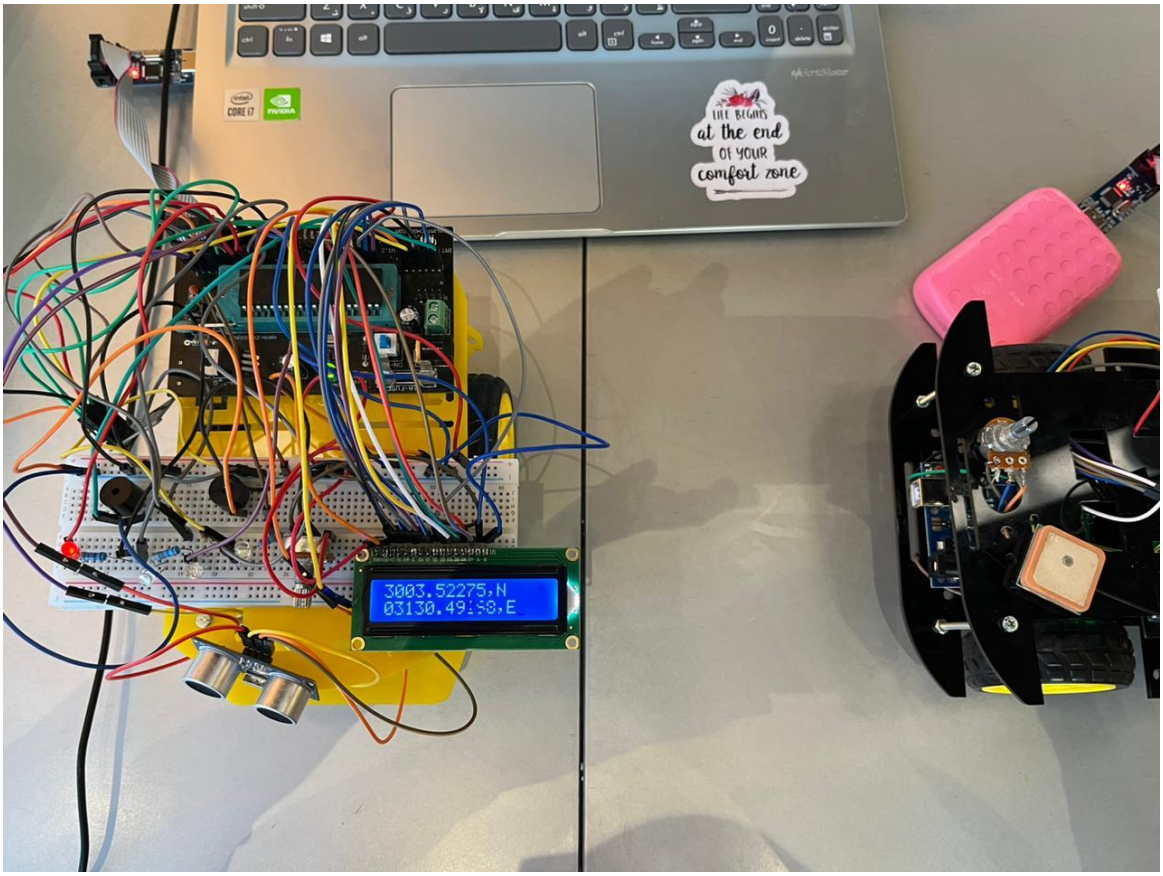



*Figure 81- GPS Results 2*

## 8.2 Speed Calculation Results

The results demonstrate how the color of the LEDs changes depending on the speed of the wheel.

- The Red LED represents 0 RPS.
- The Blue LED represents 1 RPS.
- The Yellow LED represents 2 RPS.
- The White LED represents 3 RPS.

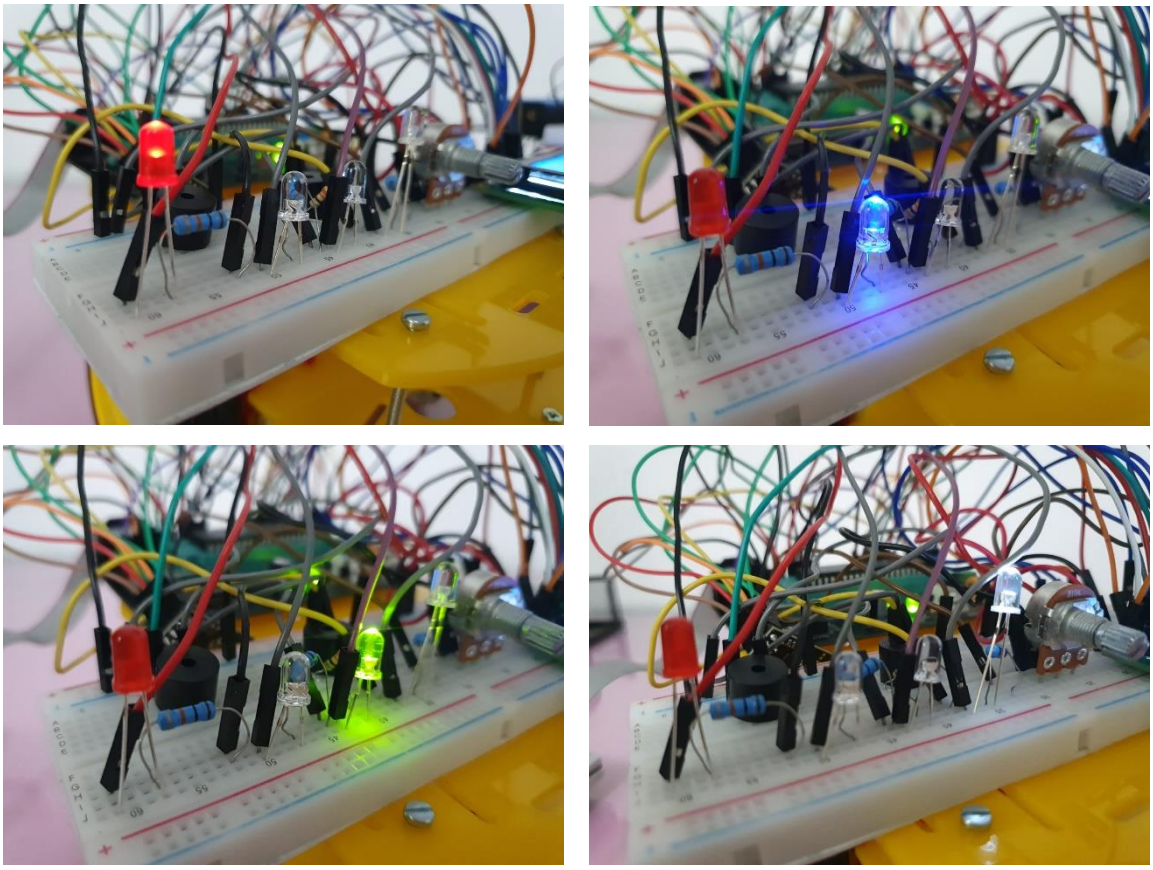


*Figure 82- Speed Calculation Results*

## 8.3 Drowsiness Detection Results

We conducted a test to determine if the RF module could send drowsiness alerts when a person is feeling drowsy. In order to accomplish this, we established a threshold of 0.9 as shown in the figure below. Any value below 90% wakefulness is classified as drowsy, which allows us to obtain a majority of results indicating drowsiness in the person. Our objective was to consistently receive drowsiness alerts, which would trigger a buzzer through an RF module, thereby alerting Car 2 from Car 1. The modified code is shown in Fig.83.

```
threshold = 0.9

if notworn==1:
    print("Headset Not Worn")
    data = '2'  # string and add a newline character
    ser.write(data.encode('utf-8'))  # Encode the string using UTF-8 and send it


elif prediction <= threshold:
    print("Drowsy")
    data = '1'  # string and add a newline character
    ser.write(data.encode('utf-8'))  # Encode the string using UTF-8 and send it


elif prediction > threshold:
    print("Not Drowsy")
    data = '2'  # string and add a newline character
    ser.write(data.encode('utf-8'))  # Encode the string using UTF-8 and send it
```

*Figure 83- Testing Drowsiness State with Threshold=0.9*

Firstly, we took off the helmet in order to test the non-drowsy state as shown in Fig.84. The buzzer was switched off.

```
1/1 [==============================] - ETA: 0s 
1/1 [==============================] - 0s 27ms/step
Headset Not Worn
1/1 [==============================] - ETA: 0s 
1/1 [==============================] - 0s 45ms/step
Headset Not Worn
1/1 [==============================] - ETA: 0s 
1/1 [==============================] - 0s 22ms/step
Headset Not Worn
1/1 [==============================] - ETA: 0s 
1/1 [==============================] - 0s 80ms/step
Headset Not Worn
1/1 [==============================] - ETA: 0s 
1/1 [==============================] - 0s 18ms/step
Headset Not Worn
1/1 [==============================] - ETA: 0s 
1/1 [==============================] - 0s 26ms/step
Headset Not Worn
1/1 [==============================] - ETA: 0s 
1/1 [==============================] - 0s 35ms/step
Headset Not Worn
```

*Figure 84- Helmet Not Worn*

Then, we wore the helmet in order to test the drowsy state as shown in Fig.85.

```
1/1 [==============================] - 0s 53ms/step
Drowsy
1/1 [==============================] - ETA: 0s
1/1 [==============================] - 0s 24ms/step
Drowsy
1/1 [==============================] - ETA: 0s
1/1 [==============================] - 0s 70ms/step
Drowsy
1/1 [==============================] - ETA: 0s
1/1 [==============================] - 0s 49ms/step
Drowsy
1/1 [==============================] - ETA: 0s
1/1 [==============================] - 0s 83ms/step
Drowsy
1/1 [==============================] - ETA: 0s
1/1 [==============================] - 0s 45ms/step
Drowsy
1/1 [==============================] - ETA: 0s
1/1 [==============================] - 0s 58ms/step
Drowsy
```

*Figure 85- Drowsy State Detected*

Once the second car received an alert indicating that the person is drowsy, the buzzer shown in Figure 86 was activated.

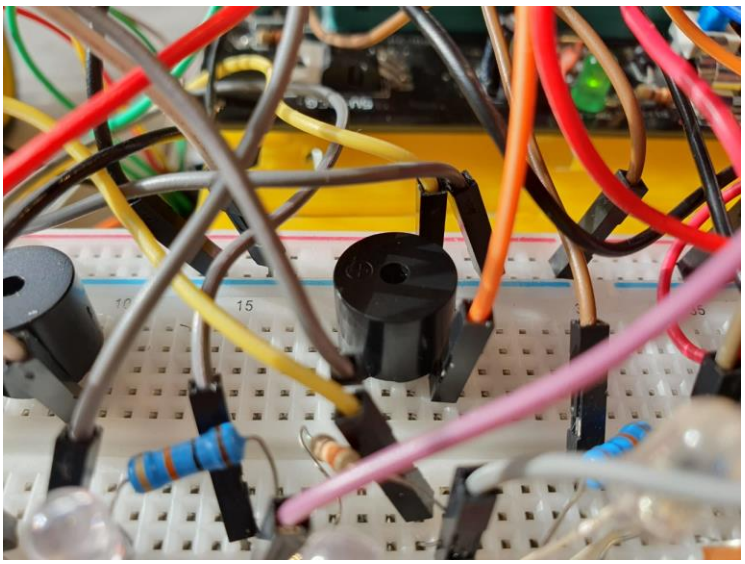


*Figure 86- Drowsiness State Buzzer On*

## 8.4 Ultrasonic Module Results

The buzzer is activated when the distance between the two cars becomes less than 15 centimeters. In Figure 87, the buzzer is off as the distance is larger than this threshold.

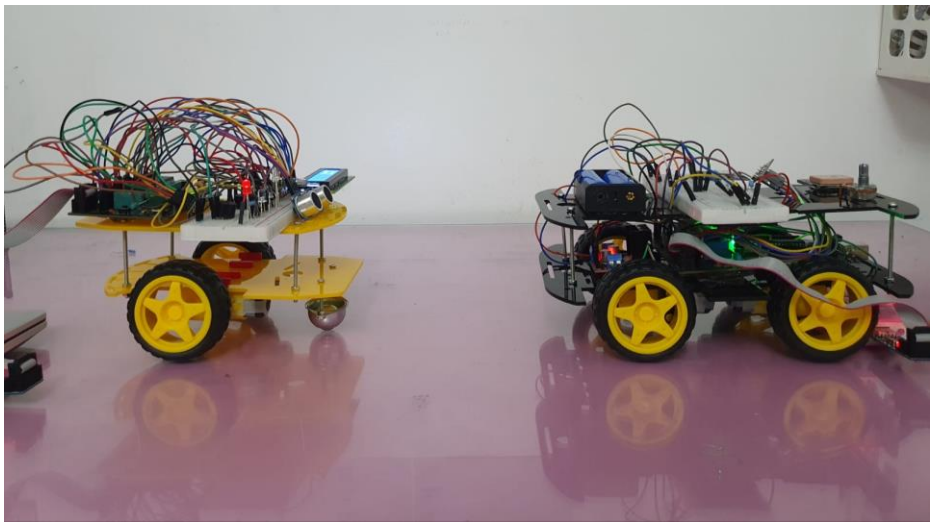


*Figure 87- Ultrasonic Results- Buzzer Off*

In Figure 88, we can see that the distance between the two cars decreased below 15 centimeters which activated the buzzer.

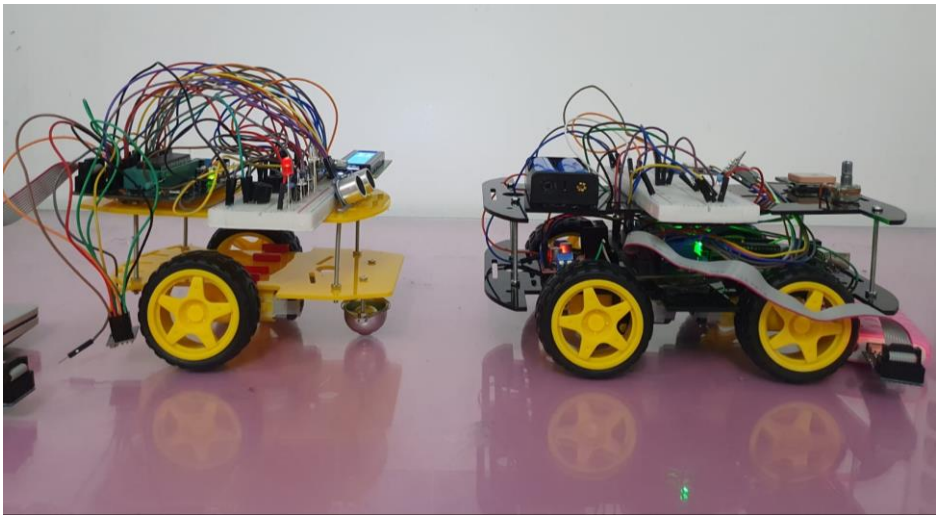


*Figure 88- Ultrasonic Results- Buzzer On*

# Chapter 9: Conclusion and Future Work

## 9.1 Conclusion

In conclusion, our study successfully implemented a simulated BCI-based V2V drowsiness detection system using an Atmega32 microcontroller. The integration of the V2V system enabled vehicles to exchange information, providing a comprehensive 360-degree situational awareness. This was achieved through the utilization of a GPS module for distance measurement, as well as a photo interrupter module to accurately measure the speed in revolutions per minute (RPM) using an encoder wheel.

The BCI component played a crucial role in classifying drowsiness levels, employing an LSTM Bidirectional AI model. By leveraging this model, we achieved accurate and reliable drowsiness classification of 78.042% accuracy. Real-time transmission of all gathered data was facilitated by an RF module, ensuring timely warnings were sent to nearby vehicles, effectively alerting them to the presence of a drowsy driver.

With the successful implementation and promising results obtained from our study, this BCI-based V2V drowsiness detection system holds great potential for real-world deployment. Its ability to save lives and prevent accidents makes it a valuable tool in addressing the dangers associated with drowsy driving, and challenges faced by individuals with conditions such as diabetes or sleep apnea.

## 9.2 Future Work

Our system, which has demonstrated good efficiency and effectiveness, has the potential for further enhancements to improve its performance and expand its capabilities. Some avenues for potential improvement and advancement include:

1) Using a BCI with three channels: The switch from a single-channel brain-computer interface (BCI) to a three-channel BCI is one option for increasing the system's accuracy. A larger dataset can be obtained by employing multiple channels, such as EEG electrodes positioned in various parts of the scalp. Because it provides more information about the user's brain activity, this increased data availability may increase the system's accuracy. It might be possible to extract features that are more robust and precise by making use of this additional data, which would result in enhanced performance. In addition, the majority of BCI datasets are gathered from headsets with three channels. Our model's ability to classify drowsiness could greatly improve if we were able to train it with this much data. However, due to the fact that our Neurosky Mindwave headset is a one-channel BCI, we were unable to implement this.

2) Integration of Multiple Drowsiness Detection Systems: To further enhance the accuracy of the drowsiness detection system, an integration with multiple drowsiness detection systems can be explored. By combining various detection approaches, such as face and eye recognition systems and heart rate monitoring, the system can take advantage of the strengths of each individual component. For example, eye and face recognition systems can capture detailed information about eye movement patterns and closure, the amount of frequent yawning of the individual, and the position of the head.

The time of day and the conditions under which a driver is driving will also be taken into account in this data. Furthermore, the heart rate monitoring can provide insights into physiological changes associated with drowsiness. A description of the face recognition system is shown in Fig.89.

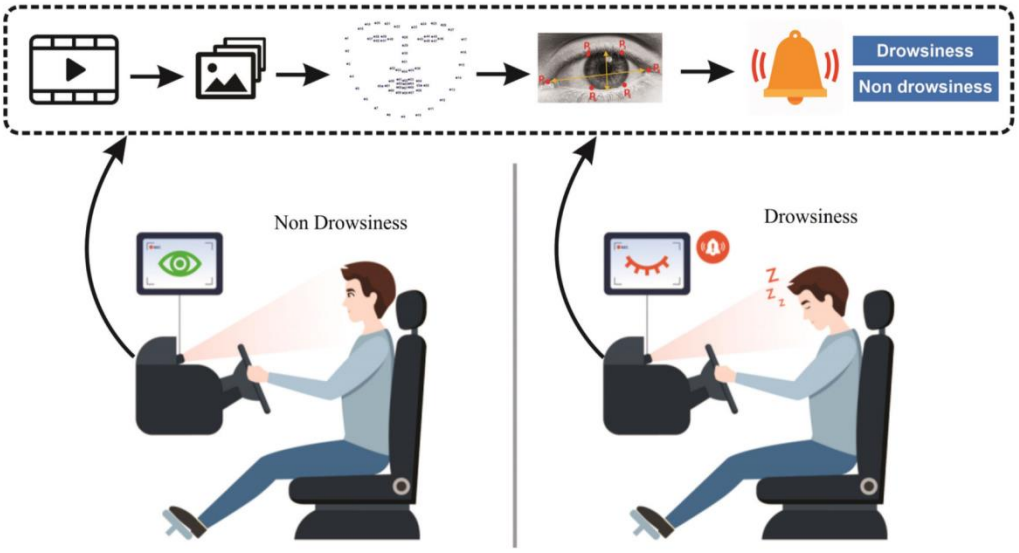



*Figure 89- Eye and Face Drowsiness Detection System*

3) Long-term Monitoring and Performance Evaluation: Long-term monitoring studies and evaluations of the system's performance over extended time frames can provide useful insights into its robustness and reliability. The system's generalizability can be enhanced by collecting data from a variety of populations and in a variety of environments.

4) Self-Parking Ability: Implementing an autonomous self-parking feature for the car when drowsiness is detected would enhance the overall safety and convenience of the system. Implementing the self-parking feature would require integrating the drowsiness detection system with the car's autonomous driving system. The drowsiness detection system would communicate with the

car's control system to initiate the self-parking sequence when drowsiness is detected. This could involve automatically finding an available parking space, maneuvering the car into the space, and safely stopping the car.

5) RTOS: utilizing a Real-Time Operating System to improve the real-time performance of our V2V communication system.

6) Different Frequency Band: The RF module works on the 2.4GHz band, which can cause interference with Wi-Fi or Bluetooth devices. In order to eliminate this interference, we can use a different frequency band dedicated for V2V communication systems.

# References


[1] Centers for Disease Control and Prevention. (n.d.). Drowsy driving. Retrieved from https://www.cdc.gov/sleep/features/drowsy-driving.html

[2] Plumstead, K. (2013). Using a Brain-Computer Interface to Control a Device (Final Year Project – EMP4110). Nelson Mandela Metropolitan University. Retrieved from https://www.plumpot.co.uk/wp-content/uploads/2019/09/Final-Report-1st-Draft.pdf

[3] NeuroSky. (n.d.). Arduino Tutorial. Retrieved from https://developer.neurosky.com/docs/doku.php?id=arduino_tutorial

[4] Nordic Semiconductor. (2008, March). NRF24L01+ Single Chip 2.4GHz Transceiver Product Specification. Retrieved from https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf

[5] Atmel Corporation. (2003, December). ATmega32 (L) Datasheet. Retrieved from https://pdf1.alldatasheet.com/datasheet-pdf/view/77378/ATMEL/ATMEGA32/+W5_J9UNRlElbDVRILHCC+/datasheet.pdf

[6] National Highway Traffic Safety Administration. (n.d.). Vehicle-to-Vehicle Communication. Retrieved from https://www.nhtsa.gov/technology-innovation/vehicle-vehicle-communication

[7] Aptiv. (2020, September 17). What is V2V? Retrieved from https://www.aptiv.com/en/insights/article/what-is-v2v

[8] NeuroSky. (n.d.). What does the ear-clip monitor measure? Retrieved from http://support.neurosky.com/kb/science/what-does-the-ear-clip-monitor

[9] ELEC Freaks. (n.d.). Ultrasonic Ranging Module HC - SR04. Retrieved from https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

[10] Chhabra, S. K. (November 6, 2021). Artificial Intelligence can make Brain computer interface more effective. Retrieved from LinkedIn: https://www.linkedin.com/pulse/artificial-intelligence-can-make-brain-computer-more-chhabra/?trk=public_profile_article_view

[11] Shih, J. J., Krusienski, D. J., & Wolpaw, J. R. (2012). Brain-Computer Interfaces in Medicine. Mayo Clinic Proceedings. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3497935/

[12] Brownlee, J. (2018, November 14). How to Develop LSTM Models for Time Series Forecasting. Retrieved from https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

[13] Photo Interrupter LM393 Datasheet. (n.d.). Retrieved from https://5.imimg.com/data5/VQ/DC/MY-1833510/lm393-motor-speed-measuring-sensor-module-for-arduino.pdf

[14] SiRF Technology, Inc. (2007). NMEA Reference Manual (Revision 2.1). Retrieved from: https://www.sparkfun.com/datasheets/GPS/NMEA%20Reference%20Manual-Rev2.1-Dec07.pdf

[15] Top Causes of Car Accidents: 25 Most Common Causes of Accidents on the Road. (2022, June 1). Retrieved from https://seriousaccidents.com/legal-advice/top-causes-of-car-accidents

[16] masasomeha. (2017, January 17). thinkgear_communications_protocol.txt. Retrieved from NeuroSky Developer website: https://developer.neurosky.com/docs/doku.php?id=thinkgear_communications_protocol

[17] javatpoint. (n.d.). Atmega32 AVR Microcontroller. Retrieved from https://www.javatpoint.com/atmega32-avr-microcontroller

[18] NeuroTechEDU. (n.d.). Introduction to BCI. Retrieved from http://learn.neurotechedu.com/introtobci/

[19] Components101. (2021, April 13). L293N Motor Driver Module. Retrieved from https://components101.com/modules/l293n-motor-driver-module

[20] javatpoint. (n.d.). SPI Protocol. Retrieved from https://www.javatpoint.com/spi-protocol

[21] Peña, E., & Legaspi, M. G. (2020, December). UART: A Hardware Communication Protocol. Analog Dialogue, 54(4). Retrieved from https://www.analog.com/media/en/analog-dialogue/volume-54/number-4/uart-a-hardware-communication-protocol.pdf