
**Alexandria University**

**Faculty of Engineering**

# Vehicle To Everything

**A Graduation Project Report Submitted To The Department Of**

**Communications and Electronics Engineering**

**Faculty Of Engineering – Alexandria University For The Partial Fulfilment Of The Requirements Of The B.Sc. Degree**

**By**

Abanoub Ibrahim Yanni

Ahmed Essam El-Din Saadawy

Asmaa Gamal Abdallah

Mark George Louiz

Marina Fouad Aziz

Merna Mansour Kamal

Semon Joseph Shehata

Tasneem Adel Khamis

Under supervision of

**Dr. Mohammed El-Shimy**

July 2023

# Acknowledgement

We would like to express our sincerest gratitude to our supervisor, Dr. Mohamed Elshimy, for his guidance and support throughout our graduation project. We are grateful for his patience, knowledge, and encouragement.

We would like to thank our parents and family for their love and support. We are also grateful to our friends for their help and encouragement.

Lastly, we would like to thank all the people who have contributed to this project in any way. Thank you all for your help and support.

# Table of content

# List Of Figures

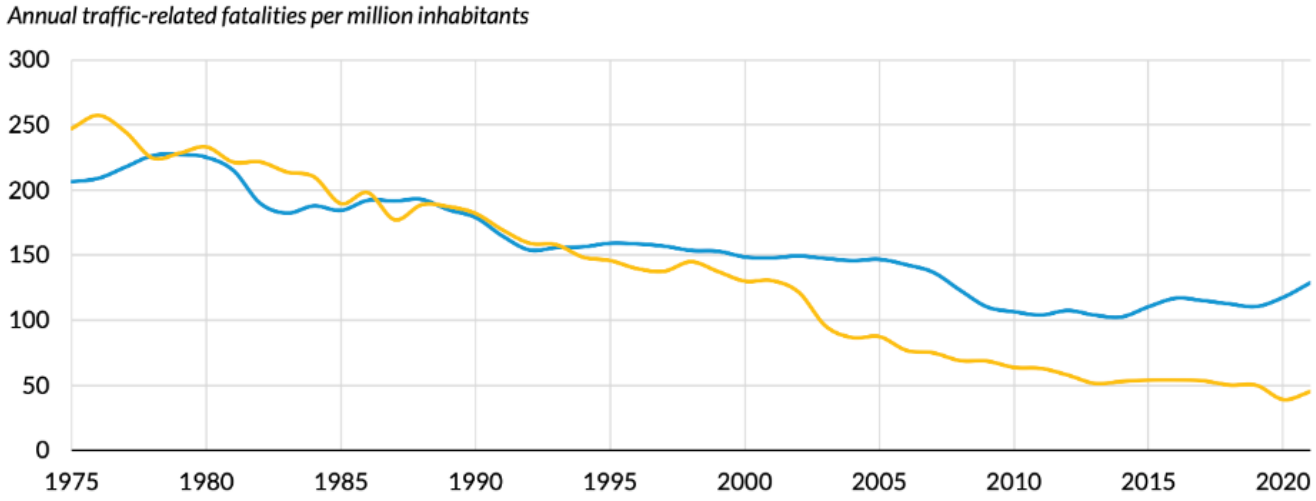
# CHAPTER 1
# Introduction

## 1.1) Introduction and motivation

People's lives are valuable. They often spend a great deal of money on stuff like life insurance as they profoundly believe this fact. Everyone wishes to avoid the loss of a dear person due to tragic events such as car accidents. Now annual highway fatalities and injuries keep increasing from year to year.

According to the World Health Organization (WHO), approximately 1.35 million people die each year as a result of road traffic accidents, and an additional 20-50 million people suffer non-fatal injuries. Low- and middle-income countries account for more than 90% of global road traffic deaths, despite having only 60% of the world's vehicles. This is due to a combination of factors, including poor road infrastructure, weak law enforcement, and inadequate safety measures. The economic cost of traffic accidents is significant. According to the International Transport Forum, the cost of traffic accidents in OECD countries is equivalent to 2% of GDP. Distracted driving is a major contributor to traffic accidents. According to the National Highway Traffic Safety Administration (NHTSA), distracted driving was a factor in 8.5% of all fatal crashes in the United States in 2019. Speeding is another major contributor to traffic accidents. According to WHO, a 5% increase in average speed leads to a 10% increase in the number of fatal crashes.

After the development of the driver systems (ADAS) and the implementation of the V2V system ,The U.S. NHTSA estimates a minimum of 13% reduction in traffic accidents if a V2V system were implemented, resulting in 439,000 fewer crashes per year.



With more advance of the driver assistance systems, imagine if we could transfer data about vehicles states and information without violating the driver privacy between vehicles and everything around it specially the infrastructure and the nearby vehicles. How many more lives we could save?

## 1.2) System overview:

Moreover, most countries are vulnerable to traffic problems and complications as consequences of road accidents. Vehicle-to-everything (V2X) is communication between a vehicle and any entity that may affect, or may be affected by, the vehicle. It is a vehicular communication system that incorporates other more specific types of communication as V2I (vehicle-to-infrastructure), V2N (vehicle-to-network), V2V (vehicle-to-vehicle), V2P (vehicle-to-pedestrian), V2D (vehicle-to-device) and V2G (vehicle-to-grid).The main motivations for V2X are road safety, traffic efficiency, and energy savings.

One of the ADAS features added is the accident detection , when accident is detected the probability of saving driver's life is higher by 20% in the fetal accidents according to NHS , in addition when the accident detected the location of the car is sent to the nearest hospital, an application of V2I communication between vehicles nodes and surrounding infrastructure.

The second ADAS feature is crash avoidance which is important ADAS feature became more accurate thanks to the V2V communication to alert the rear vehicle when accident happens. This feature ensures that our system can adapt and deal with the road circumstances safely.

Now for one of the most critical use cases, the Driver Monitoring System. According to the National Highway Traffic Safety Administration, distracted driving caused over 3,000 Pedestrians' deaths in 2019, and 4119 drivers deaths in 2019.

A recent study reported that advanced emergency braking systems alone could reduce crashes in light vehicles by 33%. And that's just a glimpse of what DMS can do. All around the globe, carmakers are also expected to rapidly increase the use of Driver Monitoring Systems or DMS -in short- in response to growing demand for driver assist systems and government regulations. Driver monitoring systems use strategically placed camera to ensure that the driver is paying attention to the road, awake and alert.

Another important unit in our project is the smart garage which is a clear application on V2I communication. The smart garage system have the driver's ability to reserve free slot in the garage using mobile application. Another feature is the insurance of the reserved vehicle using the license plate.

To implement the V2V communication , V2I communication systems in addition to these ADAS features we used RPI with edge server to run all the deep learning models in addition to V2X server to data transfer in long range communication and nRF module in low level programming to achieve reliable and fast data transfer in short range communication between vehicles. We used stm32f103 blue pill as main MCU in our system with different sensors such as IMU to send the feedback to the another MCU to control the cars in our different scenarios in the normal Operation the vehicle is controlled used Bluetooth module.

# CHAPTER 2
# Hardware Specifications

## 2.1) Microcontrollers

### 2.1.1) Blue Bill stm32f103c6t6


**Figure 1 Blue Bill stm32f103c6t6**

**Introduction**

The STM32 "Blue Pill" is a widely used development board based on the STM32F103C8T6 microcontroller from STMicroelectronics. It features an ARM Cortex-M3 core running at a clock speed of up to 72 MHz. The board offers a range of GPIO pins for digital and analog I/O, as well as various communication interfaces including USB, USART, SPI, and I2C.

The Blue Pill board supports programming and debugging through a USB interface and is compatible with the STM32Cube software development platform. It can also be programmed using the Arduino IDE, making it compatible with a wide range of Arduino libraries and shields. The board includes features such as an onboard microSD card slot for memory expansion and a built-in voltage regulator for flexible power supply options.

One of the key advantages of the STM32 Blue Pill is its affordability, making it accessible to hobbyists, students, and developers on a budget. It has gained a strong following within the maker and electronics enthusiast community, resulting in extensive online resources, tutorials, and community-driven projects.

**Interfacing**

In this project, the STM32 "Blue Pill" development board was utilized to implement sophisticated control and monitoring features for a vehicle. The Blue Pill, along with various external components, played a crucial role in enabling the following functionalities:

- An external Bluetooth module enabled wireless control of the vehicle through an RC Bluetooth controller mobile application.
- The Blue Pill processed control commands received from the mobile application, allowing precise motor control and directing the vehicle's movement.
- An IMU sensor connected to the Blue Pill detected sudden changes in acceleration, helping to identify potential accidents.
- An ultrasonic sensor was utilized to detect obstacles in front of the vehicle, enhancing safety measures.
- The Blue Pill communicated with a Raspberry Pi using UART protocol, enabling accident information to be sent to a server for remote driver access.
- An NRF sensor facilitated short-range communication between neighboring vehicles, allowing for the exchange of warnings and alerts to avoid multi-stage accidents.
- Overall, the integration of the Blue Pill, Bluetooth module, IMU sensor, ultrasonic sensor, Raspberry Pi, and NRF sensor provided a comprehensive solution for vehicle control, accident detection, and inter-vehicle communication, enhancing safety and promoting cooperative driving strategies.

In addition to the previous functionalities enabled by the “Blue Pill”, The ability to reserve a parking slot in the garage through a user-friendly interface has been proposed for the system. This feature aims to streamline the parking process and enhance the overall user experience. The integration of the Blue Pill with the Raspberry Pi and cloud server allows for seamless communication and reservation management. Here's how the proposed feature would work:

**User Interaction:**

When a user wishes to reserve a parking slot, they simply press a dedicated push button connected to the STM32 Blue Pill. This action triggers a sequence of events to initiate the reservation process.

**Signal Transmission:**

Upon detecting the button press, the STM32 Blue Pill immediately sends a signal to the Raspberry Pi, signaling the user's intention to reserve a parking slot. This communication is established using the existing UART protocol between the two devices.

**Reservation Request Handling:**

The Raspberry Pi, acting as the central control unit, receives the reservation request from the STM32 Blue Pill. It processes the request, prepares the necessary data, and establishes a secure connection with the cloud server.

**Cloud Server Integration:**

Using the established connection, the Raspberry Pi sends the reservation request and relevant information to the cloud server. The server, equipped with the necessary algorithms, updates the parking slot availability status and reserves the slot for the user.

**User Feedback:**

Once the reservation is successfully made on the cloud server, the Raspberry Pi provides feedback to the STM32 Blue Pill, indicating the status of the reservation. This feedback can be visualized through LED indicators or an LCD display, providing the user with real-time confirmation of the reserved parking slot.

**Cloud Server Updates:**

Simultaneously, the cloud server updates the parking slot availability status, making it visible to other users accessing the cloud-based application. This ensures that all users are aware of the reserved parking slot and can plan their parking accordingly.

### 2.1.2) AVR Atmega 32

**Figure 2 Atmega 32**

#### Introduction

The AVR ATmega32 microcontroller is an 8-bit microcontroller based on the AVR architecture. It features a high-performance, low-power 8-bit CPU with a clock frequency of up to 16 MHz. The microcontroller has 32 KB of flash program memory, 2 KB of SRAM, and 1 KB of EEPROM. It offers 32 I/O pins for connecting external devices and peripherals.

The ATmega32 includes a variety of integrated peripherals such as timers/counters, USART, SPI, I2C, ADC, and PWM, which enhance its capabilities for different applications. It also has a flexible interrupt system that supports both external and internal interrupts.

Microchip provides a range of development tools, including Atmel Studio IDE, compilers, debuggers, and programmers, to facilitate the development process. The ATmega32 microcontroller finds applications in : industrial automation, robotics, consumer electronics, automotive systems, and more.

#### Interfacing

In this project AVR ATmega32 is used in:

Garage unit as the integration of AVR ATmega32 microcontroller, ultrasonic sensors, 2:1 multiplexers, Raspberry Pi, and server communication to create a robust smart parking system. By utilizing AVR ATmega32, ultrasonic sensors, and multiplexers, we can accurately detect obstacles and monitor the occupancy status of each parking slot. The data collected will be transmitted to the Raspberry Pi, which will facilitate communication with a centralized server, making the parking slot availability information accessible to users remotely.

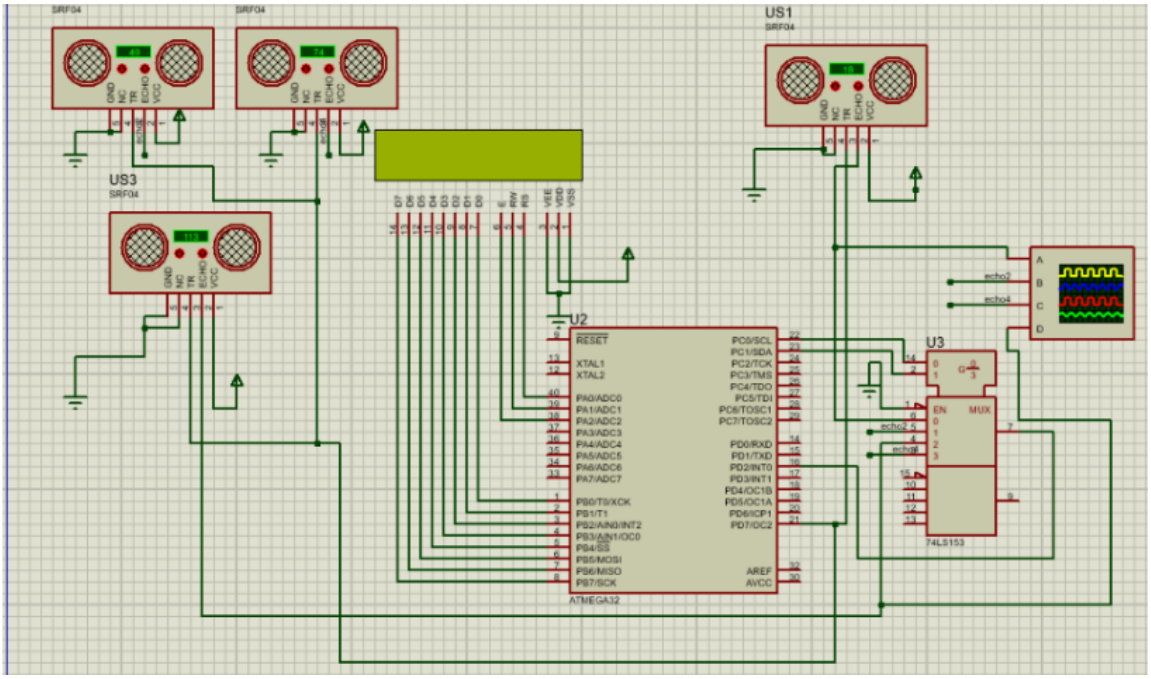



**Figure 3 Interfacing AVR ATmega32 with 4:1 Mux connected to 4 ultrasonics sensors and LCD**

### 2.1.3) ESP32

- **Introduction:**
  In our V2X (Vehicle-to-Everything) project, we aim to develop a reliable and efficient vehicle control system using the ESP32 microcontroller and the Cytron MDD10A motor driver. This section focuses on the implementation details of controlling the second vehicle in our project.

- **Hardware Setup:**
  To control the vehicle's motion, we utilize the ESP32 microcontroller board as the main controller. The Cytron MDD10A motor driver is connected to the ESP32, enabling us to regulate the speed and direction of the vehicle's movement.

- **Speed Control using PWM:**
  The Pulse Width Modulation (PWM) feature of the ESP32 is employed to achieve variable speed control for our vehicle. By adjusting the duty cycle of the PWM signal, we can control the power supplied to the motors, thus altering their rotational speed.

- **Sleep Mode Implementation:**
  To conserve energy when not in use, we have implemented a sleep mode functionality for our vehicle. When activated, this mode gradually reduces the speed of the vehicle until it reaches zero velocity. This ensures that no unnecessary power is consumed during idle periods.

- **Motion Control using Digital Write Function:**

  For different cases of motion such as forward, backward, left turn, and right turn, we utilize the digital write function provided by ESP32 to control specific pins on the Cytron MDD10A motor driver.

  - To move forward: We set both PWM and direction pins to provide high signals.
  - To move backward: We set both PWM and direction pins to provide low signals.
  - For left turn: We set one motor's direction pin high while keeping another motor's direction pin low.
  - For right turn: We set one motor's direction pin low while keeping another motor's direction pin high.

## 2.2) Sensors interfacing

### 2.2.1)Ultrasonic

Ultrasound, or ultrasonography, works on the principle that sound is reflected at different speeds by tissues or substances of different densities.

As the name indicates, ultrasonic sensors measure distance by using ultrasonic waves.

An ultrasonic sensor is an electronic device that measures the distance of a target object by emitting ultrasonic sound waves, and converts the reflected sound into an electrical signal. Ultrasonic waves travel faster than the speed of audible sound Ultrasonic Sensors measure the distance to the target by measuring the time between the emission and reception

- **Distance calculation :**

The distance can be calculated with the following formula:

Distance L = 1/2 × T × C

where L is the distance, T is the time between the emission and reception, and C is the sonic speed. (The value is multiplied by 1/2 because T is the time for go-and-return distance.)

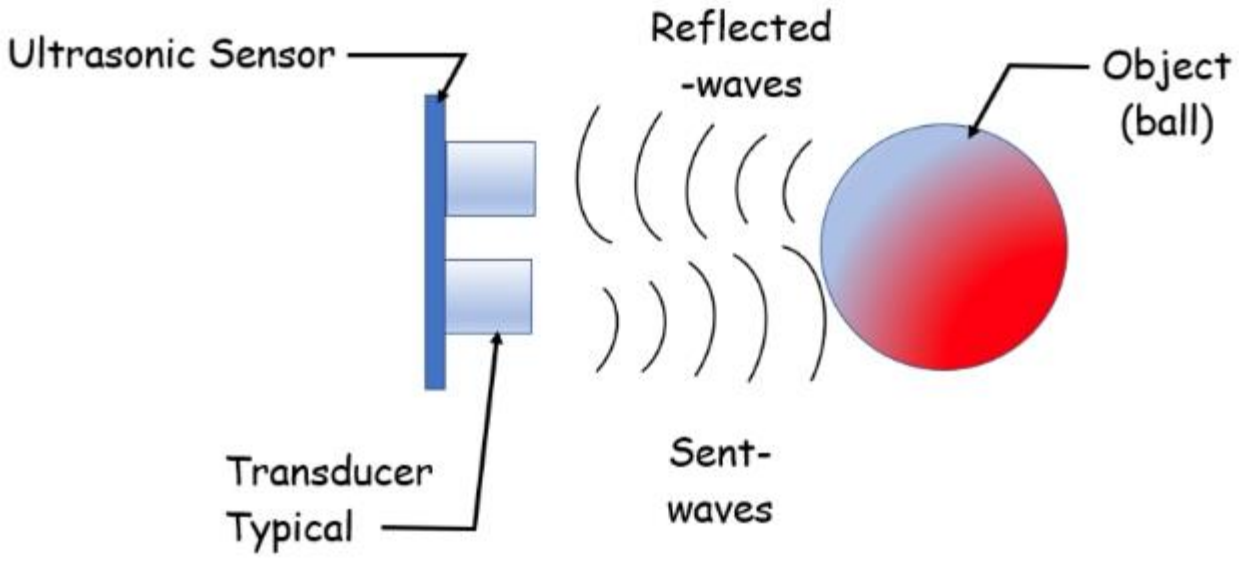



**Figure 4 Distance calculation of ultrasound**

- **Using Ultrasonic Sensor in our Project:**

The sensor has 4 pins.

• **VCC** supplies power to the HC-SR04 ultrasonic sensor.

• **Trig (Trigger) pin** is used to trigger ultrasonic sound pulses. By setting this pin to HIGH for 10µs, the sensor initiates an ultrasonic burst.

• **Echo pin** goes high when the ultrasonic burst is transmitted and remains high until the sensor

receives an echo, after which it goes low. By measuring the time the Echo pin stays high, the distance can be calculated.

• **GND** is the ground pin.

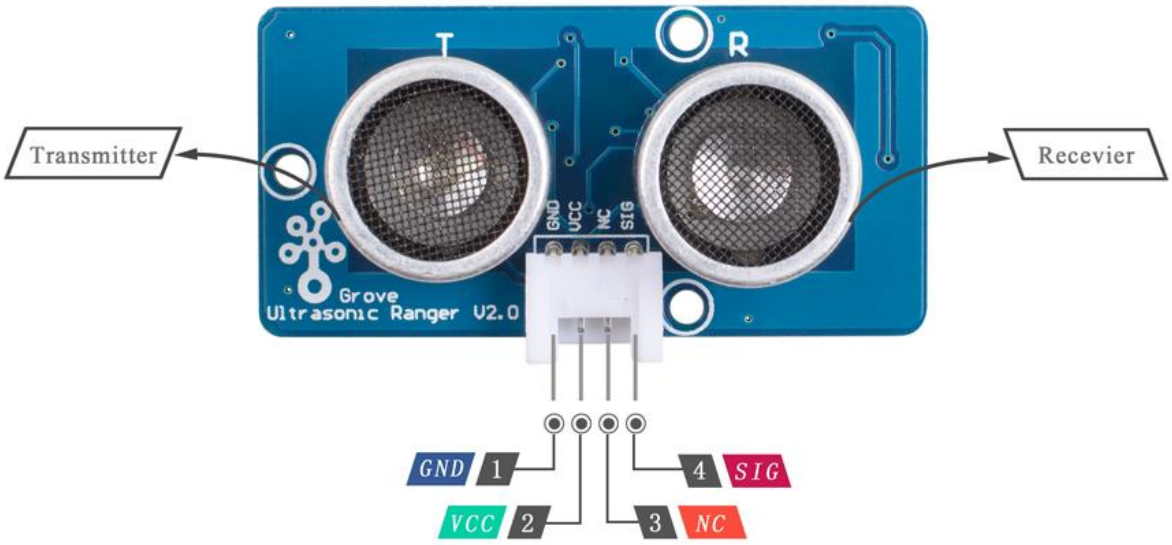



**Figure 5 Ultrasonic sensor pinout**

**1- Used in the garage unit with the AVR microcontroller**

- The power pins are connected to the +5V and ground terminals of the AVR so that it can have sufficient power to operate.
- The trigger pin is connected to pin , while the echo pin is connected to pin .
- There are four Ultrasonic in the garage , Each sensor is used to detect if there is a car in front of it or not.
- If the distance measured by sensor was greater than the threshold , there is an empty place in the garage .

**2- Used in Vehicle unit with STM microcontroller**

- In the accident scenario if there is something in front of the vehicle and the IMU sensed a great difference in the acceleration , it means that there is an accident
- The combined inputs from the IMU sensor and ultrasonic sensor allowed the Blue Pill to identify potential accident scenarios.

### 2.2.2) IMU (Inertial Measurement Unit):

An Inertial Measurement Unit (IMU) is an electronic device that measures and reports an object's specific force , angular rate, and sometimes the orientation of the body, using a combination of accelerometers, gyroscopes, and sometimes magnetometers. The IMU detects linear acceleration using one or more accelerometers and rotational rate using one or more gyroscopes. The data reported by the IMU can be fed into a processor which calculates altitude, velocity, and position in a navigation system.

### MPU-6050:

The MPU6050 sensor is a 6-axis motion tracking device that combines a 3-axis gyroscope and a 3-axis accelerometer in a single package. It is commonly used in robotics, drones, gaming controllers, and other motion sensing applications. The sensor can measure acceleration, rotation rate, and temperature, and provides data in digital format through an I2C interface. The MPU6050 sensor is known for its accuracy, low power consumption, and compact size.

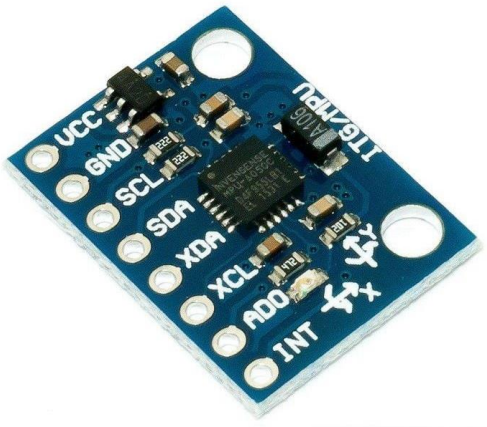
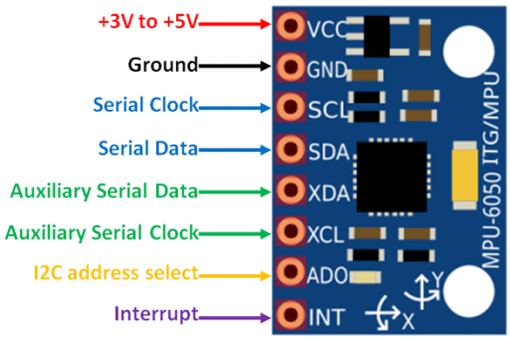



*Figure 6 MPU*

| Pin No. | Pin Name | Description |
|---|---|---|
| 1 | **Vcc** | Provides power for the module, can be +3V to +5V. |
| 2 | **GND** | Connected to Ground of system |
| 3 | **Serial Clock (SCL)** | Used for providing clock pulse for I2C Communication |
| 4 | **Serial Data (SDA)** | Used for transferring Data through I2C communication |
| 5 | **Auxiliary Serial Data (XDA)** | Can be used to interface other I2C modules with MPU6050. It is optional |
| 6 | **Auxiliary Serial Clock (XCL)** | Can be used to interface other I2C modules with MPU6050. It is optional |
| 7 | **AD0** | If more than one MPU6050 is used a single MCU, then this pin can be used to vary the address |
| 8 | **Interrupt (INT)** | Interrupt pin to indicate that data is available for MCU to read. |




**The values given by the IMU:**

- GyroZ: it is the gyroscopic or angular velocity around Z- axis.
- GyroY: it is the gyroscopic or angular velocity around Y- axis.
- GyroX: it is the gyroscopic or angular velocity around X- axis.
- AccZ: it is the gravitational acceleration around Z-axis.

- AccY: it is the gravitational acceleration around Y-axis.
- AccX: it is the gravitational acceleration around X-axis.

Although this module has both accelerometer and gyroscope, but the focus

would be on the acceleration and not the angular speed.

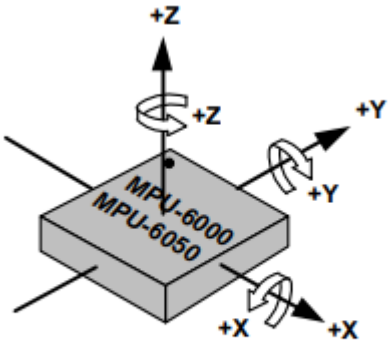



*Figure 7 values given by IMU*

**Configure the I2C address:**

The default I2C address of the MPU6050 sensor is 0x68 (0b1101000) when the AD0 pin is connected to GND. When the AD0 pin is connected to Vcc, the address becomes 0x69 (0b1101001) . However, the address can be changed by writing to the AD0 pin or to a specific register on the sensor. Changing the address can be useful to avoid conflicts with other devices on the I2C bus or to use multiple MPU6050 sensors on the same bus. It is usually done by physically changing the AD0 connection or by writing to the corresponding register of the sensor using an I2C bus controller.

**Configure the accelerometer scale:**

To configure the acceleration scale in the MPU6050 sensor, you can write to the ACCEL_CONFIG register of the sensor. This register is located at address 0x1C in the sensor's memory map, and contains bits that control the full-scale range of the accelerometer. The bits that control the range are located at bits 3-4 of this register and are labeled as AFS_SEL.

| AFS_SEL | Full Scale Range |
|---|---|
| 0 | ± 2*g* |
| 1 | ± 4*g* |
| 2 | ± 8*g* |
| 3 | ± 16*g* |

**Figure 8 Configure the accelerometer scale**

## Accelerometer Measurements:

| Register (Hex) | Register (Decimal) | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|---|---|
| 3B | 59 | ACCEL_XOUT[15:8] | | | | | | | |
| 3C | 60 | ACCEL_XOUT[7:0] | | | | | | | |
| 3D | 61 | ACCEL_YOUT[15:8] | | | | | | | |
| 3E | 62 | ACCEL_YOUT[7:0] | | | | | | | |
| 3F | 63 | ACCEL_ZOUT[15:8] | | | | | | | |
| 40 | 64 | ACCEL_ZOUT[7:0] | | | | | | | |

**Figure 9 Accelerometer Measurements**

These registers store the most recent accelerometer measurements. Accelerometer measurements are written to these registers at the Sample Rate as defined in SMPRT_DIV (Register 25).

Each 16-bit accelerometer measurement has a full scale defined in ACCEL_FS (Register 28). For each full scale setting, the accelerometers' sensitivity per LSB in ACCEL_xOUT is shown in the table below.

| AFS_SEL | Full Scale Range | LSB Sensitivity |
|---|---|---|
| 0 | ±2*g* | 16384 LSB/*g* |
| 1 | ±4*g* | 8192 LSB/*g* |
| 2 | ±8*g* | 4096 LSB/*g* |
| 3 | ±16*g* | 2048 LSB/*g* |

**Figure 10 accelerometers' sensitivity**

To get the exact acceleration we use this equation:

accelX = ACCEL_XOUT * 9.8 / LSB Sensitivity

accelX is the acceleration in X direction.

ACCEL_XOUT is the value read from the registers ACCEL_XOUT_H & ACCEL_XOUT_L.

LSB Sensitivity : is the sensitivity used according to the value configured in AFS_SEL.

Same thing for Y and Z directions.

Functions of MPU6050 Driver:

void MPU6050_vInit(void)

This function uses the I2C peripheral used to communicate with the IMU as a slave.

The IMU is powered on and configured in this function:

- PWR_MGMT_1 = 0x00 for Internal 8MHz oscillator.
- CONFIG = 0x01 for LPF with bandwidth = 184(accel) and 188(gyro).
- ACCEL_CONFIG = 0x10 , AFS_SEL = 2 for full scale range = ± 8g as mentioned previously.

void MPU6050_vReadAccel_ALL(sint16 *Copy_u16Buffer)

This function’s input is a pointer to the a 16-bit containing the variables to store the accelerations.

Puts the 8-bit ACCEL_XOUT_H register and 8-bit ACCEL_XOUT_L register in one variable holding the acceleration in X direction , same thing for Y and Z directions.

void MPU6050_vReadGyro_ALL(sint16 *Copy_u16Buffer)

This function’s input is a pointer to the a 16-bit containing the variables to store the accelerations.

Puts the 8-bit GYRO_XOUT_H register and 8-bit GYRO_XOUT_L register in one variable holding the gyroscope measurement in X direction , same thing for Y and Z directions.

uint8 MPU6050_vDetectChange(double *Copy_dOldValues)

This functions takes the old values of acceleration and compares it with the new measured value if there is no change then the function will return 0 , if there is noticeable change (using some threshold value) then the function returns 1 indicating that there might be an accident.

### 2.2.3) nRF24L01

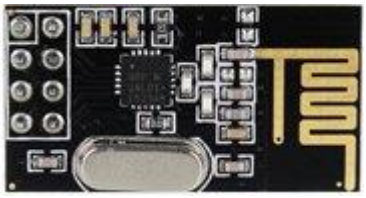


**Figure 11 nRF**

### Hardware Overview

### Radio Frequency

The nRF24L01+ module is designed to operate in the 2.4 GHz worldwide ISM frequency band and uses GFSK modulation for data transmission. The data transfer rate is configurable and can be set to 250kbps, 1Mbps, or 2Mbps.

### Power

The module's operating voltage ranges from 1.9 to 3.9V. Please keep in mind that powering the module with 5V will most likely damage your nRF24L01+ module.

Despite the fact that the module operates at 1.9V to 3.6V, the logic pins are 5-volt tolerant, so you do not need a logic level translator.

The output power of the module can be programmed to be 0 dBm, -6 dBm, -12 dBm, or -18 dBm. At 0 dBm, the module consumes only 12 mA during transmission, which is less than the consumption of a single LED.

And the best part is that it consumes only 26 µA in standby mode and 900 nA in power down mode. That's why it's the go-to wireless device for low-power applications.

## SPI Interface

The nRF24L01+ communicates over a 4-pin SPI (Serial Peripheral Interface) with a maximum data rate of 10Mbps.

All parameters, including frequency channel (125 selectable channels), output power (0 dBm, -6 dBm, -12 dBm or -18 dBm), and data rate (250kbps, 1Mbps, or 2Mbps), can be configured through the SPI interface.

The SPI bus uses the concept of a master and a slave. In most of our projects, the Arduino serves as the master and the nRF24L01+ module serves as the slave.

## Technical Specifications

Here are the specifications:

| | |
|---|---|
| Frequency Range | 2.4 GHz ISM Band |
| Maximum Air Data Rate | 2 Mb/s |
| Modulation Format | GFSK |
| Max. Output Power | 0 dBm |
| Operating Supply Voltage | 1.9 V to 3.6 V |
| Max. Operating Current | 13.5mA |
| Min. Current(Standby Mode) | 26µA |
| Logic Inputs | 5V Tolerant |
| Communication Range | 800+ m (line of sight) |

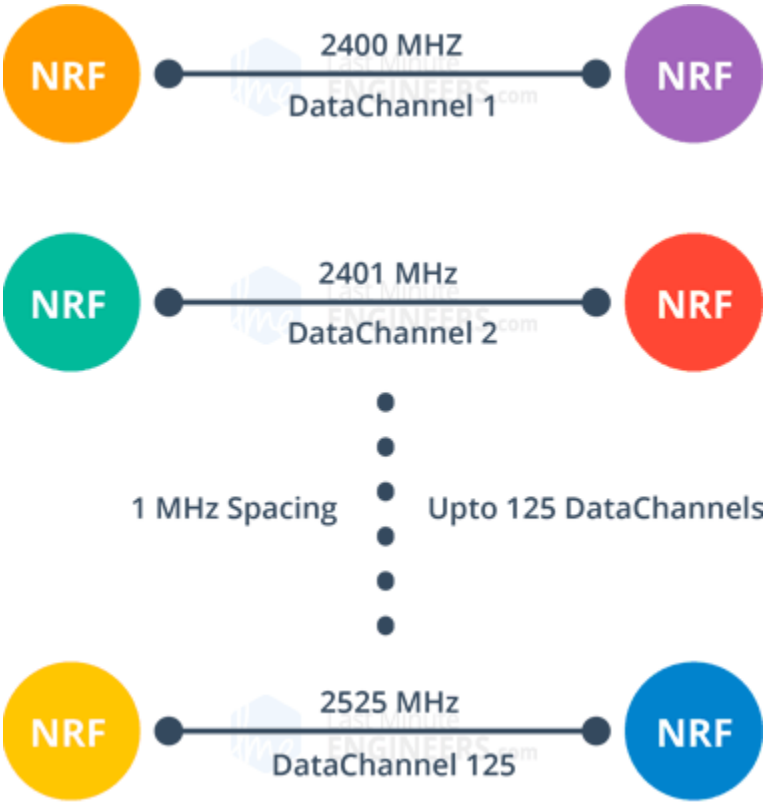
**How does the nRF24L01+ module work?**

**RF Channel Frequency**

The nRF24L01+ module transmits and receives data on a specific frequency known as a channel. For two or more modules to communicate with each other, they must be on the same channel. This channel can have any frequency in the 2.4 GHz ISM band, or more precisely, any frequency between 2.400 and 2.525 GHz (2400 to 2525 MHz).

Each channel takes up less than 1 MHz of bandwidth. This gives us 125 possible channels with a 1MHz spacing.

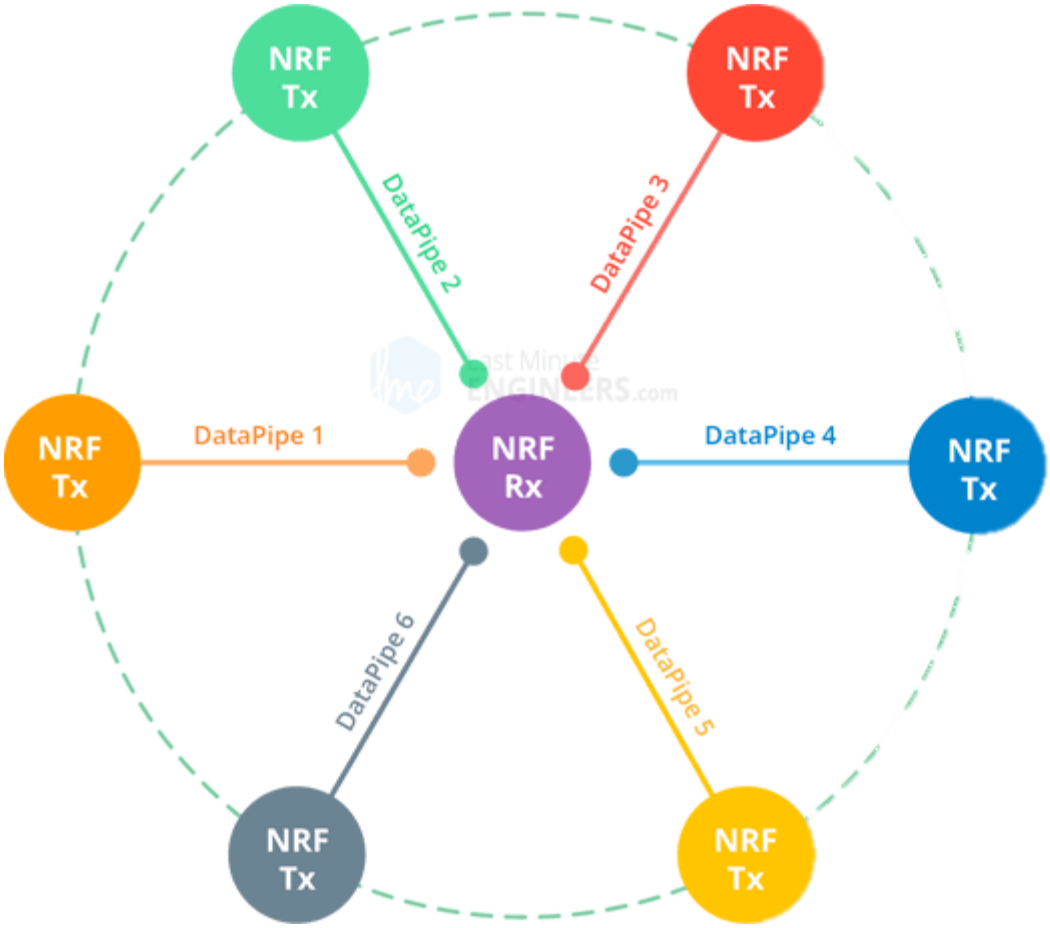
This means that the nRF24L01+ can operate on 125 different channels, allowing you to build a network of 125 independently operating modems in one location.




**nRF24L01+ Multiceiver Network**

The nRF24L01+ includes a feature known as Multiceiver. It stands for Multiple Transmitter Single Receiver.

In a multiceiver network, each RF channel is logically divided into six parallel data channels known as data pipes. In other words, the data pipe is one of six logical channels within a single physical RF channel. Each data pipe has its own unique address, known as a data pipe address. Only one data pipe can receive a packet at a time.



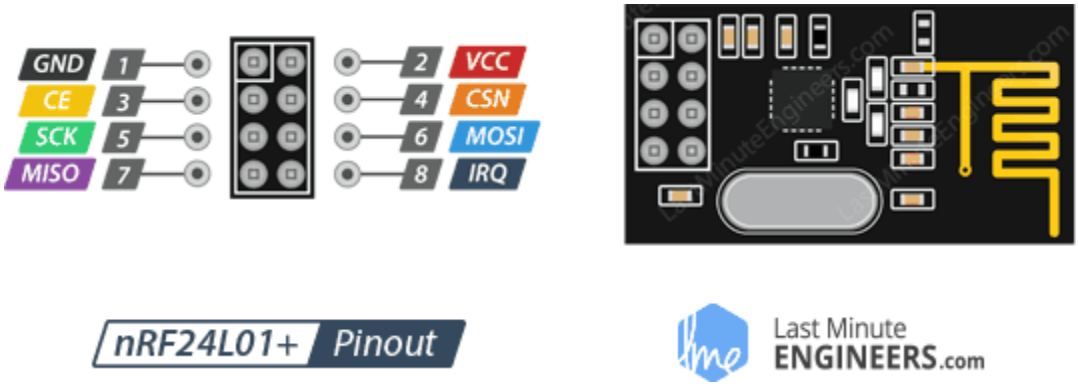



*Figure 12 nRF pinout*

**How stm32 deal with nrf24l01**

Stm32 use Spi protocol to send a data frame to nrf24 it called a command

For example we need to read status register

So we will make the following :

1-reset CSN pin to stablish the connection

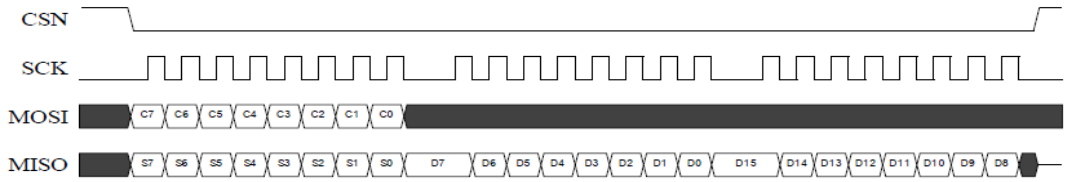
2-stm32 apply on MOSI pin the command of reading operation followed by the register address

| Command name | Command word (binary) | # Data bytes | Operation |
|---|---|---|---|
| R_REGISTER | 000A AAAA | 1 to 5 LSByte first | Read command and status registers. AAAAA = 5 bit Register Map Address |
| W_REGISTER | 001A AAAA | 1 to 5 LSByte first | Write command and status registers. AAAAA = 5 bit Register Map Address<br>Executable in power down or standby modes only. |
| R_RX_PAYLOAD | 0110 0001 | 1 to 32 LSByte first | Read RX-payload: 1 – 32 bytes. A read operation always starts at byte 0. Payload is deleted from FIFO after it is read. Used in RX mode. |
| W_TX_PAYLOAD | 1010 0000 | 1 to 32 LSByte first | Write TX-payload: 1 – 32 bytes. A write operation always starts at byte 0 used in TX payload. |

| Address (Hex) | Mnemonic | Bit | Reset Value | Type | Description |
|---|---|---|---|---|---|
| 07 | STATUS | | | | Status Register (In parallel to the SPI command word applied on the MOSI pin, the STATUS register is shifted serially out on the MISO pin) |
| | Reserved | 7 | 0 | R/W | Only '0' allowed |
| | RX_DR | 6 | 0 | R/W | Data Ready RX FIFO interrupt. Asserted when new data arrives RX FIFO[b].<br>Write 1 to clear bit. |
| | TX_DS | 5 | 0 | R/W | Data Sent TX FIFO interrupt. Asserted when packet transmitted on TX. If AUTO_ACK is activated, this bit is set high only when ACK is received.<br>Write 1 to clear bit. |
| | MAX_RT | 4 | 0 | R/W | Maximum number of TX retransmits interrupt<br>Write 1 to clear bit. If MAX_RT is asserted it must be cleared to enable further communication. |
| | RX_P_NO | 3:1 | 111 | R | Data pipe number for the payload available for reading from RX_FIFO<br>000-101: Data Pipe Number<br>110: Not Used<br>111: RX FIFO Empty |
| | TX_FULL | 0 | 0 | R | TX FIFO full flag.<br>1: TX FIFO full.<br>0: Available locations in TX FIFO. |
| | | | | | |

3-wait for nrf to get back the data from that register in the next cycles

| Abbreviation | Description |
|---|---|
| Cn | SPI command bit |
| Sn | STATUS register bit |
| Dn | Data Bit (**Note:** LSByte to MSByte, MSBit in each byte first) |




### 2.2.4)Bluetooth Module

The HC-05 is a popular module which can add two-way (full-duplex) wireless functionality to our projects. Bluetooth is a short-range wireless technology; it can be used to exchange data between two devices in a short distance, so Bluetooth communication is a suitable choice to communicate between a mobile and the project. We used this module to communicate between STM32 and a phone with

Bluetooth functionality .There are many android applications that are already available which makes this process a lot easier.

The module communicates with the help of USART at 9600 baud rate hence it is easy to interface with any microcontroller that supports USART.

It has 6 pins , button and Led :

- **The state** pin is connected to on board LED, it can be used as a feedback to check if Bluetooth is working properly.
- **TX pin** "Transmits Serial Data" Everything received via Bluetooth will be given out by this pin as serial data.
- **RX pin** "Receive Serial Data" Every serial data given to this pin will be broadcasted via Bluetooth
- **GND pin** of module, connect to system ground.
- **VCC pin ,** Powers the module. Connect to +5V Supply voltage
- **Enable pin ,** This pin is used to toggle between Data Mode (set low) and at command mode (set high). By default it is in Data mode
- **Led Indicates** the status of Module :
  - Blink once in 2 sec : Module has entered Command Mode
  - Repeated Blinking : Waiting for connection in Data Mode
  - Blink twice in 1 sec : Connection successful in Data Mode
- **Button** Used to control the Key/Enable pin to toggle between Data and command Mode

Simply we power the module with +5V and connect the Rx pin of the module to the Tx of MCU and Tx pin of module to Rx of MCU

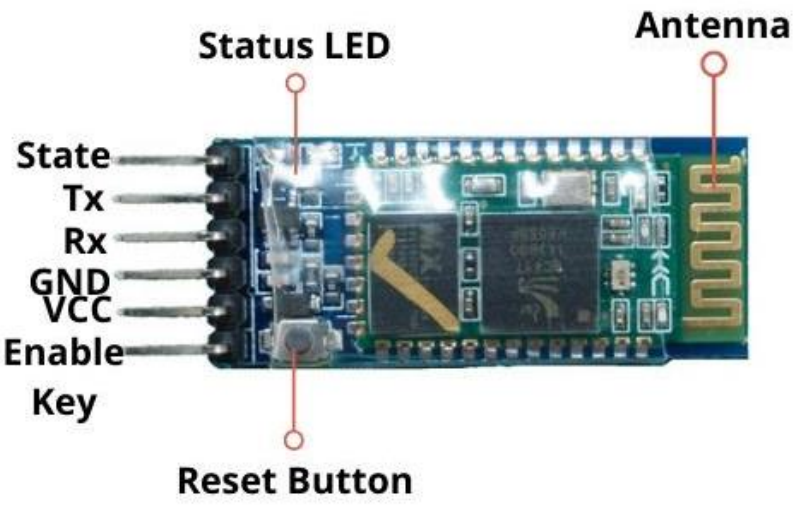



**Figure 13** **Bluetooth module pinout**

To control the car we use a mobile application called **"Bluetooth RC Controller"**

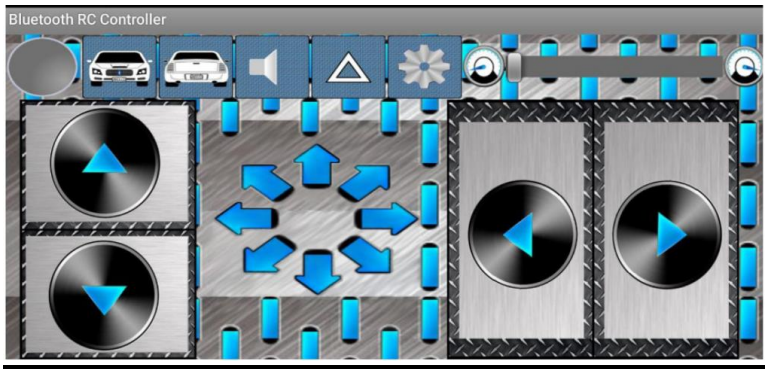



**Figure 14** **Bluetooth RC Controller**

which send character to the Bluetooth module and by sending and receiving characters we check and compare it to determine the next move

For example :

- **Receiving :**

- “F” to move Forward
- “B” to move Backward
- “L” to move Left
- “R” to move Right

### 2.2.5) GPS module

The Global Positioning System (GPS) is a satellite-based navigation system that allows precise determination of location, speed, and time information anywhere on Earth. GPS calculates distances by utilizing the principles of trilateration, a technique that involves measuring the time it takes for signals to travel between GPS satellites and the receiver.

#### 2.2.5.1) The working principle of GPS in calculating distance can be summarized as follows:

- **Satellite Signal Transmission:**

GPS satellites orbiting the Earth continuously emit signals that contain precise timing information and satellite identification codes. These signals travel at the speed of light.

- **Receiver Signal Reception:**

A GPS receiver, such as a navigation device or a smartphone, receives signals from multiple GPS satellites within range. To calculate distance, the receiver must capture signals from at least four satellites simultaneously. The receiver's internal clock precisely timestamps the moment each signal is received.

- **Time Delay Calculation:**

The GPS receiver measures the time delay between the transmission of each satellite signal and its reception. By comparing the time the signal was transmitted, as indicated by the satellite's signal, with the time it was received according to the receiver's clock, the receiver can determine the signal's travel time.

- **Pseudorange Calculation:**

Using the known speed of light, the receiver multiplies the signal's travel time by the speed of light to obtain the pseudorange. The pseudorange represents the approximate distance between the receiver and each satellite.

- **Satellite Geometry Analysis:**

The GPS receiver analyzes the geometry created by the signals from multiple satellites. This analysis involves determining the distances between the receiver and each satellite based on the pseudorange calculations. Additionally, it considers the positions of the satellites relative to the receiver.

- **Trilateration Calculation:**

Trilateration is performed to determine the receiver's location. By intersecting the spheres or circles with radii equal to the distances from the receiver to each satellite, the receiver's position can be determined at the intersection point.

- **Distance Calculation:**

Once the receiver's position is determined, the GPS receiver can calculate the distances between the receiver and various locations, such as waypoints or destinations. This calculation is based on the geometric principles and the known positions of the satellites.

By continuously receiving and processing signals from multiple satellites, the GPS receiver can update the calculated distances and the receiver's position in real-time. This enables accurate navigation, tracking, and distance calculations for various applications, including mapping, routing, and outdoor activities.

It is important to note that factors such as signal obstructions, atmospheric interference, and errors in timekeeping can introduce inaccuracies in distance calculations. GPS receivers employ advanced algorithms and techniques to mitigate these errors and provide reliable distance measurements.

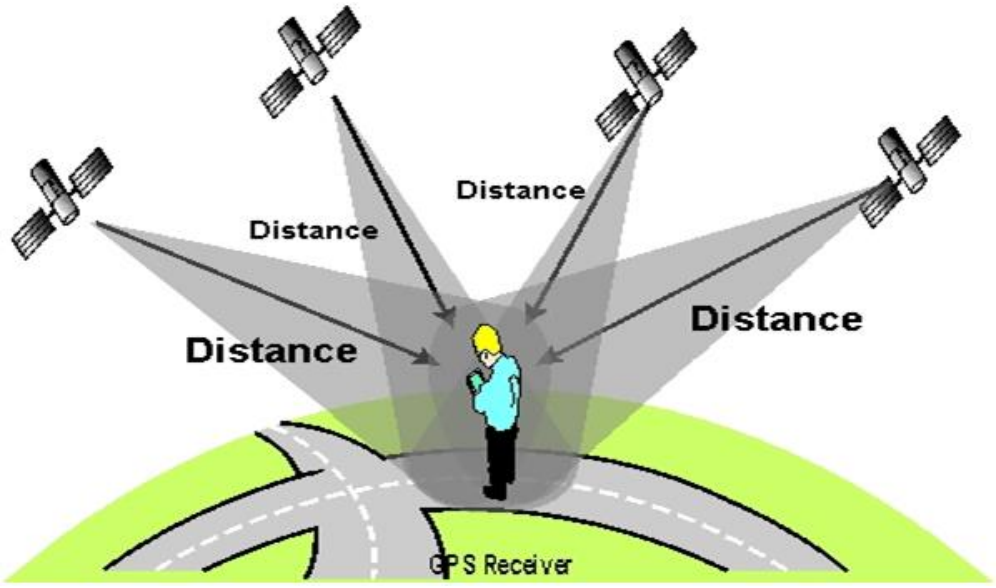



**Figure 15 GPS Distance calculation**

### 2.2.5.2) GPS module problems

- **Weak Satellite Signal Reception:**

The NEO-6M GPS module requires a clear view of the sky to receive signals from GPS satellites. If the module is obstructed by buildings, trees, or other obstacles, it may struggle to acquire or maintain a strong satellite signal. This can result in longer time-to-fix (TTFF) or intermittent signal reception, leading to reduced accuracy or even loss of positioning data.

- **Slow Time-to-Fix (TTFF):**

The time-to-fix refers to the time it takes for the GPS module to acquire satellite signals and calculate its position. In some cases, the NEO-6M module may have a relatively slow TTFF, especially when used in challenging environments with limited sky visibility or during initial power-up. Users may need to wait for a longer duration before the module can provide accurate positioning information.

- **Inaccurate Positioning Data:**

While the NEO-6M module generally provides reliable positioning data, it may occasionally exhibit slight inaccuracies. These inaccuracies can arise due to factors such as signal interference, multipath reflections, or errors in the module's internal clock. Users should be aware that small positioning errors can occur, especially in urban environments with tall buildings or areas with poor signal reception.

- **Limited Update Rate:**

The NEO-6M module has a maximum update rate of 5 Hz, meaning it can provide position updates up to 5 times per second. While this update rate is sufficient for most applications, it may not be ideal for high-speed or dynamic applications that require more frequent and real-time position updates.

- **Data Loss during Power Interruptions:**

In situations where the NEO-6M module experiences sudden power interruptions or voltage fluctuations, it may lose its settings or fail to resume proper functioning upon power restoration. This can require reinitialization or reconfiguration of the module to restore normal operation.

**2.2.5.3) Enhanced Location Detection and Emergency Response System**

In order to overcome the limitations associated with traditional GPS modules, an innovative solution has been developed to leverage a mobile application for precise location detection. This section explores the implementation of a sophisticated mobile application that efficiently detects user locations in two crucial scenarios. The first scenario involves detecting accident locations and notifying all users of the affected areas to ensure road safety. The second scenario focuses on promptly notifying nearby hospitals about the location of a user who has fainted due to a medical emergency. By seamlessly integrating the mobile application with a Raspberry Pi and a centralized server, this system enables rapid emergency response and enhances user safety on the road.

- **Accurate Location Detection during Road Accidents:**

In the event of a car accident, the mobile application, "Bluetooth GPS-Output," plays a pivotal role in obtaining precise location data. Leveraging the built-in Bluetooth capabilities of the mobile device, the application seamlessly transmits the location information to a Raspberry Pi. This Raspberry Pi serves as a communication hub and establishes a connection to a centralized server dedicated to handling emergency situations.

- **Emergency Alert Broadcasting and Road Closure Notification:**

Once the location data is received by the Raspberry Pi, it promptly relays the accident location information to the central server. The server, equipped with robust algorithms and advanced data processing capabilities, effectively disseminates the information to all users in the vicinity. This real-time notification system promptly alerts users about the accident, allowing them to avoid the affected

area and take alternative routes. Additionally, relevant authorities and emergency services can be promptly informed, enabling efficient incident management and ensuring the safety of all road users.

- **Rapid Emergency Response for Medical Emergencies:**

The mobile application's functionality extends beyond accident detection to address medical emergencies on the road. In cases where a user faints due to a sudden illness, the application leverages the same location detection capabilities. The moment the application detects an emergency situation, it triggers an immediate notification to the nearest hospital, providing them with the accurate location of the affected user. This enables healthcare professionals to respond swiftly, providing timely medical assistance and potentially saving lives

## 2.3) Other components

### 2.3.1) Servo motor


*Figure 16 servo motor*

A servo motor is a type of rotary or linear actuator that allows for precise control of angular or linear position , velocity, and acceleration. It is a closed-loop mechanism that incorporates positional feedback in order to control the rotational or linear speed and position . Servo motors are commonly used in a wide range of applications that require precise positioning and control, such as robotics, manufacturing, and automation.

unlike DC motors , with servo motors you can position the motor shaft at a specific position (angle) using control signal.

The motor shaft will hold at this position as long as the control signal not changed.

## Pinout:

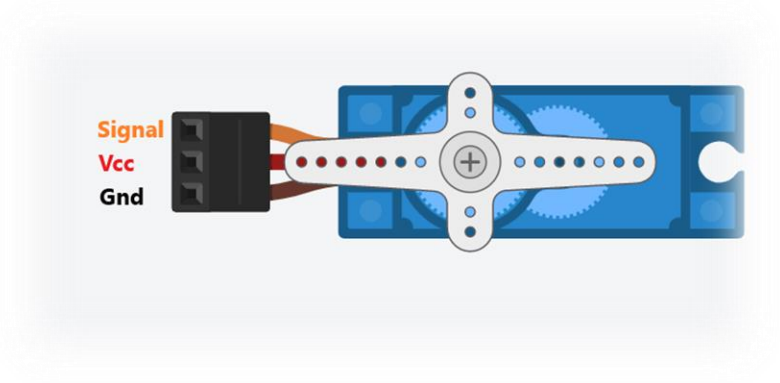



*Figure 17 servo motor pinout*

## Why Servo motor?

useful for controlling robot arms. Any object that you want it to move at certain angle and stay at its new position.

## How Does a Servo Motor Work?

- Usually a servo motor turns 90° in either direction, i.e. maximum movement can be 180°
- Three wires are taken out of a servo: positive, ground and control wire.
- A servo motor is controlled by sending a Pulse Width Modulated (PWM) signal through the control wire. A pulse is sent every 20 milliseconds.
- Width of the pulses determine the position of the shaft. For example, a pulse of 1ms will move the shaft anticlockwise at -90°, a pulse of 1.5ms will move the shaft at the neutral position that 0° and a pulse of 2ms will move the shaft clockwise at +90° (if the range is from 0° to 180° then a pulse of 1ms move the shaft at 0° , a pulse of 1.5ms will move the shaft at 90° and a pulse of 2ms will move the shaft at 180°)

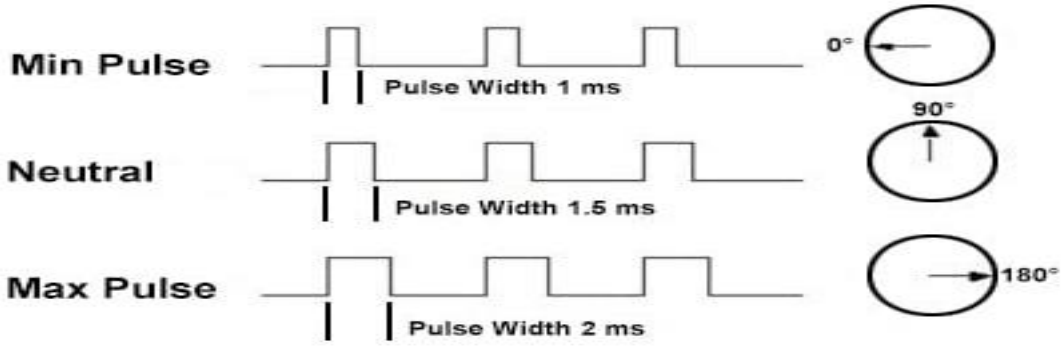



**Figure 18 servo motor pulses**

In our project, Servo motor is used at the garage unit to open up when a vehicle approaches the garage's gate.

### 2.3.2) Motor Drivers

Motor drivers acts as an interface between the motors and the control circuits. Motor require high amount of current whereas the controller circuit works on low current signals. So the function of motor drivers is to take a low-current control signal and then turn it into a higher-current signal that can drive a motor

#### 2.3.2.1) L298 Motor Driver:

L298 module is a high voltage, high current dual full-bridge motor driver module for controlling DC motor and stepper motor. It can control both the speed and rotation direction of two DC motors.

This module consists of an L298 dual-channel H-Bridge motor driver IC. This module uses two techniques for the control speed and rotation direction of the DC motors. These are PWM – For controlling the speed and H-Bridge – For controlling rotation direction. These modules can control two DC motor or one stepper motor at the same time. This motor driver module consists of two main key components, these are L298 motor driver IC and a 78M05 5V regulator.

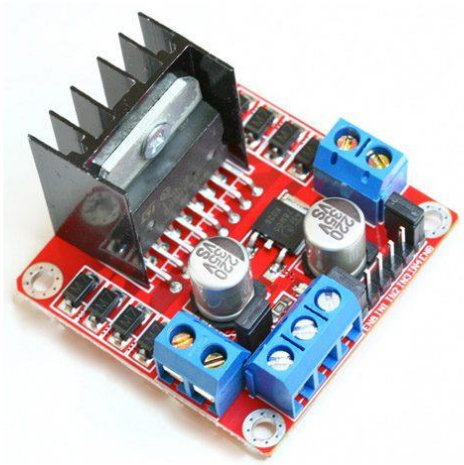


*Figure 19 L298 Motor Driver*

## Motor Driver Pinout:

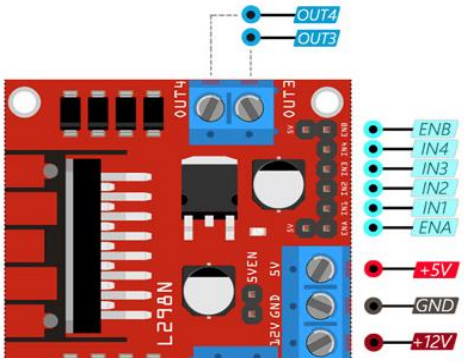



*Figure 20 Motor Driver Pinout*

| Pin No. | Pin Name | Description |
|---|---|---|
| **Power Supply Pins** | | |
| 1 | 12V | supply power to the motor. |
| 2 | GND | GND is a ground pin. |
| 3 | +5V | +5V pin supplies power for the switching logic circuitry inside the L298N IC. |
| **Control Pins** | | |
| 1 | IN1 | These pins are input pins of Motor A. These are used to control the rotating direction of Motor A. |
| 2 | IN2 | |
| 3 | IN3 | These pins are input pins of Motor B. These are used to control the rotating direction of Motor B. |
| 4 | IN4 | |
| **Speed Control Pins** | | |
| 1 | ENA | We need to connect this pin to a PWM input of the microcontroller. In that way, we can control the speed of Motor A |
| 2 | ENB | we need to connect this pin to a PWM input of the microcontroller. In that way, we can control the speed of Motor B. |
| **Output Pins** | | |
| 1 | OUT1 & OUT2 | This terminal block will provide the output for Motor A. |
| 2 | OUT3 & OUT4 | This terminal block will provide the output for Motor B. |

## Motor Direction:

| ENA | IN1 | IN2 | Description |
|---|---|---|---|
| **LOW** | X (any state) | X (any state) | Motor A is OFF |
| **HIGH** | LOW | LOW | Motor breaks and stops |
| **HIGH** | LOW | HIGH | Motor turns clockwise |
| **HIGH** | HIGH | LOW | Motor turns anti-clockwise |
| **HIGH** | HIGH | HIGH | Motor breaks and stops |

### 2.3.2.2) Cytron Dual Channel 10A DC Motor Driver (MDD10A):

MDD10A is the dual channel motor driver which is designed to drive 2 brushed DC

motors with high current up to 10A continuously, the MDD10A also supports locked-antiphase and sign-magnitude PWM signal. It is also using full solid state components which result in faster response time and eliminate the wear and tear of the mechanical relay.


*Figure 21 MDD10A*

**Motor Driver Pinout:**

| Pin No. | Pin Name | Description |
|---|---|---|
| | Input | |
| 1 | **GND** | GND is a ground pin. |
| 2 | ***PWM2** | PWM input for speed control (Motor 2) |
| 3 | **DIR2** | Direction input (Motor 2) |
| 4 | ***PWM1** | PWM input for speed control (Motor 1) |
| 5 | **DIR1** | Direction input (Motor 1) |
| | Terminal Block | |
| 1 | **Motor 1 Output B** | Connect to motor 1 terminal B |
| 2 | **Motor 1 Output A** | Connect to motor 1 terminal A |
| 3 | **POWER +** | Positive Supply (positive terminal of battery) |
| 4 | **POWER -** | Negative Supply (negative terminal of battery) |

| | | |
|---|---|---|
| 5 | **Motor 2 Output A** | Connect to motor 2 terminal A |
| 6 | **Motor 2 Output B** | Connect to motor 2 terminal B |

**Motor Direction:**

| PWM | DIR | OUTPUT A | OUTPUT B | Description |
|---|---|---|---|---|
| **LOW** | X (any state) | LOW | LOW | Motor is OFF |
| **HIGH** | LOW | HIGH | LOW | Motor turns clockwise |
| **HIGH** | HIGH | LOW | HIGH | Motor turns anti-clockwise |

### 2.3.3) Multiplexing

**Why Multiplexer?**

In garage unit, number of empty spaces are more than the number of timers and external interrupts in the used Microcontroller (atmega32), so due to resources shortage, we needed another way to cover all the ultrasonics used in the garage , so we used a Multiplexer to select an Ultrasonic's ECHO signal using only one external interrupt and one timer then measure its ECHO time to calculate the distance, then select the next Ultrasonic, by this way we can cover more than just 3 ultrasonics .

in our project we used four ultrasonics for four places in the garage, so needed 4x1 MUX

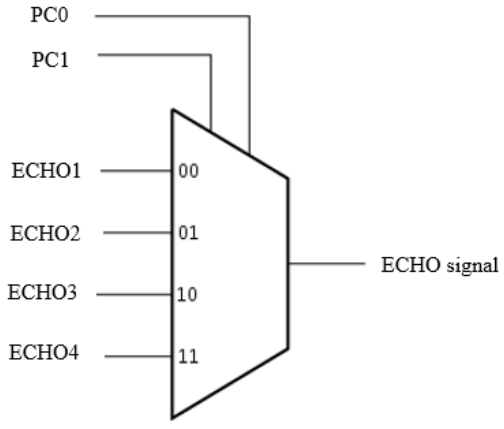



**Figure 22 Multiplexer**

**SN74LS157N:**


*Figure 23 SN74LS157N IC*

The SN74LS157N is an Integrated Circuit (IC) that belongs to the 74LS family of logic gates. It is a quad 2-line to 1-line data selector/multiplexer. This IC has four independent data selectors/multiplexers, each with two inputs (A and B) and one output (Y). It is commonly used in digital circuits for data selection, multiplexing, and demultiplexing applications.

**Truth Table:**

**TRUTH TABLE**

| ENABLE | SELECT INPUT | INPUTS | | OUTPUT |
|---|---|---|---|---|
| $\overline{E}$ | S | $I_0$ | $I_1$ | Z |
| H | X | X | X | L |
| L | H | X | L | L |
| L | H | X | H | H |
| L | L | L | X | L |
| L | L | H | X | H |

H = HIGH Voltage Level
L = LOW Voltage Level
X = Don't Care

**Figure 24 Multiplexer truth table**

We need three 2x1 MUX to use it as 4x1 MUX

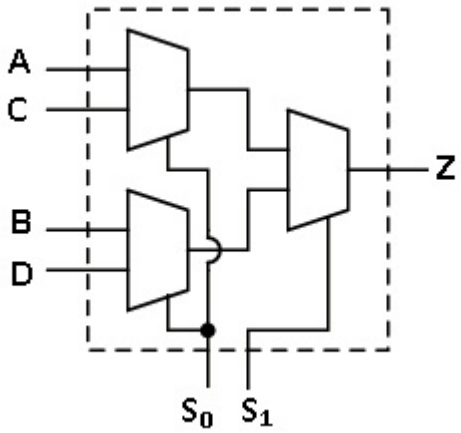



**Figure 25 2x1 MUX**

So we used two SN74LS157N ICs as each one has four 2x1 MUX with one selector , we needed one IC for each selector, we used two MUXs for the first selector S0 and one MUX for the second selector S1.

# CHAPTER 3
# Communication Protocols

## 3.1) Introduction

Wired communication protocols are important for several reasons:

**Reliability**: Wired communication protocols offer a higher level of reliability compared to wireless protocols. They are not affected by interference from other devices or environmental factors like walls or distance. This makes wired protocols suitable for critical applications where uninterrupted and error-free data transmission is crucial.

**Security:** Wired protocols provide a more secure means of communication compared to wireless protocols. Since wired connections are physical and localized, it is more difficult for unauthorized parties to intercept or access the data being transmitted. This is particularly important for sensitive information such as financial transactions or confidential data transfers.

**Speed and Bandwidth:** Wired protocols generally offer higher data transfer speeds and greater bandwidth compared to wireless protocols. This makes them ideal for applications that require large amounts of data to be transmitted quickly, such as video streaming, file transfers, or real-time communication.

**Low Latency:** Wired protocols typically have lower latency compared to wireless connections. Latency refers to the delay between the transmission and reception of data. Low latency is important for real-time applications like online gaming, video conferencing, or remote control systems, where even minor delays can affect the user experience or operational efficiency.

**Compatibility:** Wired communication protocols often have well-defined standards, ensuring compatibility between different devices and systems. This allows for seamless integration and communication between various devices, facilitating interoperability in complex networks or interconnected systems.

# 3.2)UART

## Abstract

UART, or universal asynchronous receiver-transmitter, is one of the most used device-to-device communication protocols.

When properly configured, UART can work with many different types of serial protocols that involve transmitting and receiving serial data. In serial communication, data is transferred bit by bit using a single line or wire. In two-way communication, we use two wires for successful serial data transfer. Depending on the application and system requirements, serial communications needs less circuitry and wires, which reduces the cost of implementation.

### 3.2.1 ) Interface

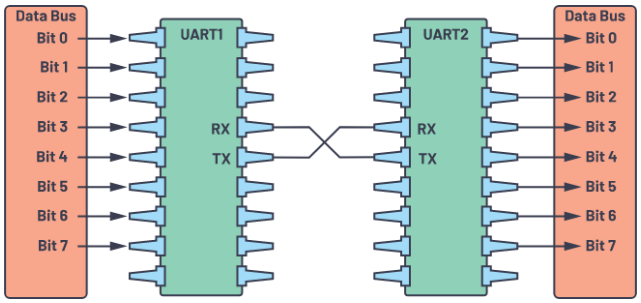



**Figure 26 UART with data bus.**

The transmitting UART is connected to a controlling data bus that sends data in a parallel form. From this, the data will now be transmitted on the transmission line (wire) serially, bit by bit, to the receiving UART. This, in turn, will convert the serial data into parallel for the receiving device.

For UART and most serial communications, the baud rate needs to be set the same on both the transmitting and receiving device. The baud rate is the rate at which information is transferred to a communication channel. In the serial port context, the set baud rate will serve as the maximum number of bits per second to be transferred.

### 3.2.2) Structure of a UART Protocol

In UART, the mode of transmission is in the form of a packet. The piece that connects the transmitter and receiver includes the creation of serial packets and controls those physical hardware lines. A packet consists of a start bit, data frame, a parity bit, and stop bits.


**Figure 27 UART packet.**

**Start bit**

The UART data transmission line is normally held at a high voltage level when it's not transmitting data. To start the transfer of data, the transmitting UART pulls the transmission line from high to low for one (1) clock cycle. When the receiving UART detects the high to low voltage transition, it begins reading the bits in the data frame at the frequency of the baud rate.

**Data frame**

The data frame contains the actual data being transferred. It can be five (5) bits up to eight (8) bits long if a parity bit is used. If no parity bit is used, the data frame can be nine (9) bits long. In most cases, the data is sent with the least significant bit first.

**Parity bits**

Parity describes the evenness or oddness of a number. The parity bit is a way for the receiving UART to tell if any data has changed during transmission. Bits can be changed by electromagnetic radiation, mismatched baud rates, or long-distance data transfers.

After the receiving UART reads the data frame, it counts the number of bits with a value of 1 and checks if the total is an even or odd number. If the parity bit is a 0 (even parity), the 1 or logic-high bit in the data frame should total to an even number. If the parity bit is a 1 (odd parity), the 1 bit or logic highs in the data frame should total to an odd number.

When the parity bit matches the data, the UART knows that the transmission was free of errors. But if the parity bit is a 0, and the total is odd, or the parity bit is a 1, and the total is even, the UART knows that bits in the data frame have changed.

**Stop bits**

To signal the end of the data packet, the sending UART drives the data transmission line from a low voltage to a high voltage for one (1) to two (2) bit(s) duration.

### 3.2.3) UART Operations

**UART Data Transmission Steps:**

1. Enable the USART: Set the UE (USART Enable) bit in the USART_CR1 (Control Register 1) to enable the USART module.

2. Set Word Length: Program the M (Word Length) bits in USART_CR1 to define the length of the data word (typically 8 or 9 bits).

3. Configure Stop Bits: Program the number of stop bits in USART_CR2 (Control Register 2) to specify the number of stop bits to be used (usually 1 or 2).

4. Set Baud Rate: Select the desired baud rate by configuring the USART_BRR (Baud Rate Register) with the appropriate values to match the desired communication speed.

5. Enable Transmitter: Set the TE (Transmitter Enable) bit in USART_CR1 to enable the transmitter. This will initiate the transmission by sending an idle frame as the first transmission.

6. Write Data: Write the data to be transmitted into the USART_DR (Data Register). This clears the TXE (Transmit Data Register Empty) bit. Repeat this step for each data to be transmitted in case of a single buffer.

7. Wait for Completion: After writing the last data into the USART_DR register, wait until the TC (Transmission Complete) bit is set to 1. This indicates that the transmission of the last frame is complete, ensuring the data has been fully transmitted.

### 3.2.4) UART Receiver Operations:

1. Enable the USART: Set the UE bit in USART_CR1 to enable the USART module.

2. Set Word Length: Program the M bit in USART_CR1 to define the word length for data reception.

3. Configure Stop Bits: Program the number of stop bits in USART_CR2.

4. Set Baud Rate: Select the desired baud rate using the USART_BRR register.

5. Enable Receiver: Set the RE (Receiver Enable) bit in USART_CR1. This enables the receiver, allowing it to search for a start bit and begin data reception.

During data reception, the RXNE (Receive Data Register Not Empty) bit is set when a character is received. It indicates that the content of the shift register is transferred to the RDR (Receive Data Register). This means that data has been received and can be read, along with its associated error flags. An interrupt can be generated if the RXNEIE (Receive Data Register Not Empty Interrupt Enable) bit is set, allowing for timely handling of received data.

### 3.2.5) Interfacing with UART

In this project, the UART protocol is utilized for interfacing between various components, enabling data transmission and reception. The following interactions employ UART:

1. **Raspberry Pi and STM:**

    - Data Transmission: The Raspberry Pi and STM (microcontroller) communicate with each other using UART protocol.

    - Raspberry Pi as Receiver: The Raspberry Pi receives vehicle data output from the STM, which is then sent to the server for further processing.

    - Raspberry Pi as Transmitter: The Raspberry Pi receives data from the server and transmits it to the STM for processing and analysis.

2. **Bluetooth Module and STM:**

    - Data Transmission: The UART protocol facilitates communication between the Bluetooth module and the STM microcontroller.

    - Control of Vehicle Motion: A Bluetooth RC Controller mobile application is used to control the motion of the vehicle.

    - Bluetooth Module as Transmitter: The Bluetooth module transmits control commands wirelessly to the STM microcontroller.

- STM as Receiver: The STM microcontroller receives the control commands from the Bluetooth module, enabling vehicle motion control.

By utilizing UART for these interfaces, seamless and reliable communication is achieved between the Raspberry Pi and STM microcontroller, as well as between the Bluetooth module and STM. This enables the project to effectively transmit and receive data, control vehicle motion, and integrate with external systems for enhanced functionality.

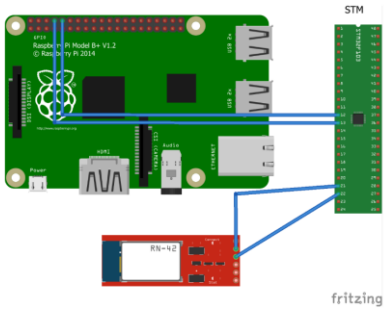



**Figure 28 Interfacing between STM and both Raspberry Pi and Bluetooth module.**

## 3.3) I2C

The name IIC is shorthand for a standard Inter-IC (integrated circuit) bus.

With I2C, you can connect multiple slaves to a single master and you can have multiple masters controlling single, or multiple slaves. This is really useful when you want to have more than one microcontroller logging data to a single memory card or displaying text to a single LCD.

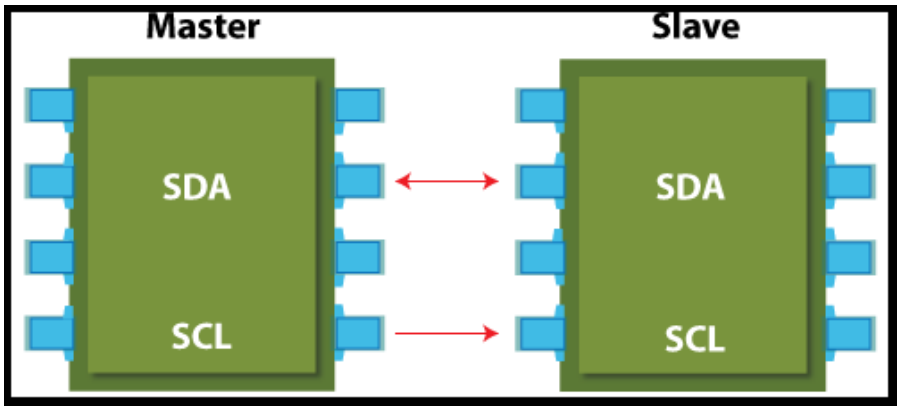



**Figure 29 I2C**

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

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

I2C is synchronous, so the output of bits is synchronized to the sampling of bits by a clock signal shared between the master and the slave. The clock signal is always controlled by the master.

I2C is a two-wire serial bus, as shown in the Figure. There's no need for chip select or arbitration logic, making it cheap and simple to implement in hardware.

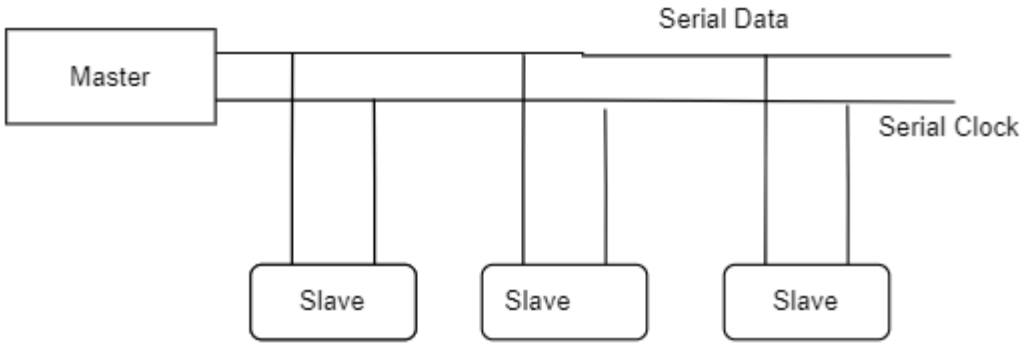



**Figure 30 Master and slave**

### 3.3.1) I2C Protocol Working Principle:

The working of the I2C communication protocol happens through open drain lines which are Serial Data (SDA) and SCL (Serial Clock). Initially, both the SDA and SCL lines are pulled high and the bus mainly functions in two modes which are Master and Slave.

### 3.3.2) HOW I2C WORKS:

With I2C, data is transferred in messages. Messages are broken up into frames of data. Each message has an address frame that contains the binary address of the slave, and one or more data frames that contain the data being transmitted. The message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame:

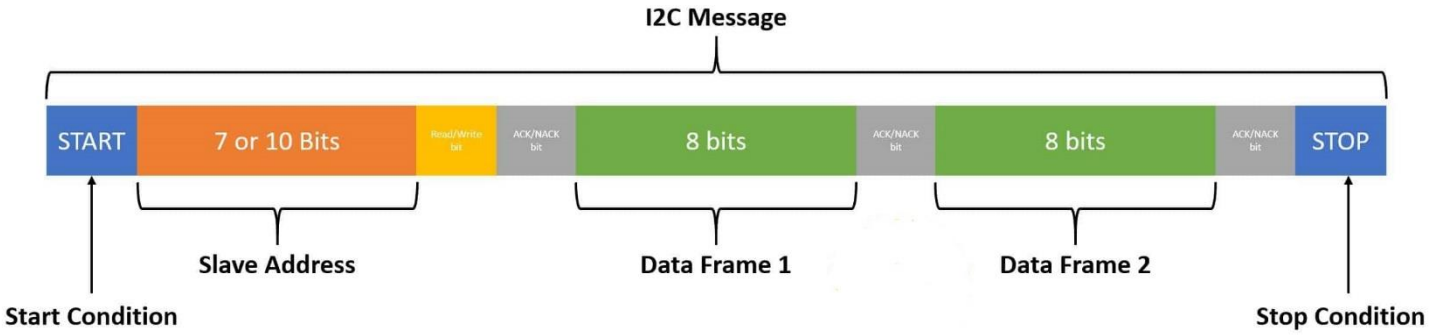



**Figure 31 How I2C works**

1. **Start Condition:** The SDA line switches from a high voltage level to a low voltage level before the SCL line switches from high to low.

2. **Address Frame:** A 7 or 10 bit sequence unique to each slave that identifies the slave when the master wants to talk to it, I2C doesn't have slave select lines like SPI, so it needs another way to let the slave know that data is being sent to it, and not another slave. It does this by addressing.

3. **Read/Write Bit:** A single bit specifying whether the master is sending data to the slave (low voltage level) or requesting data from it (high voltage level).

4. **ACK/NACK Bit:** Each frame in a message is followed by an acknowledge/no-acknowledge bit. If an address frame or data frame was successfully received, an ACK bit is returned to the sender from the receiving device.

5. **Data Frame:** After the master detects the ACK bit from the slave, the first data frame is ready to be sent.

6. **Stop Condition:** The SDA line switches from a low voltage level to a high voltage level after the SCL line switches from low to high.

### 3.3.3) Advantages and Disadvantages:

The I2C protocol advantages and disadvantages are as follows:

**Advantages:**

- Even though there are multiple devices on the bus, the signals required are less
- It has the ability to support many masters
- It has enhanced adaptability where the requirements of various slave devices can be achieved
- It has multiple-master and multiple-slave flexibility
- I2C protocol has no complications in design

**Disadvantages:**

- I2C protocol is designed with an open-drain configuration where it limits the speed and also pull-up resistors are used which further lessens the communication speed
- It needs more space

### 3.3.4) The use of I2C in our project:

I2C is used with the sensors like MPU6050, STM is the master and MPU6050 is the slave

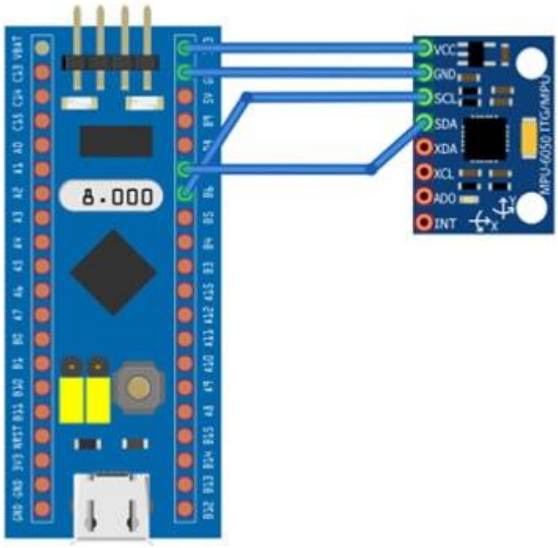



**Figure 32 I2C in our project**

Stm32f103 MCU sends a request with I2C with the address of MPU6050, MPU6050 responds to that request.

## 3.4) SPI Serial Peripheral Interface (SPI):

Almost all modern microcontrollers featuring build in SPI communication protocol system. SPI is master multi-slave synchronous serial protocol that needs four wires ,MISO (Master input slave output) , MOSI (Master output slave input) , SCLK (Serial Clock) and NSS (negative slave select), for communication.

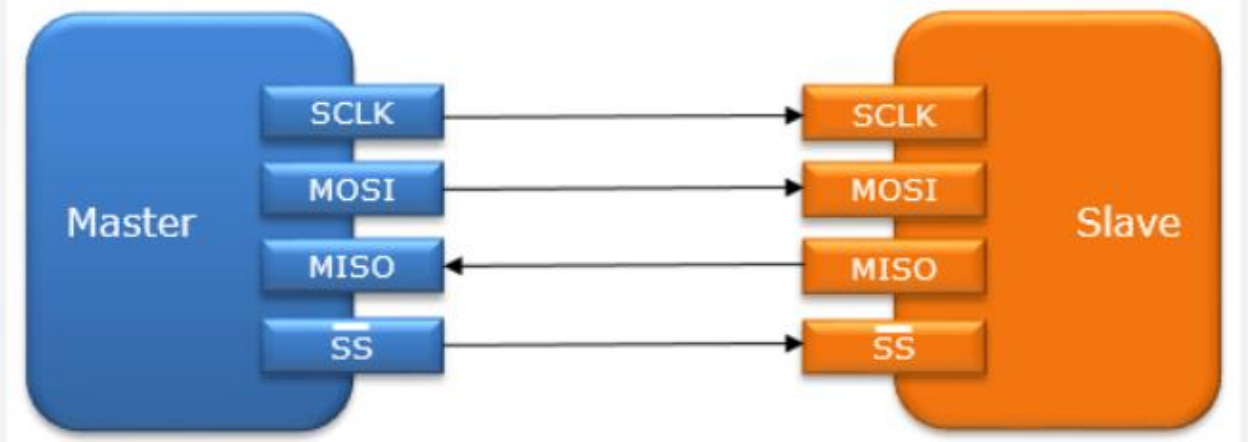



**Figure 33 SPI**

### 3.4.1 ) How SPI works:

SPI communication typically begins with the master device selecting the slave device by setting its corresponding NSS line low. The master then sends data to the slave by toggling the SCLK line, with each clock pulse indicating the transfer of one bit of data on the MOSI line. The slave device receives the data on the MOSI line and sends its response back to the master on the MISO line. The data transfer continues until the master device sets the NSS line high again to deselect the slave device and end the communication.

### 3.4.2) Data frame format:

Data can be shifted out either MSB-first or LSB-first depending on the value of the LSBFIRST bit in the SPI_CR1 Register.

Each data frame is 8 or 16 bits long depending on the size of the data programmed using the DFF bit in the SPI_CR1 register. The selected data frame format is applicable for transmission and/or reception.

### 3.4.3) SPI in stm32 :

STM32F103C6T6 Blue Pill has one SPI interface with speed 18Mbits/s the pin mapping:

MOSI→ PB5 or PA7

MISO→ PB4 or PA6

SCK→ PB3 or PA5

NSS→ PA15 or PA4

SPI main Features:

- Full-duplex synchronous transfers on three lines
- Simplex synchronous transfers on two lines with or without a bidirectional data line
- 8- or 16-bit transfer frame format selection
- Master or slave operation
- 8 master mode baud rate prescalers (fPCLK/2 max.)
- Slave mode frequency (fPCLK/2 max)
- Faster communication for both master and slave
- NSS management by hardware or software for both master and slave: dynamic change of master/slave operations
- Programmable clock polarity and phase
- Programmable data order with MSB-first or LSB-first shifting
- Dedicated transmission and reception flags with Interrupt capability
- SPI bus busy status flag

The interface can operate in one of the four following modes:

1. Slave Mode
2. Master Mode

# CHAPTER 4
# Vehicle Unit

# 4.1) Introduction

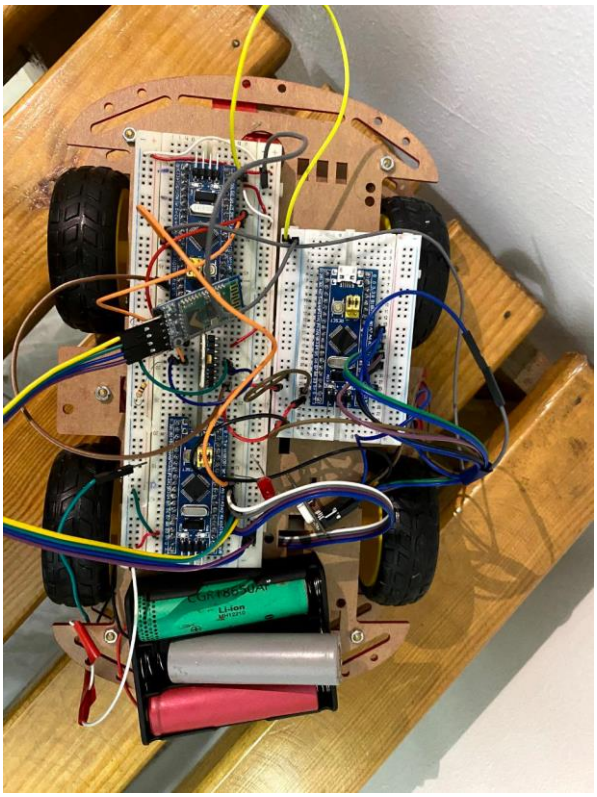
## Objective

V2V units aim to enhance road safety by enabling vehicles to communicate with each other. They provide early warning systems, support collision avoidance, and enable cooperative driving strategies. V2V units also contribute to intelligent traffic management, facilitate emergency vehicle priority, and improve pedestrian and cyclist safety. They enable vehicle-to-infrastructure communication, foster innovation, and promote standardization and interoperability in the automotive industry. Overall, V2V units play a vital role in creating a connected ecosystem that enhances road safety and shapes the future of mobility.

## Solution overview

V2V communication in V2X projects aims to create a connected ecosystem where vehicles can communicate, collaborate, and interact with their surroundings. By enhancing safety, improving traffic efficiency, and enabling new services and applications, V2V communication revolutionizes the automotive industry, paving the way for a safer, smarter, and more efficient transportation system.

# Hardware real pictures



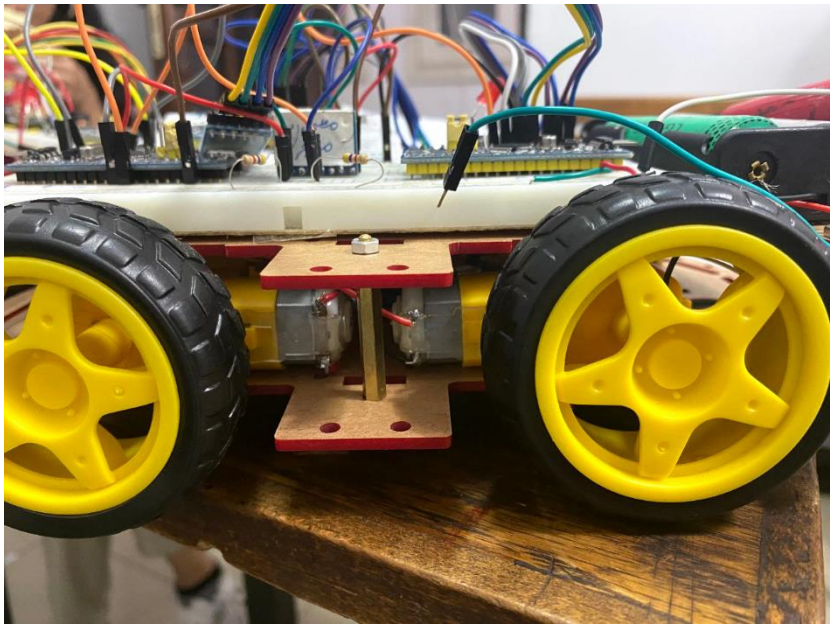
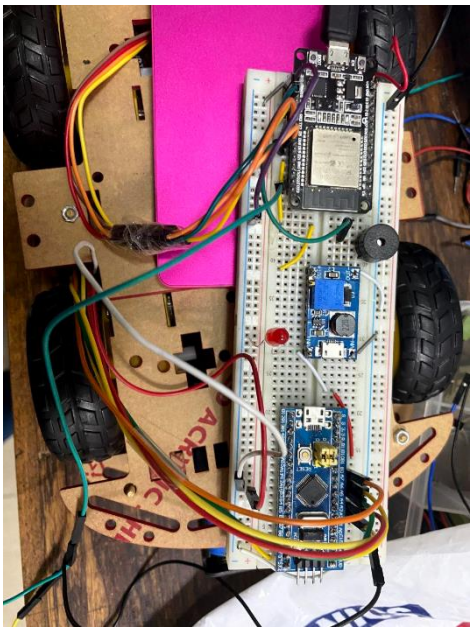


*Figure 35 First vehicle*



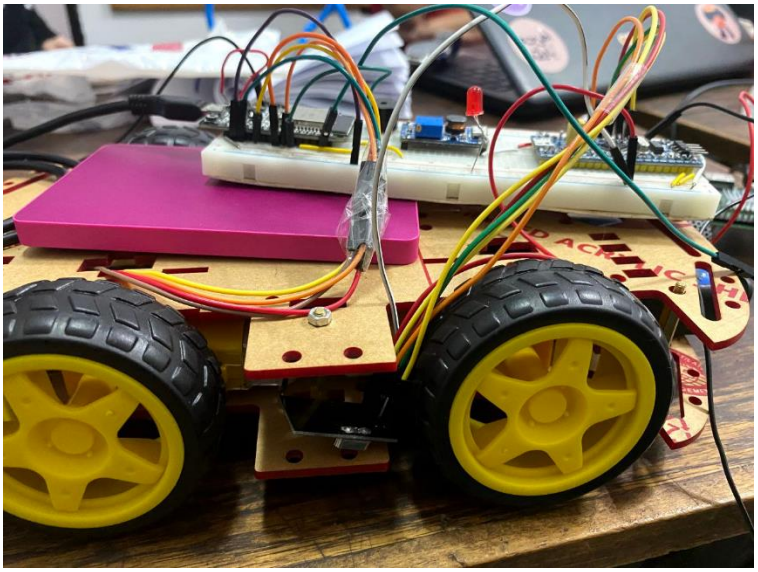


*Figure 34 second vehicle*

## 4.3) Accident handling

This process is consist of main parts :

- Detecting there is an accident
- Dealing with this accident when it occur

### 4.3.1) Accident detection

As we discussed before the IMU and the Ultrasonic sensors are connected to the Blue Pill "stm32f103c6t6" board

And their function appears in this case:

- When the IMU sensor suddenly detect change in the acceleration, from zero acceleration as the vehicle move with constant speed to negative acceleration, it sense that there is an accident.
- But also to confirm that there is an accident, The Ultrasonic sensor was used for the purpose of detecting obstructions in front of the vehicle.

And these two processes are done in the same time, In parallel to each other.

By the outputs of the IMU and Ultrasonic sensors, The blue Pill confirmed that there is an accident was detected.

### 4.3.2) Dealing with the accident

- The blue pill board is connected to the Raspberry Pi by UART communication protocol and the Raspberry Pi forwards all the data to the sever.

- When the accident is occurred, There is two processes are done in parallel:

    - The server sends signals to the nearest hospital to the location of the vehicle to inform it that there is an accident and send the location of the accident to it.
    - By nRF the vehicle inform all the nearby vehicles about the accident and decrease their speed till all vehicles stop.

## 4.4) Communication between cars

Small range communication between vehicles is becoming increasingly important in emergency scenarios, where accidents can result in multiple vehicle collisions and traffic congestion, leading to waste of time, money, and even lives. To address this issue, our team has developed a system using nRF24L01+ modules, 2.4 GHz transceivers, to enable fast and efficient communication between vehicles in proximity.

The nRF24L01+ module is a low-cost wireless transceiver that operates in the 2.4 GHz frequency band and provides a range of up to 100 meters in open space. It uses a proprietary protocol called Enhanced ShockBurst to enable efficient and reliable communication, with data rates of up to 2 Mbps and a channel bandwidth of 1 MHz. This makes it an ideal choice for small range communication between vehicles, where low latency and high reliability are critical.

In our system, each vehicle is equipped with an nRF24L01+ module and an MCU (microcontroller unit) that controls the operation of the module. When an accident occurs, the MCU in the vehicle that is closest to the accident becomes the PTX (primary transmitter) and enables the nRF24L01+ module to send one byte of data to the receiver's address. We have chosen to use the data link layer protocol shock burst and physical layer wireless in ISA band of 2.4 GHz with channel bandwidth of 1 MHz and a data rate of 2 Mbps to reduce delays.

In the receiver side of the system, we have assumed for simplicity that the second vehicle is always the PRX (primary receiver). When the nRF24L01+ module detects a message with a valid address equal to the address in the address data pipe register, it accepts the message and sends it to the master STM32F103C6T6 Blue Pill. When the message arrives at the MCU, the adaptive cruise control system adjusts the speed to prevent another accident. In real-life scenarios, this vehicle becomes the PTX to the vehicle in the rear, and so on, in addition to sending a message to a cloud-based server to close the road in the city maps to prevent traffic congestion. One of the key advantages of using the nRF24L01+ module for communication between vehicles is its low latency and high reliability. By avoiding cloud servers and using a direct wireless communication link, we can reduce delays and ensure that messages are transmitted and received quickly and reliably. This is especially important in emergency scenarios where immediate action may be necessary to prevent further accidents or injuries. Another advantage of our system is its simplicity and low cost. The nRF24L01+ module is an inexpensive and widely available component that can be easily integrated into existing vehicle systems. Additionally, the shock burst data link layer protocol used in our system is simple and efficient, allowing for fast and reliable transmission of data.

Our system also has the potential to enhance overall safety on the road beyond emergency scenarios. For example, it can be used to communicate information about traffic flow, road conditions, and other hazards in real-time, allowing drivers to adjust their behavior accordingly and avoid accidents.

Of course, there are also some challenges to be considered when implementing a small range communication system between vehicles. One of the key challenges is ensuring compatibility between different vehicles and communication protocols. Additionally, there are privacy and security concerns to be addressed when transmitting sensitive information between vehicles. To address these challenges, it is important to develop standardized protocols and hardware interfaces that can be used across different vehicle types and manufacturers. Additionally, strong encryption and authentication mechanisms should be used to ensure the privacy and security of transmitted data. Despite these challenges, the potential benefits of small range communication between vehicles using the nRF24L01+ module are significant. By enabling fast and efficient communication between vehicles in close proximity, we can reduce the risk of multiple vehicle collisions and traffic congestion, leading to improved safety and efficiency on the road.


**Figure 36 communication between cars**

## 4.5) Driver monitor system

We have successfully integrated a cutting-edge driver monitor system into our vehicles, aimed at ensuring utmost safety and well-being of our users. This advanced system is designed to detect any signs of drowsiness or fatigue in the driver's health, particularly during long highway journeys.

Utilizing state-of-the-art technology, such as the Raspberry Pi, we are able to accurately identify instances of drowsiness in real-time. Once a significant level of drowsiness is detected, an immediate

signal is transmitted to our system. This signal acts as a prompt for our vehicles to take proactive measures in order to safeguard the driver and other road users.

In response to the detected drowsiness, our vehicles promptly initiate a series of precautionary actions. Firstly, the vehicle's speed is automatically reduced to ensure safer driving conditions. Additionally, a strategically positioned buzzer within the vehicle is activated at regular intervals of two seconds. This audible alert serves as an effective means of awakening the driver from their drowsy state.

By implementing this comprehensive driver monitor system, we prioritize the well-being and safety of our users above all else. Our commitment to utilizing cutting-edge technology ensures that potential risks associated with driver fatigue are mitigated effectively.

# CHAPTER 5
# Garage Unit

## 5.1) Introduction

### Objective

Implement an intelligent parking system utilizing AVR ATmega32 microcontroller and Raspberry Pi technology to monitor the availability of parking slots in a garage. The system aims to provide real-time updates on slot availability by utilizing ultrasonic sensors connected to the AVR ATmega32 microcontroller. This data will then be transmitted to a Raspberry Pi, which will further send the information to a server for remote access by users.

### Solution Overview

Our solution involves the integration of AVR ATmega32 microcontroller, ultrasonic sensors, Raspberry Pi, and server communication to create a robust smart parking system. By utilizing AVR ATmega32 and ultrasonic sensors, we can accurately detect and monitor the occupancy status of each parking slot. The data collected will be transmitted to the Raspberry Pi, which will facilitate communication with a centralized server, making the parking slot availability information accessible to users remotely.

## 5.2) PCB and garge implementation

in our garage we have 4 ultrasonics sensors each one has four wires to operate and we have a 16x2 LCD which need a 16 wire to display our output and also we have a servo motor which need to connect the microcontroller with 3 wires also we have one raspberry pi which contact the microcontroller with 2 wires for communication so we have about 40 wire in the garage unit which is very hard for these reasons

- It can be unreliable
- It’s difficult to replicate versus other circuit boards
- It’s a temporary solution
- It’s heavier than other circuit boards
- Breadboards can be noisier than other boards
- Difficult to troubleshoot because of human error
- Poor connection problems
- Not feasible for high current applications
- Cannot withstand voltages above 48 V

- Poorly setup for high-speed design
- Difficult to modify complex components for prototyping
- Takes up more physical space
- And we think to implement our PCB for the following reasons
- PCBs have low cost and easy mass production
- They’re reworkable and widely available
- PCBs have excellent shelf life and low electronic noise
- They feature compact sizes
- Inspection time and error is reduced in PCBs
- Less time assembling than other circuit boards
- No chance of loose connections
- Less short circuiting errors
- Easy to replace failing components
- Uniformity in shared electrical characteristics

Our PCB Designed By Eagle Software To Solve The Connection Problems

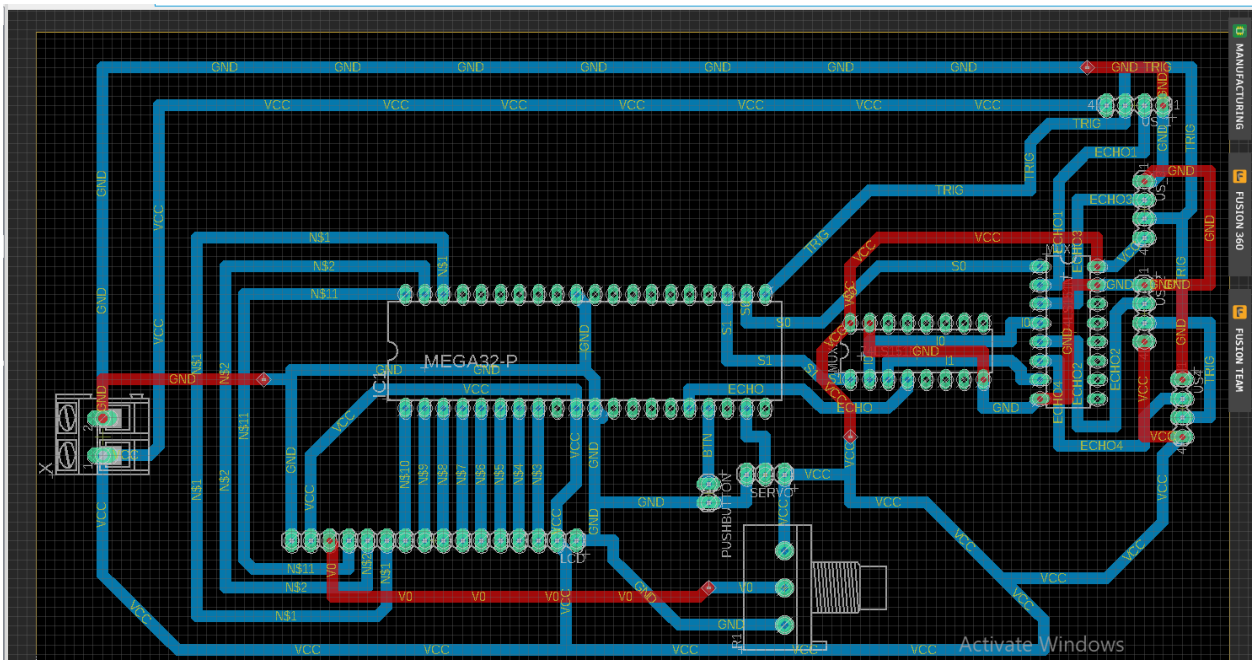



**Figure 37 Garage PCB**

## 5.3) Resources shortage and TDM sloution

In garage unit, number of empty spaces are more than the number of timers and external interrupts in the used atmega32 Microcontroller (it has only three external interrupts and only three timers just one of them (Timer1) has ICU (Input Capture Unit))

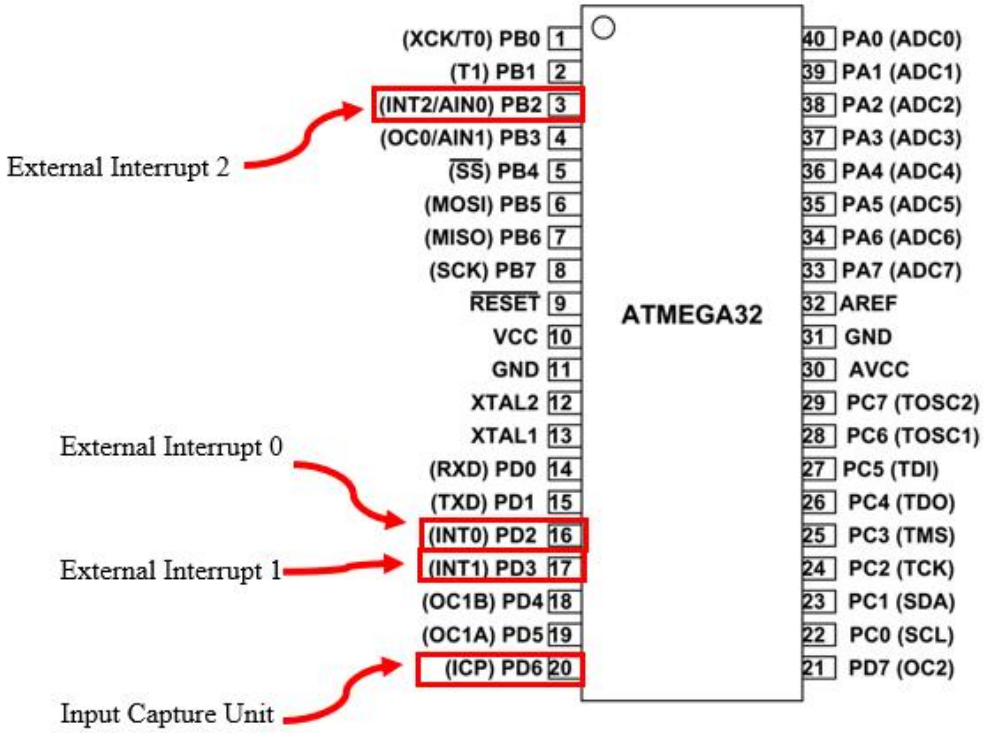



**Figure 38 Atmega32 PINOUT**

Due to the lack of resources , this microcontroller won't be able to serve more than three vehicles at the same time ,So there are two solutions to solve this problem:

1. Use more than just one microcontroller, this solution is more expensive
2. The other solution is to use only one external interrupt and one timer and divide the time into slots , each ultrasonic has a time slot

In out project we used the second solution to serve all the places in the Garage, we used a Multiplexer to select an Ultrasonic's ECHO signal using only one external interrupt and one timer then measure its ECHO time to calculate the distance, then select the next Ultrasonic, by this way we can cover more than just three ultrasonics .

This way looks like Time Division Multiplexing in the Modulation Process in Communication as the time is divided into slots, each Ultrasonic has a time slot about 500mSec.

## 5.4) Reservation system

### 5.4.1) Cloud server

We designed an application to help drivers to reserve a slot in neighboring garage if there is a free slot application verifies that. if not, application notify the driver

to join reserved slot:

A message arrived at the raspberry pi in the garage when reservation is required the raspberry pi communicate with the AVR microcontroller (atmega 32) using UART communication protocol to asks the number of free slots then RPI replay with reservation confirmation if there is a free slot and driver wants to reserve a slot the MCU update the number of slots if the confirmation is arrived to send the new number of slots to RPI if a new reservation is needed.

The message sent from the application to the RPI have the id inside it.

### 5.4.2) LP detection

The entrance to the garage is equipped with a servo motor that acts as a barrier gate. The gate is wide enough to allow two cars to pass through at the same time.

When a car approaches the gate, the camera at the entrance scans the label on the car. The label contains the car's ID number, which is used to identify the car in the reservation system.

The camera uses computer vision software to identify the characters on the label. The computer vision software is typically written in Python, and it is running on a laptop then sending the output to the Raspberry Pi.

Once the computer vision software has identified the car's ID number, it compares the number to the list of reservations. The list of reservations is typically stored in a database. If the car's ID number is on the list, the Raspberry Pi will send an output signal to the servo motor.

The servo motor will then open the gate. The gate will remain open for a short period of time, typically a few seconds. This is enough time for the car to enter the garage. Once the car has entered the garage, the gate will close.

The combination of a Raspberry Pi and a camera provides a secure and efficient way to control access to a garage. This system can be used to improve security, reduce traffic congestion, and save time for drivers.

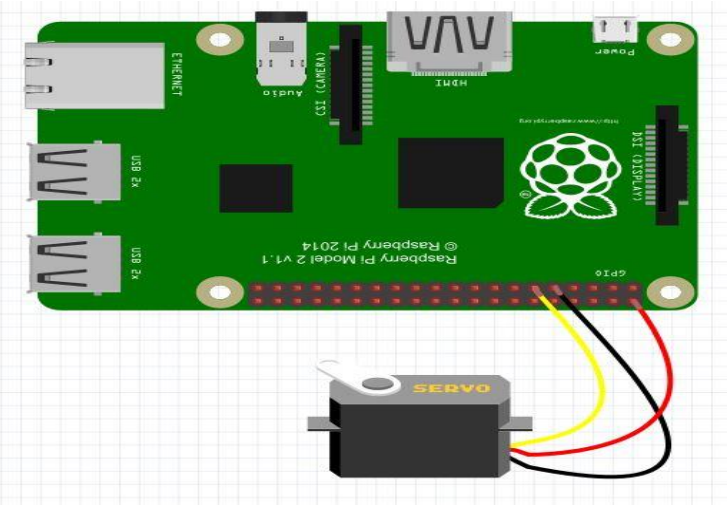



**Figure 39 Servo Motors With RPI**

# CHAPTER 6
# Machine learning

## 6 .1) Machine learning and Deep learning.

Machine learning and deep learning are both types of AI. In short, **machine learning** is AI that can automatically adapt with minimal human interference. **Deep learning** is a subset of machine learning that uses artificial neural networks to mimic the learning process of the human brain.

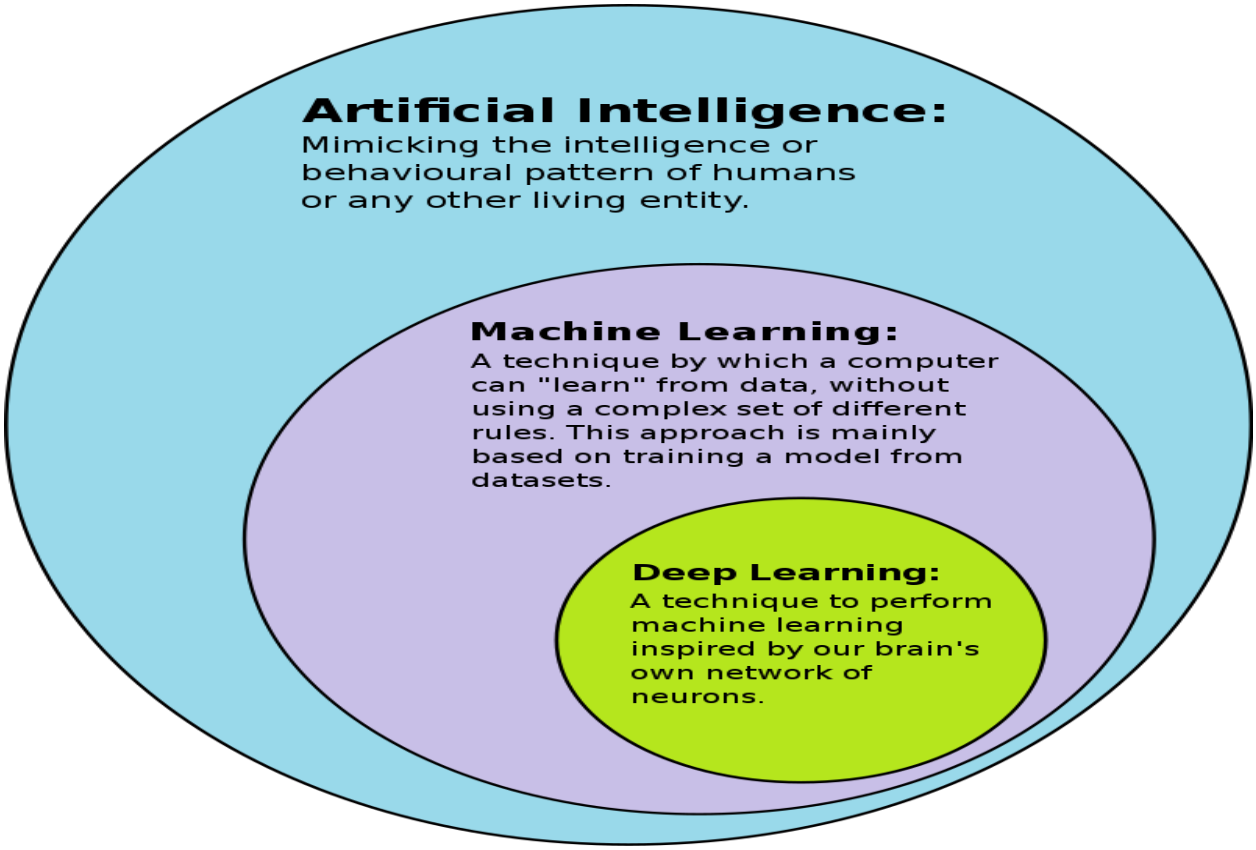



**Figure 40 AI VS. Machine learning VS. Deep learning.**

## 6.2) Deep learning works

Deep learning neural networks attempts to mimic the human brain through a combination of data inputs, weights, and bias. These elements work together to accurately recognize, classify, and describe objects within the data.

Deep neural networks consist of multiple layers of interconnected nodes, each building upon the previous layer to refine and optimize the prediction or categorization. This progression of computations through the network is called **forward propagation**. The input and output layers of a deep neural network are

called **visible layers**. The input layer is where the deep learning model ingests the data for processing, and the output layer is where the final prediction or classification is made.

Another process called **backpropagation** uses algorithms, like gradient descent, to calculate errors in predictions and then adjusts the weights and biases of the function by moving backwards through the layers in an effort to train the model. Together, forward propagation and backpropagation allow a neural network to make predictions and correct for any errors accordingly. Over time, the algorithm becomes gradually more accurate.

There are different types of neural networks to address specific problems or datasets. For example,

- Convolutional neural networks (CNNs), used primarily in computer vision and image classification applications, can detect features and patterns within an image, enabling tasks, like object detection or recognition. In 2015, a CNN bested a human in an object recognition challenge for the first time.

- Recurrent neural network (RNNs) are typically used in natural language and speech recognition applications as it leverages sequential or times series data.

## 6.3) Convolutional neural networks (CNNs)

CNN is a type of deep learning model for processing data that has a grid pattern, such as images and adaptively learn spatial hierarchies of features, from low- to high-level patterns.

CNN is a mathematical construct that is typically composed of three types of layers (or building blocks):

- convolution layer.
- pooling layer.
- fully connected layer.

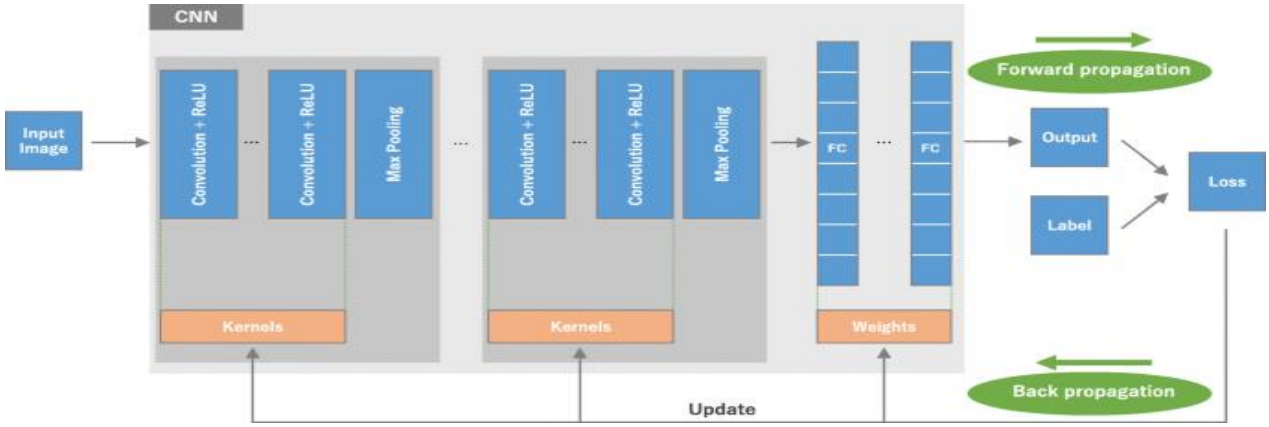



**Figure 41** **An overview of a convolutional neural network (CNN) architecture and the training process.**

In digital images, pixel values are stored in a two-dimensional (2D) grid, an array of numbers (Fig. 3), and a small grid of parameters called kernel (Filter), an optimizable feature extractor, is applied at each image position, which makes CNNs highly efficient for image processing, since a feature may occur anywhere in the image. As one layer feeds its output into the next layer, extracted features can hierarchically and progressively become more complex.

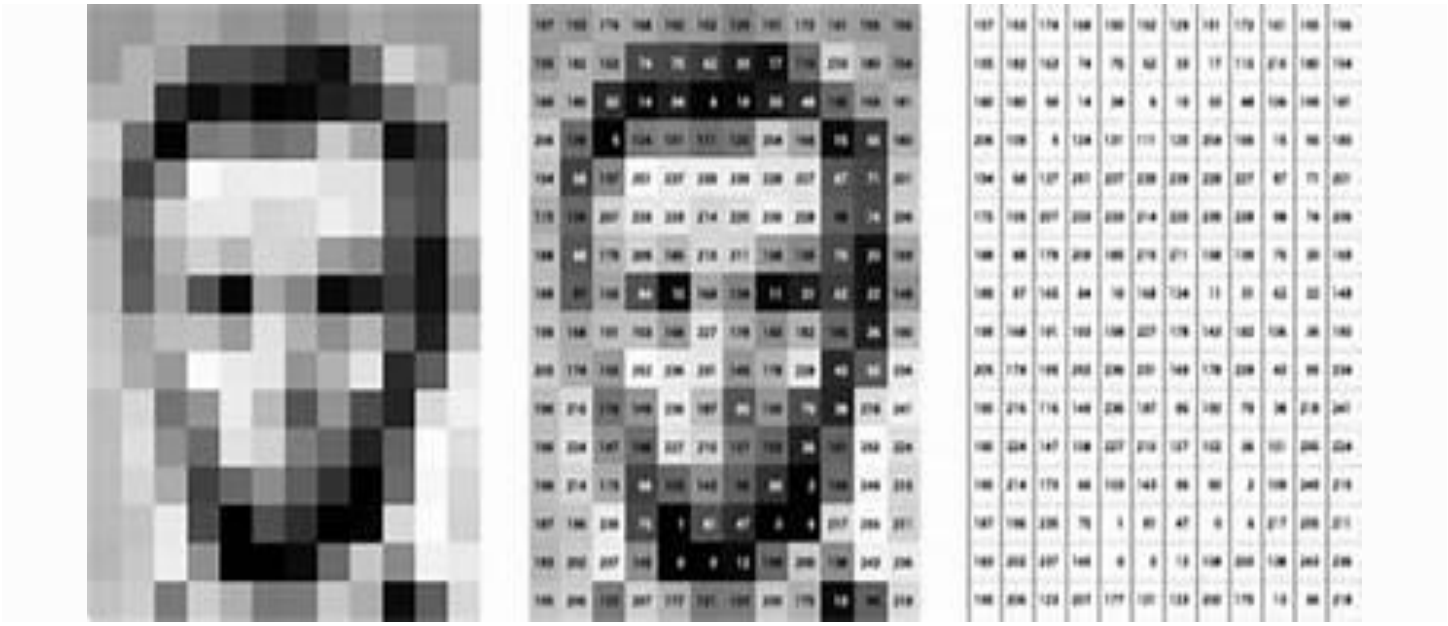


**Figure 42** **A computer sees an image as an array of numbers.**

### 6.3.1) Building blocks of CNN architecture

The CNN architecture consists of repetitions of a stack of several convolution layers and a pooling layer, followed by one or more fully connected layers.

#### 6.3.1.1) Convolution layer

A convolution layer is a fundamental component of the CNN architecture that performs feature extraction, which typically consists of a combination of linear and nonlinear operations, i.e., convolution operation and activation function.

#### 1. Convolution

Convolution is a specialized type of linear operation used for feature extraction, where a small array of numbers, called a kernel, is applied across the input, which is an array of numbers, called a tensor. An element-wise product between each element of the kernel and the input tensor is calculated at each location of the tensor and summed to obtain the output value in the corresponding position of the output tensor, called a feature map. This procedure is repeated applying multiple kernels to form an arbitrary number of feature maps, which represent different characteristics of the input tensors; different kernels can, thus, be considered as different feature extractors such as a horizontal edge detector, a vertical edge detector, and an outline detector.

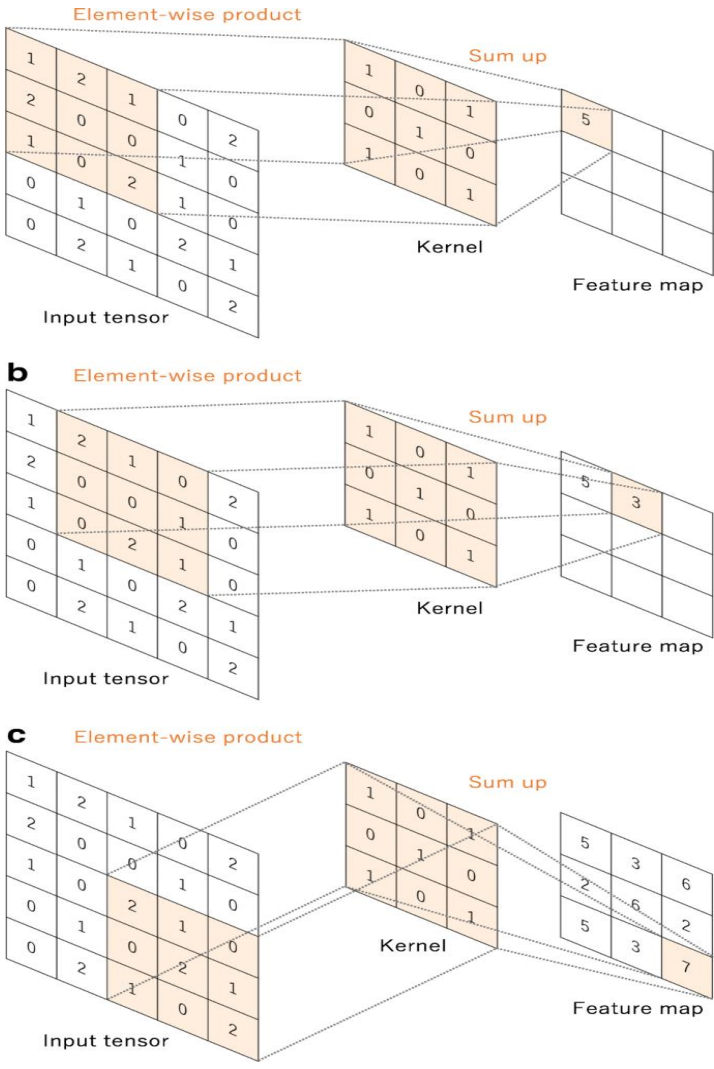



*Figure 43 convolution operation*

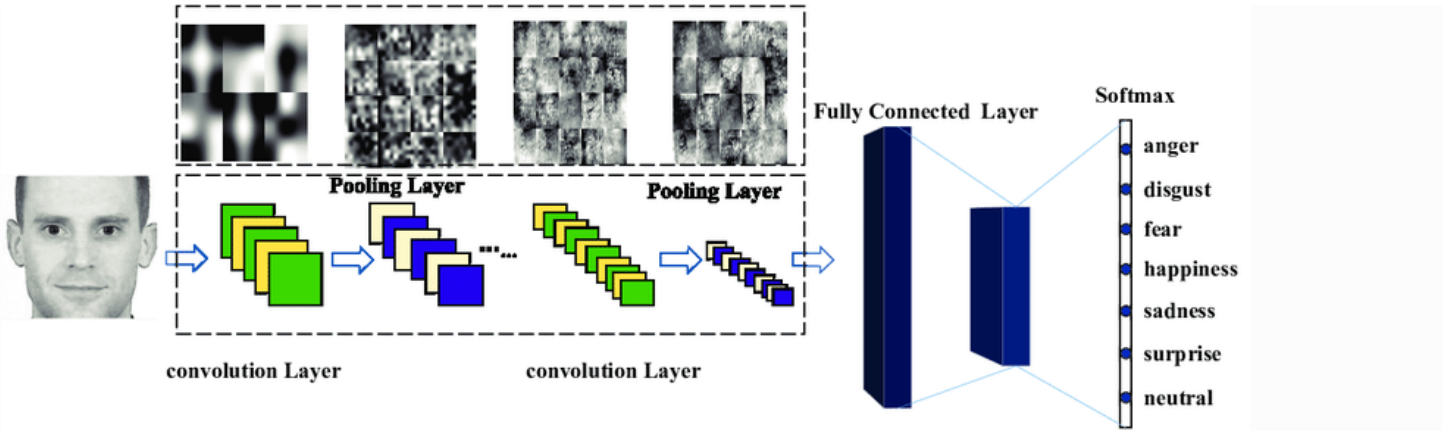



**Figure 44 Example of how kernels in convolution layers extract features from an input tensor.**

## 2. Nonlinear activation function

The outputs of a linear operation such as convolution are then passed through a nonlinear activation function. Although smooth nonlinear functions, such as ReLU( Rectified Linear Unit) , Softmax function

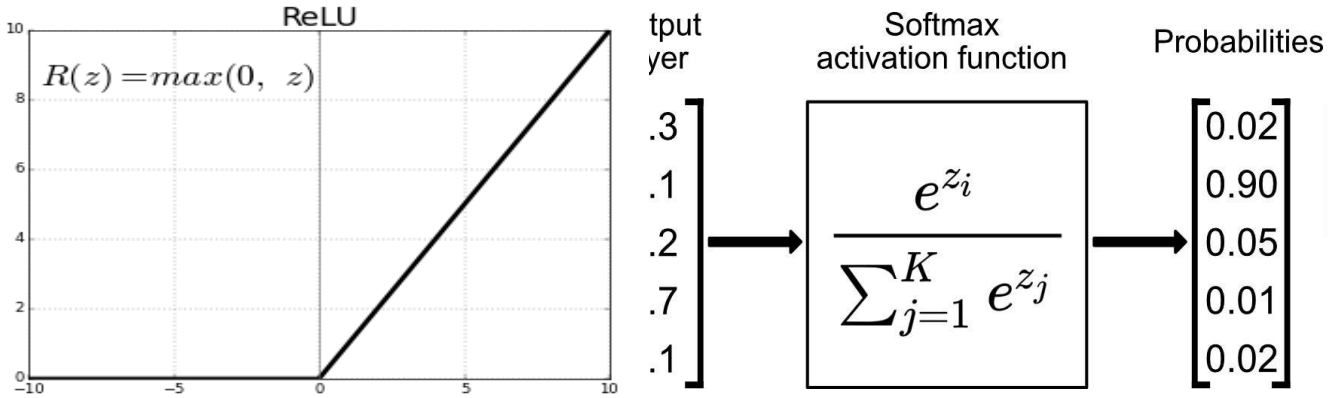



*Figure 45 Activation functions commonly applied to neural networks.*

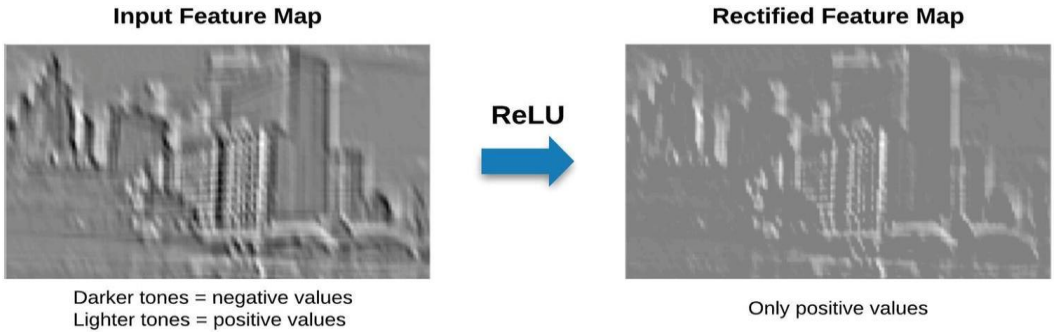



*Figure 7 ReLU Example*

### 6.3.1.2) Pooling layer

A pooling layer provides a typical **downsampling** operation which reduces the in-plane dimensionality of the feature maps in order to introduce a translation invariance to small shifts and distortions, and decrease the number of subsequent learnable parameters.

- **Max pooling**

The most popular form of pooling operation is max pooling, which extracts patches from the input feature maps, outputs the maximum value in each patch, and discards all the other values (**Fig. 8**). A max pooling with a filter of size 2 × 2 with a stride of 2 is commonly used in practice. This downsamples the in-plane dimension of feature maps by a factor of 2. Unlike height and width, the depth dimension of feature maps remains unchanged.

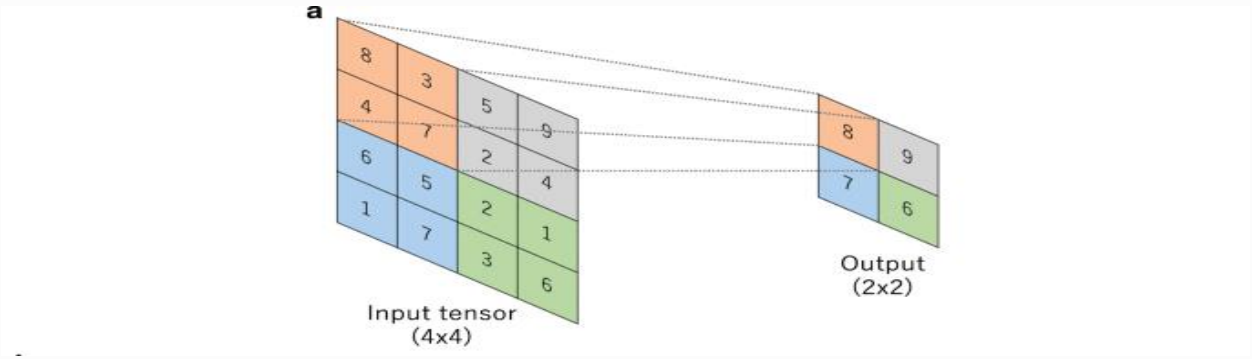



**Figure 46 The max pooling operation**

### 6.3.1.3) Fully connected layer

The output feature maps of the final convolution or pooling layer is typically flattened and transformed into a one-dimensional (1D) array of numbers and connected to one or more fully connected layers, also known as dense layers, in which every input is connected to every output by a learnable weight. Once the features extracted by the convolution layers and downsampled by the pooling layers are created, they are mapped by a subset of fully connected layers to the final outputs of the network, such as the probabilities for each class in classification tasks. The final fully connected layer typically has the same number of output nodes as the number of classes. Each fully connected layer is followed by a nonlinear function, such as ReLU, as described above.

- **Last layer activation function**

The activation function applied to the last fully connected layer is usually different from the others. An appropriate activation function needs to be selected according to each task. An activation function applied to the multiclass classification task is a softmax function which normalizes output real values

from the last fully connected layer to target class probabilities, where each value ranges between 0 and 1 and all values sum to 1.

## 6.4) Training a network

Training a network is a process of finding kernels in convolution layers and weights in fully connected layers which minimize differences between output predictions and given ground truth labels on a training dataset. Backpropagation algorithm is the method commonly used for training neural networks where loss function and gradient descent optimization algorithm play essential roles. A model performance under particular kernels and weights is calculated by a loss function through forward propagation on a training dataset, and learnable parameters, namely kernels and weights, are updated according to the loss value through an optimization algorithm called backpropagation and gradient descent, among others.

### 6.4.1) Loss function

A loss function, also referred to as a cost function, measures the compatibility between output predictions of the network through forward propagation and given ground truth labels. Commonly used loss function for multiclass classification is cross entropy, whereas mean squared error is typically applied to regression to continuous values. A type of loss function is one of the hyperparameters and needs to be determined according to the given tasks.

### 6.4.2) Gradient descent

Gradient descent is commonly used as an optimization algorithm that iteratively updates the learnable parameters, kernels and weights, of the network so as to minimize the loss. The gradient of the cost function provides us the direction in which the function has the steepest rate of increase, and each learnable parameter is updated in the negative direction of the gradient with an arbitrary step size determined based on a hyperparameter called learning rate. The gradient is formulated as follows:

$$\omega := \omega - \alpha * \frac{\partial J}{\partial \omega}$$

**Where**

**w** stands for each learnable parameter,

**α** stands for a learning rate,

**J** stands for a cost function.

It is of note that, in practice, a learning rate is one of the most important hyperparameters to be set before the training starts. In practice, for reasons such as memory limitations, the gradients of the loss function with regard to the parameters are computed by using a subset of the training dataset called mini-batch,

and applied to the parameter updates. In addition, many improvements on the gradient descent algorithm have been proposed and widely used, such as SGD with momentum, RMSprop, and Adam .

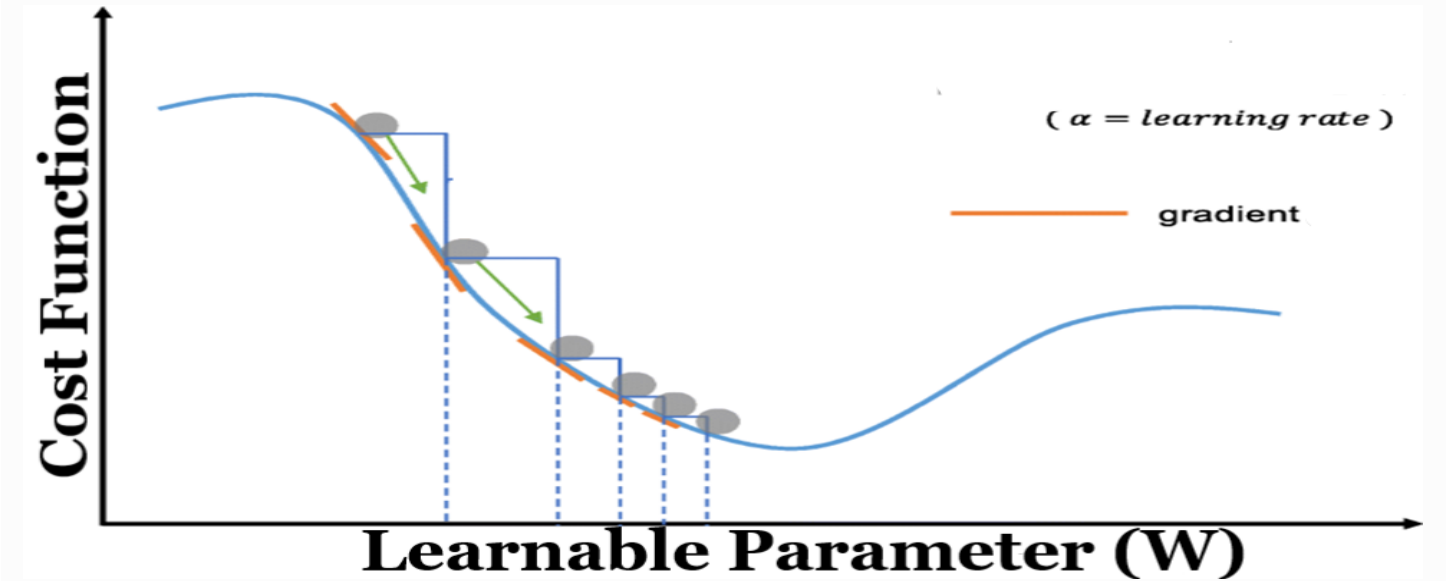



**Figure 47 Cost function**

## 6.5) Data Sets

Data is the most important components in applying deep learning or other machine learning methods. Careful collection of data with which to train and test a model is mandatory for a successful deep learning project, Available data are typically split into three sets: a training, a validation, and a test set ,though there are some variants, such as cross validation.

- A training set is used to train a network, where loss values are calculated via forward propagation and learnable parameters are updated via backpropagation.
- A validation set is used to evaluate the model during the training process, fine-tune hyperparameters, and perform model selection.
- A test set is ideally used only once at the very end of the project in order to evaluate the performance of the final model that was fine-tuned and selected on the training process with training and validation sets.

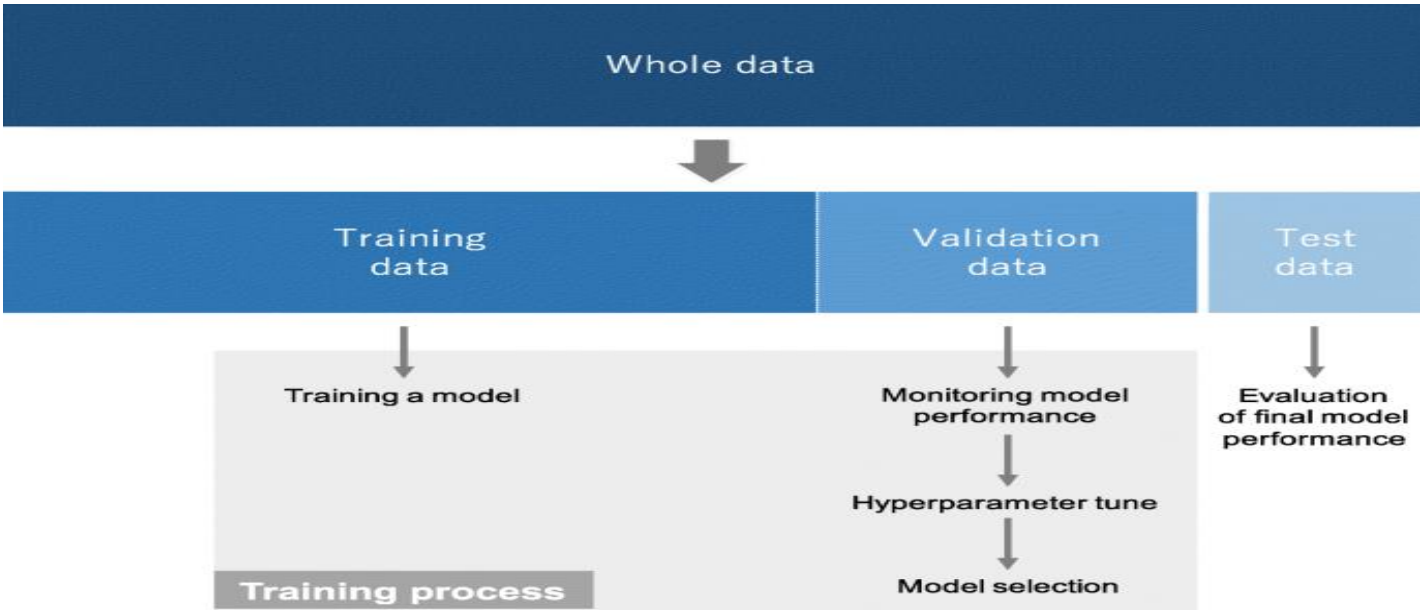



**Figure 48** Training set

### 6.5.1) Challenges We Face Through The Training Process

#### 6.5.1.1) Overfitting (High variance)

Overfitting refers to a situation where a model learns statistical regularities specific to the training set, i.e., ends up memorizing the irrelevant noise instead of learning the signal and, therefore, performs **less** well on a subsequent new dataset. This is one of the main challenges in machine learning, as an overfitted model is not generalizable to never-seen-before data. In that sense, a test set plays a pivotal role in the proper performance evaluation of machine learning models. A routine check for recognizing overfitting to the training data is to monitor the loss and accuracy on the training and validation sets . If the model performs well on the training set compared to the validation set, then the model has likely been overfit to the training data.

**Solution**

There have been several methods proposed to minimize overfitting.

- **First Method**
  The solution for reducing overfitting is to obtain more training data. A model trained on a larger dataset typically generalizes better.
- **Second Method**
  The other solutions include regularization with dropout , batch normalization and data augmentation, as well as reducing architectural complexity.

- **Dropout** is a recently introduced regularization technique where randomly selected activations are set to 0 during the training, so that the model becomes less sensitive to specific weights in the network .
- **Batch normalization** is a type of supplemental layer which adaptively normalizes the input values of the following layer, mitigating the risk of overfitting, as well as improving gradient flow through the network, allowing higher learning rates, and reducing the dependence on initialization.
- **Data augmentation** is also effective for the reduction of overfitting, which is a process of modifying the training data through random transformations, such as flipping, translation, cropping, rotating, and random erasing, so that the model will not see exactly the same inputs during the training iterations .

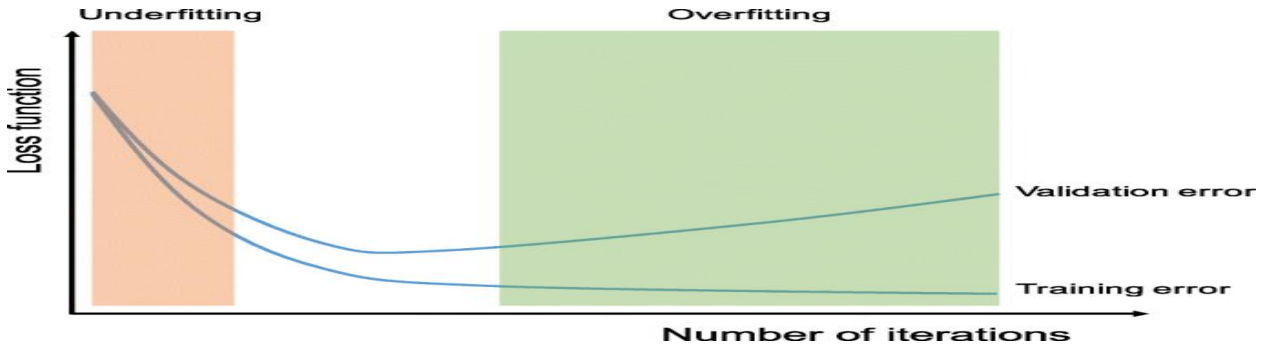



**Figure 49** Overfitting

| How to mitigate overfitting |
|---|
| More training data |
| Data augmentation |
| Regularization (weight decay, dropout) |
| Batch normalization |
| Reduce architecture complexity |

**Figure 50** A list of common methods to mitigate overfitting

### 6.5.1.2) Training on a small dataset

There are a couple of techniques available to train a model efficiently on a smaller dataset:

1. data augmentation
2. transfer learning.

As data augmentation was briefly covered in the previous section, this section focuses on transfer learning.

## Transfer learning

Transfer learning is a common and effective strategy to train a network on a small dataset, where a network is pretrained on an extremely large dataset, such as ImageNet, which contains 1.4 million images with 1000 classes, then reused and applied to the given task of interest. The underlying assumption of transfer learning is that generic features learned on a large enough dataset can be shared among seemingly disparate datasets. This portability of learned generic features is a unique advantage of deep learning that makes itself useful in various domain tasks with small datasets. At present, many models pretrained on the ImageNet challenge dataset are open to the public and readily accessible, along with their learned kernels and weights, such as AlexNet , VGG , ResNet, InceptionV3 , and DenseNet . In practice, there are two ways to utilize a pretrained network: fixed feature extraction and fine-tuning.

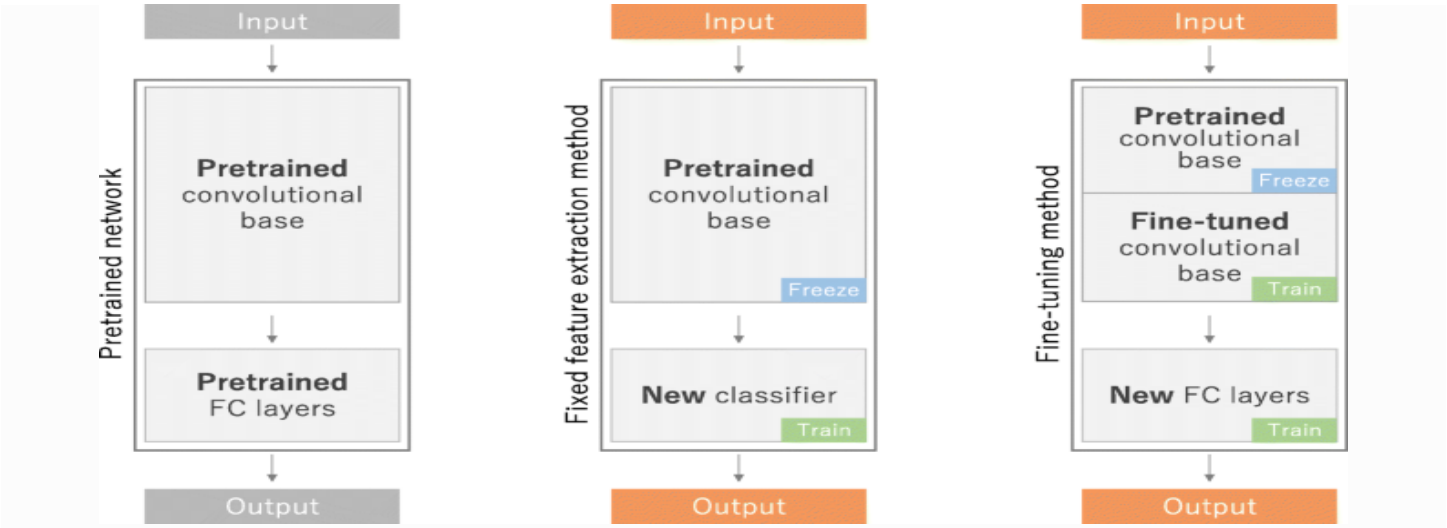



**Figure 51 utilization of a pretrained network**

- A fixed feature extraction method is a process to remove FC layers from a pretrained network and while maintaining the remaining network, which consists of a series of convolution and pooling layers, referred to as the convolutional base, as a fixed feature extractor.

- A fine-tuning method, which is to not only replace fully connected layers of the pretrained model with a new set of fully connected layers to retrain on a given dataset, but to fine-tune all or part of the kernels in the pretrained convolutional base by means of backpropagation. All the layers in the convolutional base can be fine-tuned or, alternatively, some earlier layers can be fixed while fine-tuning the rest of the deeper layers. This is motivated by the observation that the early-layer features appear more generic, including features such as edges applicable

to a variety of datasets and tasks, whereas later features progressively become more specific to a particular dataset or task .

## 6.6) Object detection

### 6.6.1) Introduction

Object detection is a computer technology that identifies objects in images and videos. It involves detecting or locating a moving object in a frame and classifying the detected objects into different categories. Object tracking, on the other hand, locates moving objects over time in video sequences. It relies on object detection and classification as preceding steps. Tracking objects in consecutive frames can be challenging due to various factors such as complex object motion, irregular shapes, occlusion, and real-time processing requirements.

Object recognition methods use extracted features and learning algorithms to recognize instances of an object or images belonging to an object category. Object class recognition involves classifying objects into specific classes or categories, while object detection aims to localize a specific object of interest in digital images or videos. Each object or object class has unique features that characterize them and differentiate them from others, aiding in the recognition of similar objects in other images or videos. Classification involves determining which category is present in an image patch, while localization and detection involve determining if a specific object of interest is present in a complex image and providing accurate location information about the object.

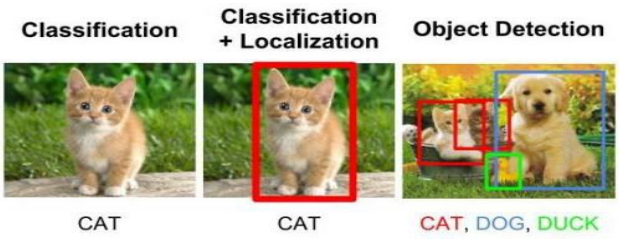



*Figure 52 object detection*

In an image recognition scenario, a model would simply identify that there is a dog in the image. However, in an object detection scenario, the model would not only identify that there is a dog but also draw a bounding box around the dog to indicate its location in the image. This additional information provided by object detection allows for more detailed analysis and understanding of the objects present in an image or video.

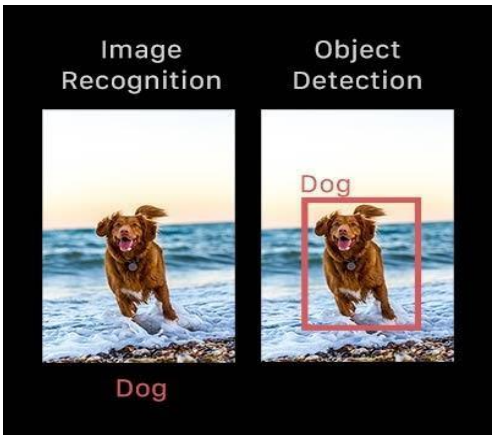



**Figure 53 Image Recognition VS. object detection**

### 6.6.2) Face and Eye detection

Face detection can be regarded as a specific case of object-class detection. In object-class detection, the task is to find the locations and sizes of all objects in an image that belong to a given class.
Face detection simply answers two questions:

1. Is there any face in the collected images or video?
2. Where was it located?

- Face-detection algorithms focus on the detection of frontal human faces. It is analogous to image detection in which the image of a person is matched bit by bit. Image matches with the image stored in the database. Any facial feature changes in the database will invalidate the matching process.
- Eye detection has become an important research topic in computer vision and pattern recognition, because the human eye's locations are essential information for many applications, including psychological analysis, facial expression recognition, auxiliary driving, and medical diagnosis. However, eye detection is quite challenging in many practical applications. The cameras are sensitive to light variations and the shooting distance, which makes the human eyes very eccentric in a facial image.

## 6.7) The Inception V3

### 6.7.1) Introduction

The Inception V3 is a deep learning model based on Convolutional Neural Networks, which is used for image classification. The inception V3 is just the advanced and optimized version of the inception V1 model

The Inception V3 model used several techniques for optimizing the network for better model adaptation.

- It has higher efficiency
- It has a deeper network compared to the Inception V1 and V2 models, but its speed isn't compromised.
- It is computationally less expensive.
- It uses auxiliary Classifiers as regularizes.

### 6.7.2) Inception V3 Model Architecture

The inception v3 model was released in the year 2015, it has a total of 42 layers and a lower error rate than its predecessors. Let's look at what are the different optimizations that make the inception V3 model better.

The major modifications done on the Inception V3 model are

1. Factorization into Smaller Convolutions
2. Spatial Factorization into Asymmetric Convolutions
3. Utility of Auxiliary Classifiers
4. Efficient Grid Size Reduction

Let's how each one of these optimizations was implemented and how it improved the model.

#### 6.7.2.1) Factorization into Smaller Convolutions

One of the major assets of the Inception V1 model was the generous dimension reduction. To make it even better, the larger Convolutions in the model were factorized into smaller Convolutions.

For example, consider the basic module of the inception V1 module.

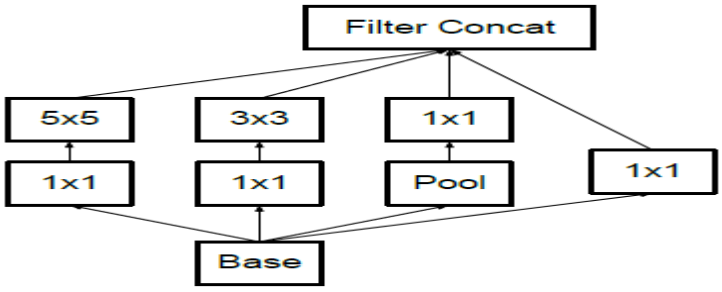



Figure 54

It has a 5×5 convolutional layer which was computationally expensive as said before. So to reduce the computational cost the 5×5 convolutional layer was replaced by two 3×3 convolutional layers as shown below.

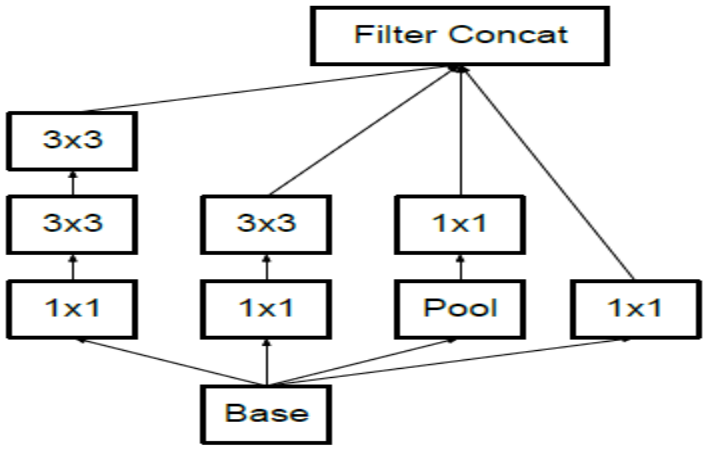



Figure 55

As a result of the reduced number of parameters the computational costs also reduce. This factorization of larger convolutions into smaller convolutions resulted in a relative gain of 28%.

#### 6.7.2.2) Spatial Factorization into Asymmetric Convolutions

Even though the larger convolutions are factorized into smaller convolutions. But, a better alternative to make the model more efficient was Asymmetric convolutions.

Asymmetric convolutions are of the form n×1.
So, what they did is replace the 3×3 convolutions with a 1×3 convolution followed by a 3×1 convolution. Doing so is the same as sliding a two-layer network with the same receptive field as in a 3×3 convolution.

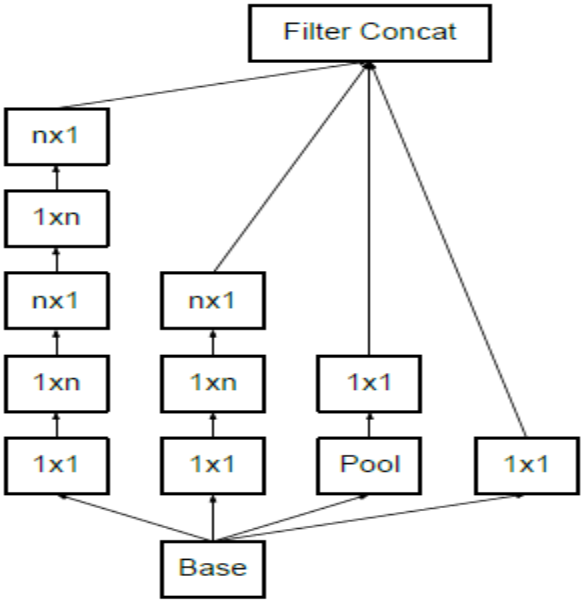



**Figure 56 Module 2**

**Structure of Asymmetric Convolutions**

The two-layer solution is 33% cheaper for the same number of output filters if the number of input and output filters is equal.

After applying the first two optimization techniques the inception module looks like this.

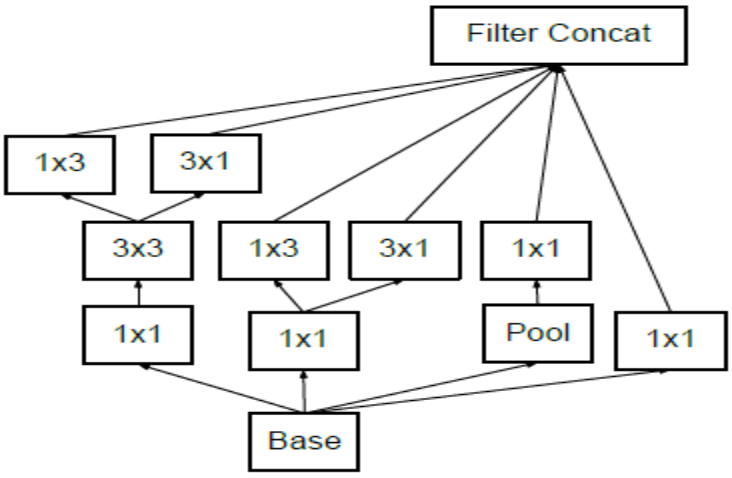



**Figure 57 Module 3**

#### 6.7.2.3) Utility of Auxiliary classifiers

The objective of using an Auxiliary classifier is to improve the convergence of very deep neural networks. The auxiliary classifier is mainly used to combat the vanishing gradient problem in very deep networks.

The auxiliary classifiers didn't result in any improvement in the early stages of the training. But towards the end, the network with auxiliary classifiers showed higher accuracy compared to the network without auxiliary classifiers.

Thus the auxiliary classifiers act as a regularizer in Inception V3 model architecture.

#### 6.7.2.4) Efficient Grid Size Reduction

Traditionally max pooling and average pooling were used to reduce the grid size of the feature maps. In the inception V3 model, in order to reduce the grid size efficiently the activation dimension of the network filters is expanded.

For example, if we have a d×d grid with k filters after reduction it results in a d/2 × d/2 grid with 2k filters.

And this is done using two parallel blocks of convolution and pooling later concatenated.

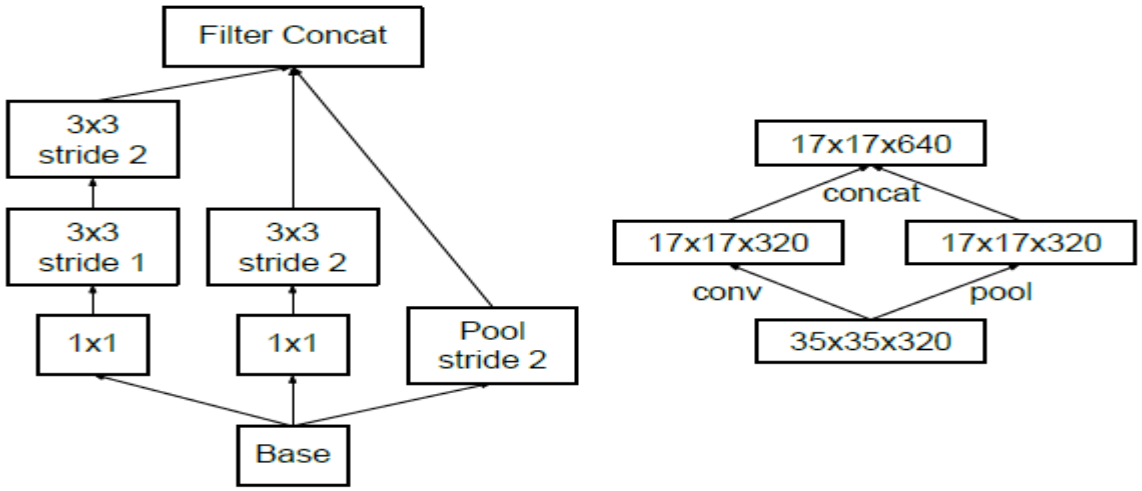



**Figure 58 Grid size**

The above image shows how the grid size is reduced efficiently while expanding the filter banks.

### 6.7.2.5) The final Inception V3 model

After performing all the optimizations the final Inception V3 model looks like this

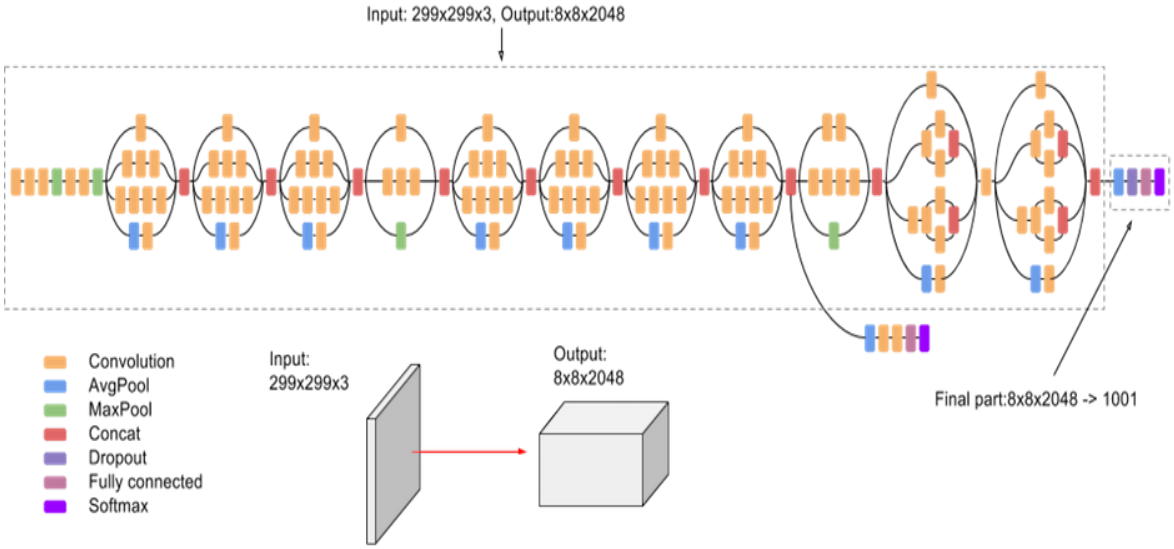



**Figure 59 The structure of Inception V3 model Performance of Inception V3**

As expected the inception V3 had better accuracy and less computational cost compared to the previous Inception version.

| Network | Models Evaluated | Crops Evaluated | Top-1 Error | Top-5 Error |
|---|---|---|---|---|
| VGGNet [18] | 2 | - | 23.7% | 6.8% |
| GoogLeNet [20] | 7 | 144 | - | 6.67% |
| PReLU [6] | - | - | - | 4.94% |
| BN-Inception [7] | 6 | 144 | 20.1% | 4.9% |
| Inception-v3 | 4 | 144 | **17.2%** | **3.58%*** |

**Figure 60 Multi-crop reported results.**

# CHAPTER 7
# Applications

## 7.1) Drowsiness Detection System

### 7.1.1) Introduction

Drowsiness is one of the significant reasons for road crashes that results in considerable damaging consequences to the individuals who suffer fatal or non-fatal injuries, property damage and economic losses to the nation.

- According to a survey conducted by the National Highway Traffic Safety Administration (NHTSA) estimated that 8.8% to 9.5% crashes resulted from road accidents in 2018 in the United States.
- Another study made by the American Automobile Association (AAA) Foundation for Traffic Safety identified that around 328,000 drowsy driving crashes occur yearly.
- According to National sleep foundation and experimental values obtained, it was confirmed that the factors that contribute to drowsy driving include:
  - participants with less than 7 hours of sleep, participants with sleep disorders, driving at late hours, and frequent traveling through different time zones (commercial drivers), and working late night shifts or long shift hours, influence of medication, stress and sedentary lifestyles.
  - It was observed that drunk driving also led to drowsy driving since alcohol affects the brain cells and causes sleepiness depending on blood alcohol concentration. It took around 30 minutes for alcohol to start affecting a person through tested results. It was also confirmed that alcohol enters the bloodstream within about 20 minutes before it starts to affect the person. External factors observed that affect drowsiness level include environmental conditions like temperature and humidity inside the vehicle .

There are many symptoms that can help detect sleepiness or distraction of the driver.

Common symptoms that have been identified during drowsy driving include:

1. constant nodding
2. difficulty opening eyes
3. missing road signs and turns
4. frequent lane drifting and difficulty in maintaining speed.

The main symptom is the eyes of the driver. Eye detection has been engaged in development of artificial intelligence (AI) in the form of eye tracking technology that understands, supports and predicts a person's intentions and actions. By carefully studying eye, facial and head movement, our technology can draw conclusions about a person's awareness and mental state. Using Machine Learning & Computer Vision in our project. Image processing is a method to perform some operations on an image, to get an enhanced image or to extract some useful information from it. So our project aims to detect the driver's eyes and if they are

closed The system will alert the driver and if the driver's attention is still distracted the car will park and send to the server asking for help.

### 7.1.2) Techniques for Driver Drowsiness Detection

There are different techniques used to detect measure and predict driver drowsiness can be splits into two approaches, intrusive and non-intrusive, to identify drowsy driving. In an intrusive approach, the drowsy condition of a person is identified using physiological parameters, but in a non-intrusive approach, this is identified by installing devices and sensors on the vehicle. Based on intrusive and non-intrusive approaches, five different measures are utilized to identify driver drowsiness at an early stage. These measures are as follows:

1. Subjective measures (SM)
2. Vehicle-based measures (VBM)
3. Physiological measures (PM)
4. Behavioral measures (BM)
5. Hybrid measures (HM)

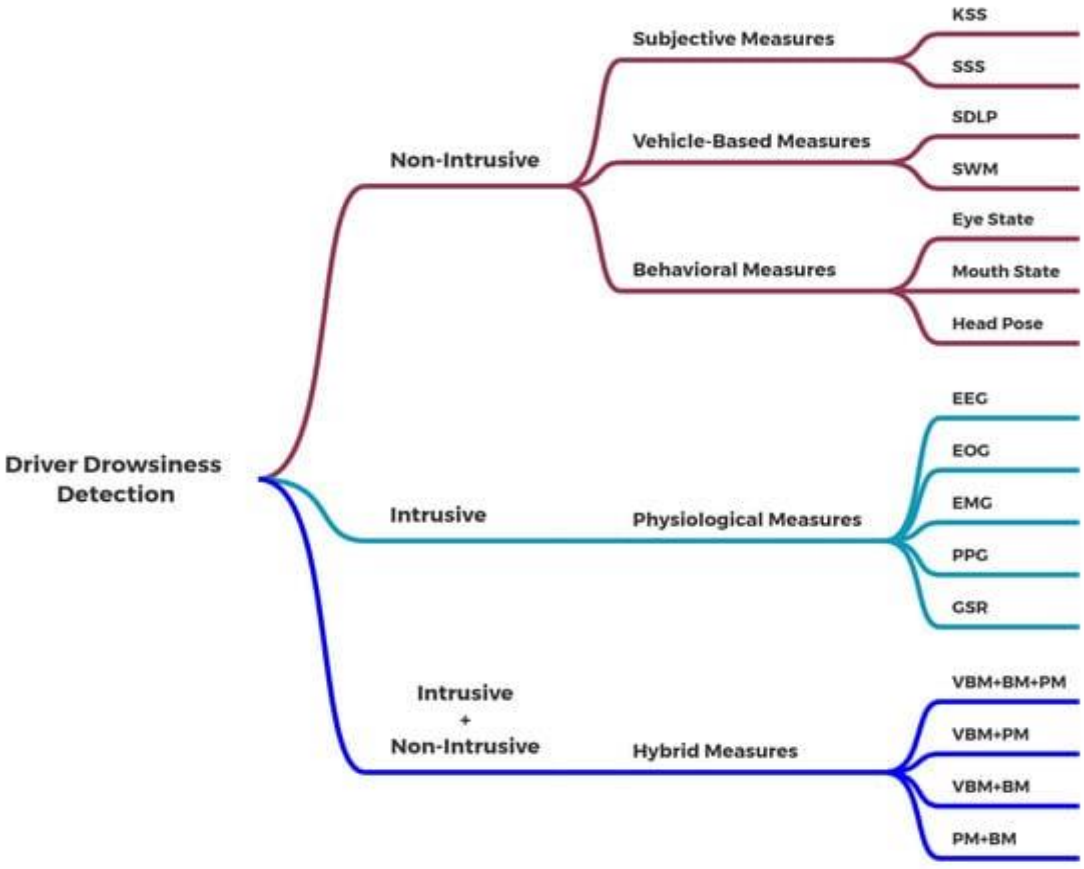



**Figure 61 Techniques for Driver Drowsiness Detection**

- **Behavioral measures technique**

Behavioral measures are based on the driver's features, such as eyes. To identify drowsy driving, the researcher primarily focuses on eye blink rate and percentage of eye closure, which are further examined by machine learning (ML) and deep learning (DL) algorithms. These behavioral measurement techniques are frequently used in simulated and real driving conditions. The present state of the art reveals that behavioral measures are more accurate than vehicle-based measures .

Due to its non-intrusiveness, the behavioral measure is one of the most widely used drowsiness detection techniques. A camera is mounted on the dashboard of the vehicle to capture the driver's facial features.

depicts the procedure to determine the drowsy state of the driver using behavioral measures. Three phases comprise the entire process: data acquisition, feature extraction and classification. The first step in data acquisition is gathering the driver's image or video. Thereafter, in the second phase, the face is detected by applying pre-processing techniques. Through the feature extraction phase, the region of interest (ROI) is identified. The ROI includes capturing eye using HaarCascade -based algorithms then feed it to the classifier (Model).

. In the classification phase, the binary classification method is used to evaluate the drowsy or non-drowsy state of the driver.

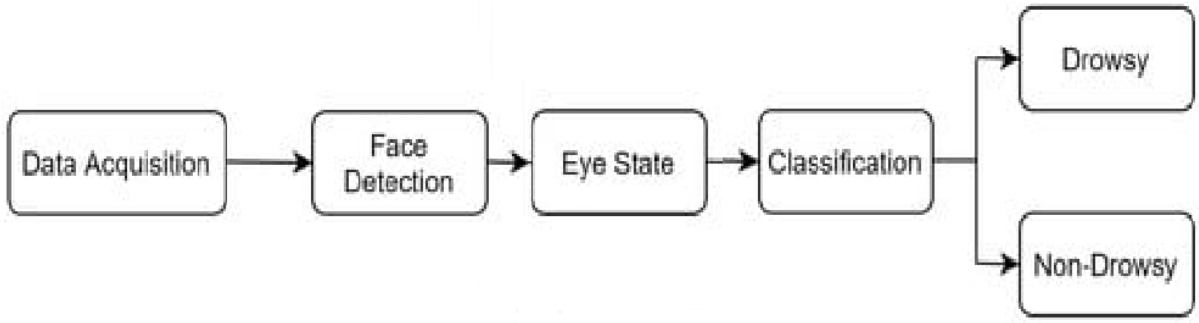



**Figure 62 Detection of driver drowsiness using behavioral measures.**

### 7.1.3) Configuring development environment

The following packages installed:

- OpenCV
- NumPy
- matplotlib
- TensorFlow

- Keras
- OS

## 7.1.4) Our Dataset:

### 7.1.4.1) The MRL Eyes dataset

MRL Eyes dataset is the large-scale dataset of human eye images. This dataset contains infrared images in low and high resolution, all captured in various lightning conditions and by different devices. The dataset is suitable for testing several features or trainable classifiers.

We select some images from this dataset with (open/Closed) eyes And (with/without) glasses.

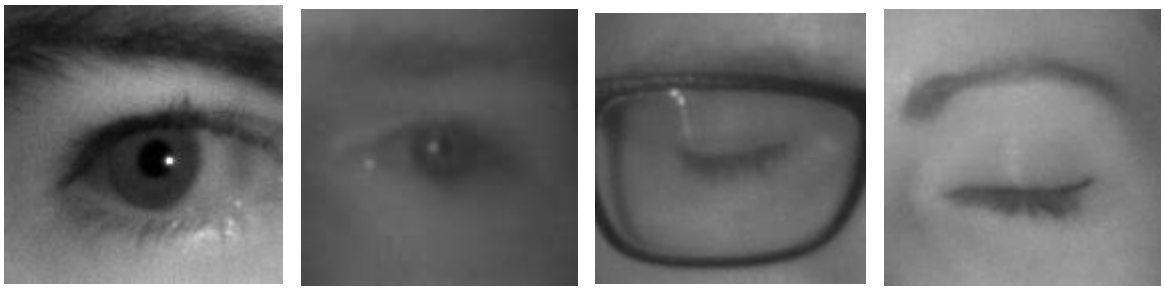


**Figure 63 The MRL Eyes dataset**

### 7.1.4.2) Kaggle dataset

***Figure 64 Kaggle dataset***

Convert it to gray scale.

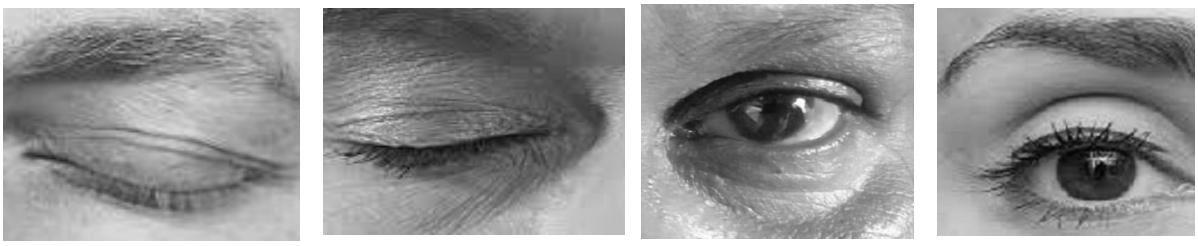


***Figure 65 kaggle dataset (gray scale)***

Then we mix MRL dataset and grayed Kaggle dataset and select about 6500 images randomly.

You can download our final dataset from the following link :
https://drive.google.com/drive/folders/1uTTrBlG_2Gwy_o0FCNn4e5ac6YWF3ddp?usp=sharing

### 7.1.5) The Model

#### 7.1.5.1) Data Augmentation

We did some Data Augmentations on our dataset (only training set) like normalization, rotation, shearing, zooming and flipping.

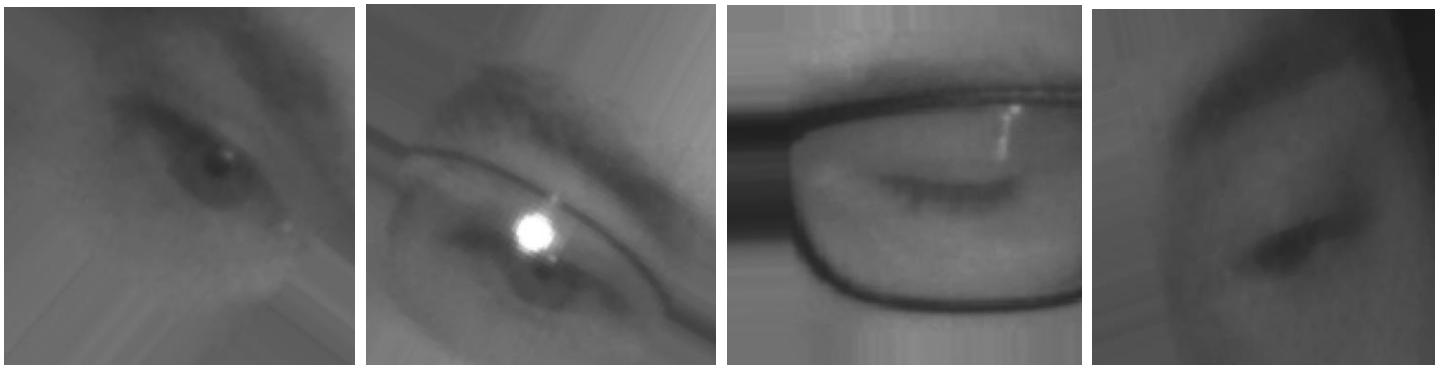


**Figure 66 Data Augmentation**

#### 7.1.5.2) Transfer Learning

We tried many pretrained models like Inception_V3 , Resnet 50 and MobileNet_V1,Then we decided to work with Inception_V3 as it has the best accuracy for our model among the other pretrained models.

We used the weights of “Imagenet”, removed the fully connected layer and froze the weights of the inception model as it will be untrainable.

```
pre_trained_model_inception = tf.keras.applications.InceptionV3(input_shape = (150, 150, 3),
                                include_top = False,
                                weights = 'imagenet')

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

We chose “mixed7’ to be our last layer as the image size will be (7,7,768) and after that it will have size smaller than that size so we chose this layer as it will have more features and can deal with it.

```
activation_68 (Activation)     (None, 7, 7, 192)    0           ['batch_normalization_68[0][0]']

activation_69 (Activation)     (None, 7, 7, 192)    0           ['batch_normalization_69[0][0]']

mixed7 (Concatenate)           (None, 7, 7, 768)    0           ['activation_60[0][0]',
                                                                 'activation_63[0][0]',
                                                                 'activation_68[0][0]',
                                                                 'activation_69[0][0]']
```

### 7.1.5.3) Fully connected layer

```
x=Flatten()(pre_trained_model_inception.output)
x=Dense(128,activation='relu')(x)
x=Dropout(0.4)(x) #reduce overfitting
x=Dense(2,activation='softmax')(x)
```

### 7.1.5.4) Loss Function and Optimizer

We chose categorical cross entropy and ADAM optimizer.

```
Model_.compile(loss='categorical_crossentropy',optimizer='Adam',metrics = ['accuracy'])
```

### 7.1.5.5) Training The Model

We iterate about 50 epochs.

```
History=Model_.fit(train_gen,
                   validation_data=valid_gen,
                   epochs=50,
                   verbose=1,
                   callbacks=callbacks)
```

### 7.1.5.6) The Model Accuracy

The accuracy of training set is about 99% and validation set is 95% so it is acceptable model.

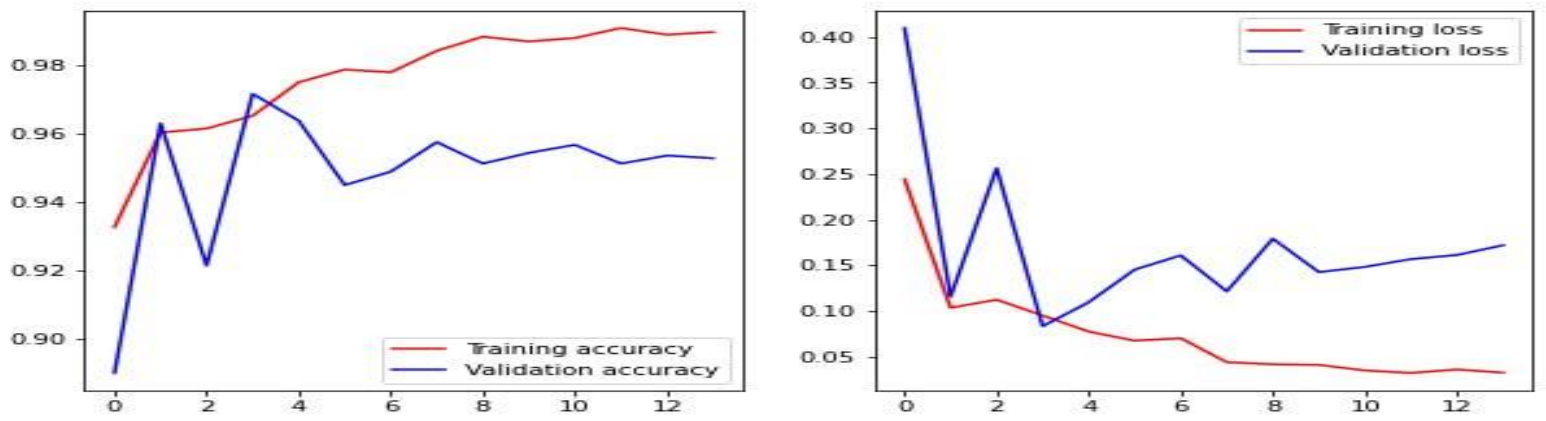



**Figure 67 accuracy of drowsiness detection model**

Feel free to check our Colab Notebook : https://colab.research.google.com/drive/1y1qB9mRI5e9zrCKp9bW-GrrKkesvWwWN?usp=sharing

### 7.1.6) Launching The Model

### 7.1.6.1) Haar-Cascade algorithm

-We used Haar-cascade algorithm which is embedded in Open-CV library to detect the eyes from the incoming frame.

-Haar-cascade is an algorithm that can detect objects in images, irrespective of their scale in image and location.

-This algorithm is not so complex and can run in real-time. We can train a haar-cascade detector to detect various objects like cars, bikes, buildings, fruits, etc.

-Haar cascade uses the cascading window, and it tries to compute features in every window and classify whether it could be an object.

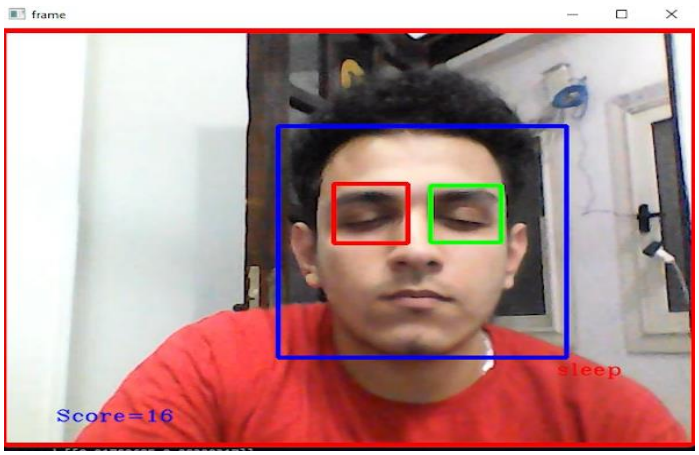
### 7.1.6.2) Steps of the launching

1) Take image as input from a camera in front of the driver.
2) Detect the eyes using Haar-casscade algorithm.
3) Feed the eye frame to the model.
4) Depending on the predictions we will decide if the driver is sleep or not.
5) Using the GPIO pins of the Raspberry pi we will take the decision (if the driver is sleep we will have

3.3 V ,else it will be zero volt).

6) Take that output and make the action on the car (there is a buzzer and the car will reduce its speed then it will stop).

### 7.1.6.3) Real Examples




Sleep

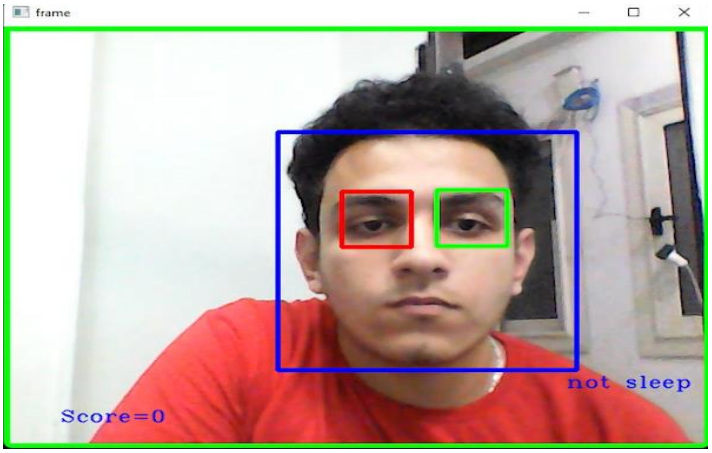



**Figure 68** Real Example

## 7.2) Digit Detection System.

### 7.2.1) Introduction

In our second model, we aim to match the digit on a vehicle the digit used for reservation. This helps ensure the right vehicle is allocated to the right customer.

We'll use image recognition techniques to analyze the license plate or other identifying features, focusing on digits 0-9.

To start, we need a dataset of labeled vehicle images with corresponding reservation digits. This data can be collected by photographing vehicle and recording the reservation digit at image capture.

Next, we'll use computer vision algorithms to preprocess the images, extracting the license plate area and isolating the digits of interest.

Techniques like image segmentation, edge detection, and character recognition can help with this.

Once the digits are extracted, we'll compare them to the reservation digit using convolutional neural networks (CNNs). After training the model, we can deploy it in real-time to classify new vehicle images and determine if the digit matches the reservation. The model can be integrated into our reservation system for automated allocation.

Regular monitoring and evaluation of the model's performance is crucial to ensure accuracy. We can update the model with new data, improving its performance over time. This second model will effectively detect if the digit on a vehicle matches the reservation digit, ensuring correct allocation to each customer.

### 7.2.1) Technique for Digit Detection

To detect if the digit of the vehicle number is the same as the digit used for the reservation, you can follow these steps:

1. Retrieve the vehicle number from the second model's input.
2. Extract the last digit from the vehicle number.
3. Retrieve the digit used for the reservation.
4. Compare the extracted digit from the vehicle number with the digit used for the reservation.
5. If they are the same, return a positive result indicating that the digit matches.
6. If they are different, return a negative result indicating that the digit does not match.

### 7.2.2) Our Datasets

#### 7.2.3.1) MNIST Dataset

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples.

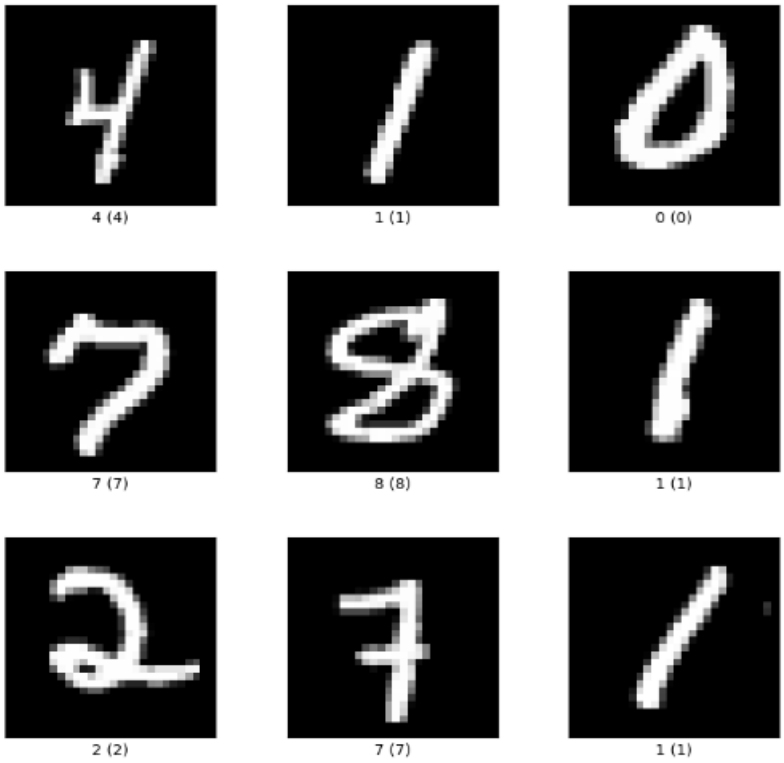



**Figure 69 MNIST Dataset**

### 7.2.3) The Model

#### 7.2.3.1) Data Augmentation

We did not make any data augmentation because we have a lot of images which can be trained well and the augmentation may hurt the digits images for example if we flip image of ‘6’ it may be ‘9’ so we will have wrong dataset which will affect the model.

#### 7.2.3.2) Transfer Learning

As our input image of MNIST has size (28,28) we can not feed these images to any pretrained model for example inception_v3 model requires images with minimum size = (75,75) and if we try to resize the image to bigger one the image will be destroyed, So we add convolution and pooling layer before fully connected layer.

### 7.2.3.3) Conv 2D, Max Pooling and Fully connected layer

```
model = Sequential()
model.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(28,28,1),padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
model.add(Flatten())
model.add(Dense(128,activation=tf.nn.relu))
model.add(Dense(128,activation=tf.nn.relu))
model.add(Dropout(0.3))
model.add(Dense(10,activation=tf.nn.softmax)) # softmax for probability distribution
model.compile(loss='categorical_crossentropy',optimizer='Adam',metrics = ['accuracy'])
```

### 7.2.3.4) Loss Function and Optimizer

We chose categorical cross entropy and ADAM optimizer.

```
Model_.compile(loss='categorical_crossentropy',optimizer='Adam',metrics = ['accuracy'])
```

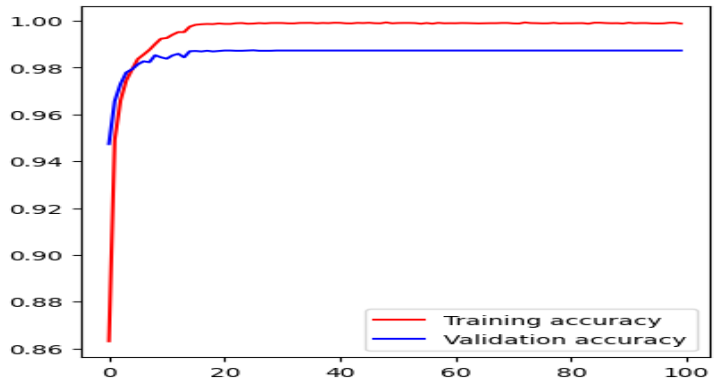
### 7.2.3.5) Training The Model

We iterate about 100 epochs.

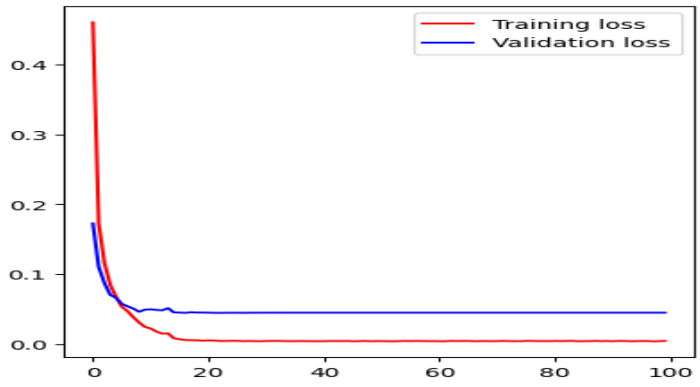
```
History=model.fit(x=x_train,
                  y=y_train_hot,
                  batch_size=512,
                  validation_data=(x_test,y_test_hot),
                  epochs=100,
                  verbose=1,
                  callbacks=callbacks
                  )
```

### 7.2.3.6) The Model Accuracy

The accuracy of training set is about 99% and validation set is 98% so it is acceptable model




Epochs




Epochs

Feel free to check our Colab Notebook :
https://colab.research.google.com/drive/15xtZggk9dzfYHfZQZADih0V56uWzM2jA?usp=sharing

### 7.2.4) Launching The Model

#### 7.2.4.1) Drawing Rectangular Box

As we have a fixed camera in front of the garage entrance so we draw a rectangular box to make the ID reservation label always inside that box, so the reservation label is the only thing that can be captured and fed to the model to make the recognition process.

#### 7.2.4.2) Steps of the launching

1) Take image as input from a camera in front of the car.
2) Take the frame of the rectangular box only.
3) Feed the digit frame to the model.
4) Depending on the probability of each digit we will take the max probability.
5) Each digit will be connected to a GPIO pin and the detected digit will have 3.3 V on its pin and zero Volt on the other pins.
6) Take that output from GPIO pins and make the action in the garage (the servo motor will open the door of the garage if the label on the car is equal to the reservation ID).

#### 7.2.4.3) Real Examples

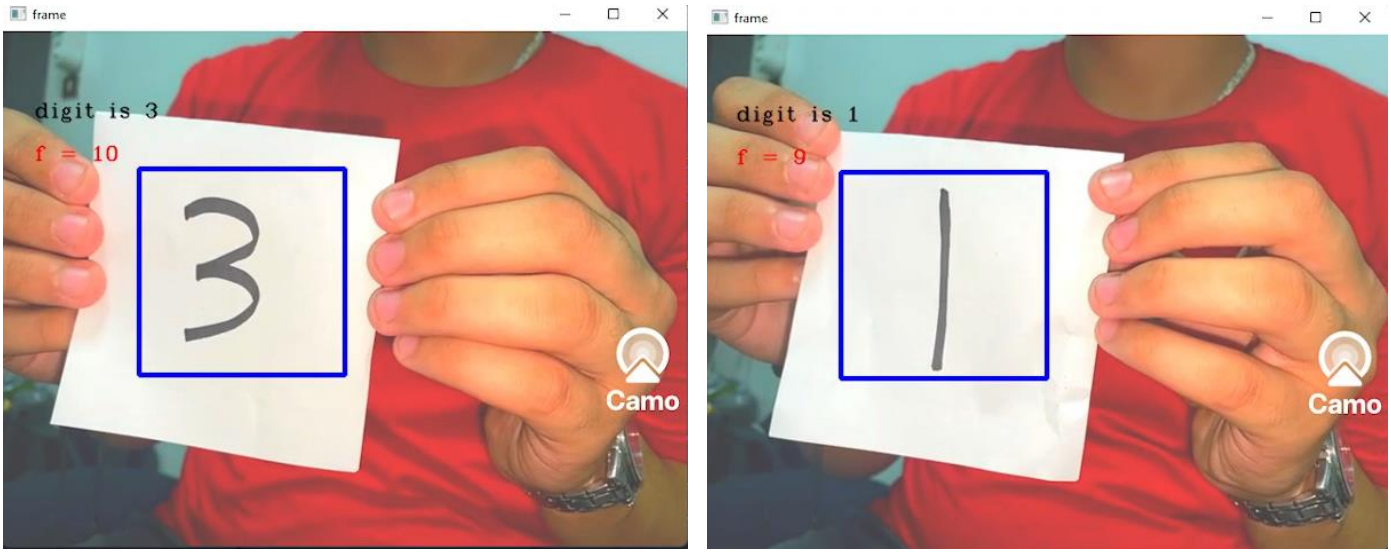



Figure 70 **Real Example**

# CHAPTER 8
# Vehicle To Garage Communication

## 8.1) Introduction:

This part is concerned with the communication between the vehicle unit and Garage unit in which when the user in the vehicle decides to park in the garage, they can reserve a parking slot before reaching the garage in order to guarantee an empty slot when reaching their destination. This is done using two Raspberry Pis, one existing in the Garage and the other existing in the vehicle. The two Raspberry Pis are being the clients(things) in an IOT network. The IOT network is established via two methods:

- AWS cloud (AWS IOT core service).
- LAN network (Edge)

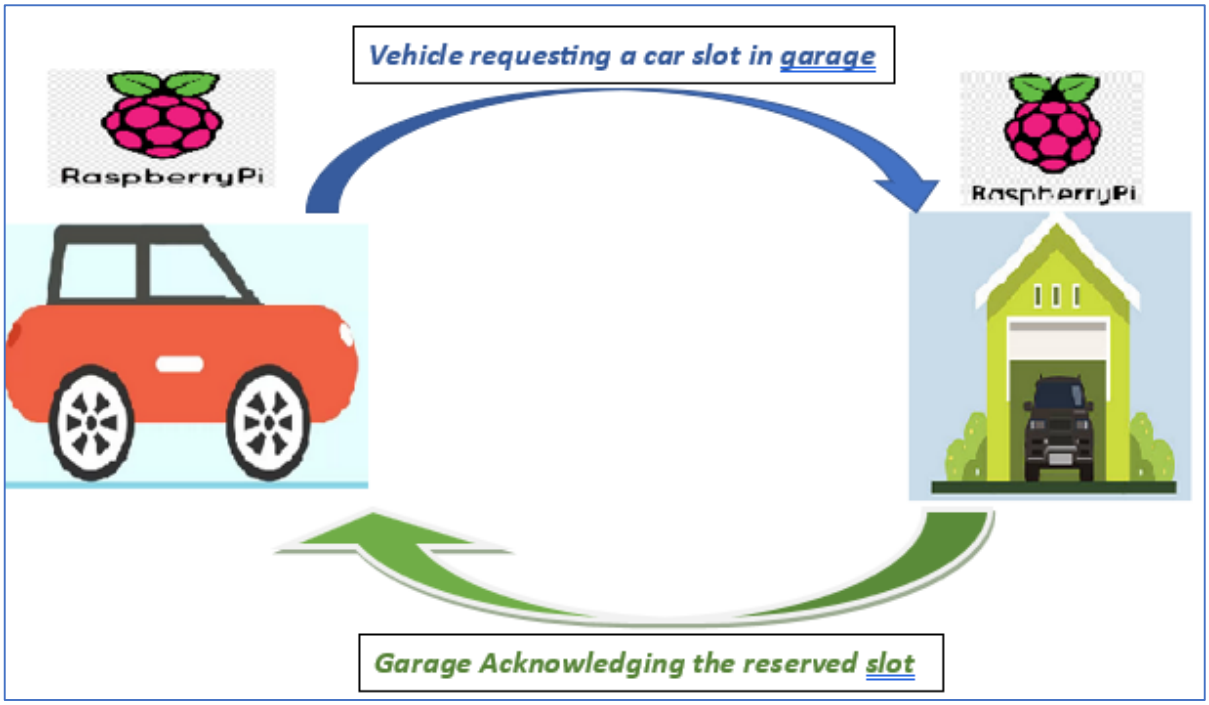



*Figure 71 Vehicle to Garage Comm design*

## 8.2) User Interface:

- The user is going to interact with the Raspberry pi existing in the vehicle via a mobile application installed on their cell phone.
- The communication between the mobile application and the Raspberry pi is achieved by connecting the Bluetooth of the cell phone with the Bluetooth of the Raspberry pi.

The Mobile application has been implemented using MIT app inventor website, MIT App Inventor is a web application integrated development environment originally provided by Google, and now maintained by the Massachusetts Institute of Technology.

It is an intuitive, visual programming environment that **allows everyone to build fully functional apps for smartphones and tablets**.

### 8.2.1) The GUI of the Mobile application:

- **Screen1:**


*Figure 72 Screen1 Mobile App*

- **Screen2:**

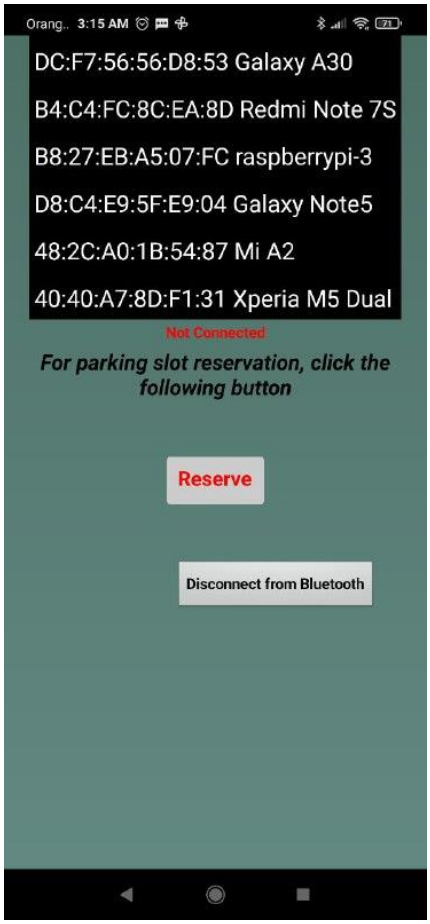



*Figure 73 Screen2 Mobile App*

## 8.2.2) The Code used for implementing the functionality:

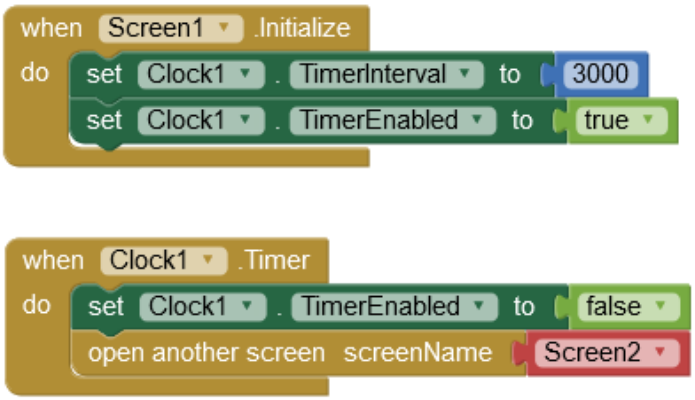



**Figure 74 Code screen1**

**Explanation:** When screen1 initializes it stays for 3 seconds and then turns to the second screen

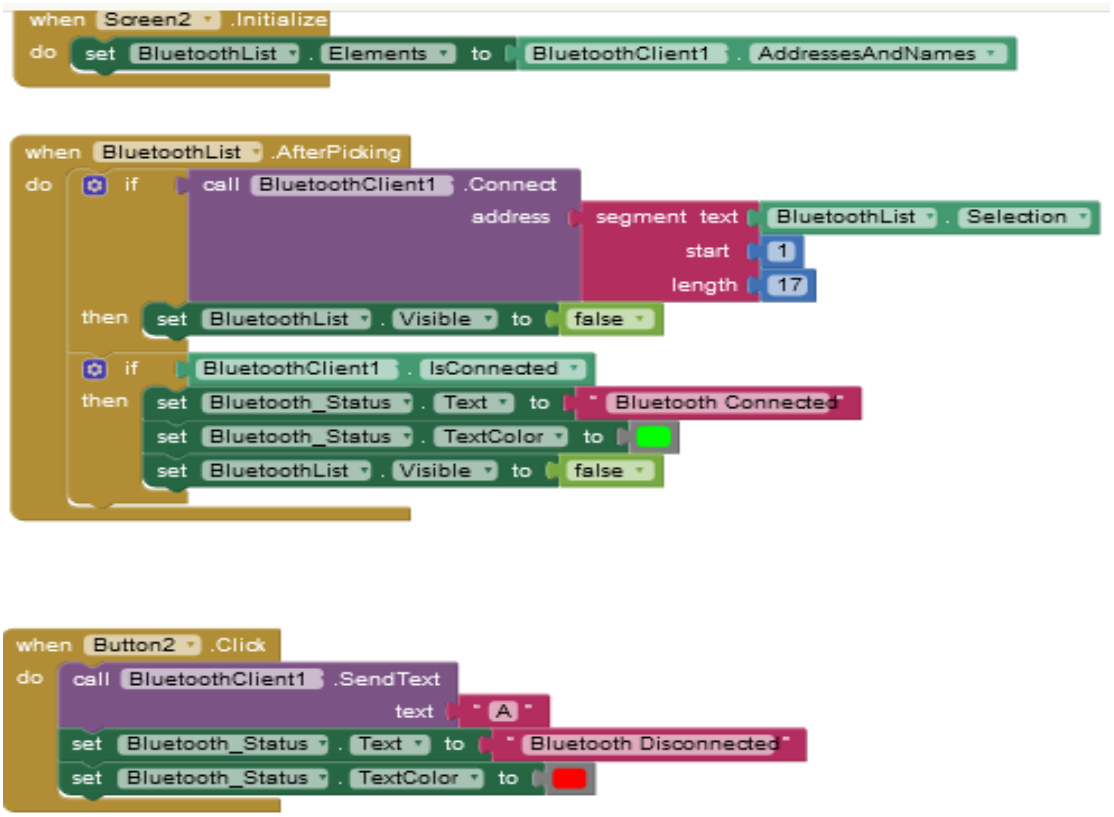
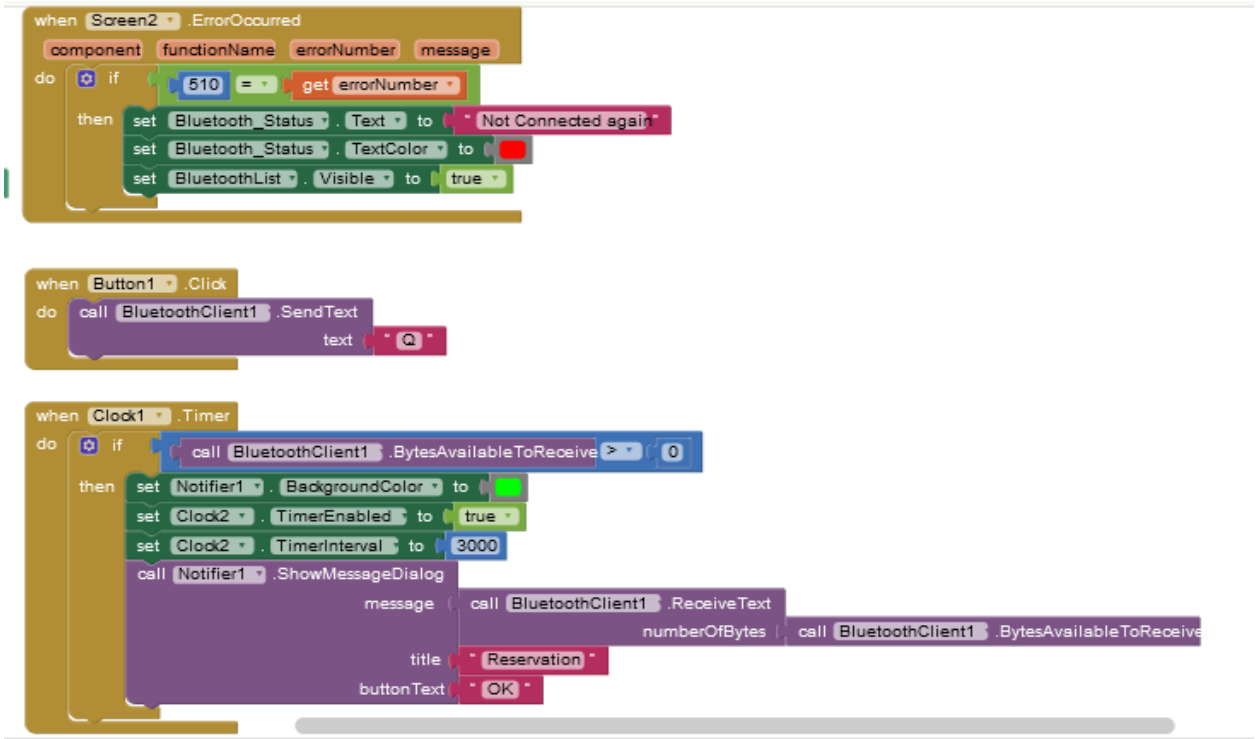



*Figure 75 Code screen2*




**Explanation:** When screen 2 initializes, it shows the list of available paired devises and after picking the list disappears and the text of a label is changed from “Not Connected” to

"Bluetooth Connected" to ensure the user that its ready for reserving. When the Reserve button is clicked, "Q" letter is transmitted when the Raspberry pi receives it, it begins the procedure of reserving the parking slot.

The Raspberry pi then sends the acknowledgement that the slot has been reserved which appears to the user in a notifier that notifies them that their place has been reserved.

When the user finishes using the application, they can click the "Disconnect from Bluetooth" button which disconnects the user from the Bluetooth as well as end the script in the Raspberry pi.

### 8.2.3) Setting Up Raspberry Pi as Bluetooth Server:

- **Libraries installation:**

  sudo apt-get install bluetooth blueman bluez
  sudo apt-get update
  sudo apt-get install bluetooth bluez libbluetooth-dev
  sudo apt-get install python3-bluez

- **Pairing device and adding a channel for communication:**

  service bluetooth start
  service bluetooth status [To quit: Ctrl Z]
  sudo bluetoothctl
  [bluetooth]# power on
  [bluetooth]# agent on
  [bluetooth]# discoverable on
  [bluetooth]# pairable on
  [bluetooth]# scan on
  [bluetooth]# pair MAC BLUETOOTH MOBILE
  [bluetooth]# paired-devices
  [bluetooth]#quit

  sudo sdptool add --channel=22 SP
  sudo sdptool browse local

  If you get this error: Failed to connect to SDP server on FF:FF:FF:00:00:00:No such file or directory

  open, edit this file:

  **sudo nano /lib/systemd/system/bluetooth.service**

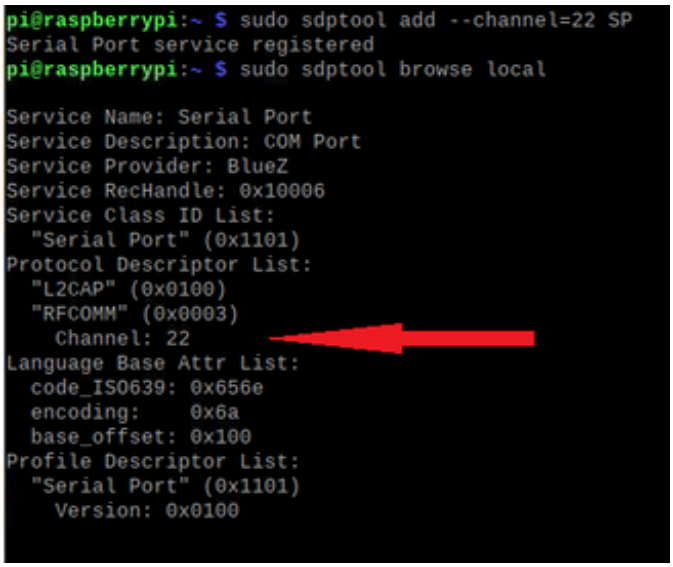
and add --compat in this line:
**ExecStart=/usr/lib/bluetooth/bluetoothd –compat**

Then Restart.

**sudo systemctl daemon-reload**
**sudo systemctl restart Bluetooth**

**sudo sdptool browse local**

channel 22 activated




*76 figure showing port 22 registered*

➔ Now the Raspberry pi is ready for receiving Bluetooth client connections through the python script in our case the Car script.

## 8.3) Protocols used:

There are three main protocols used in the IOT Model (MQTT, TCP) and the TLS/SSL protocol which is used as an extension to TCP protocol for acquiring security. Hence, MQTT is then called MQTTs same as HTTP protocol.

- The following figure shows the main TCP/IP model and the corresponding protocols in each layer for the IOT field.

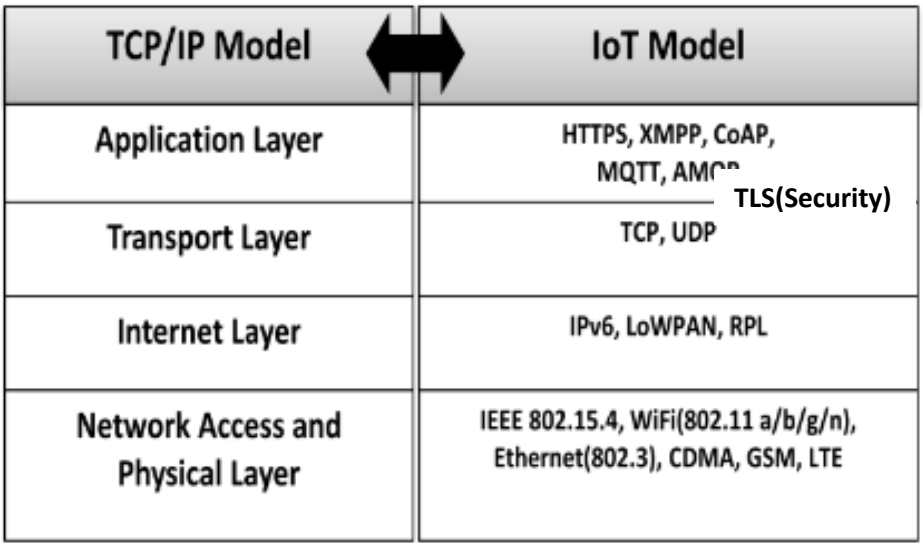



*Figure 77TCP Model*

- **Application layer:** is the layer which interacts with the application (in our case it's the python script) and connects it with the lower layer (Transport layer) in the OSI model.
    - The application layer protocol here is the MQTT, utilizing a series of actions to determine what's being done with the messages sent or received.
- **Transport layer:** It is responsible for multiplexing and demultiplexing of different processes existing in the application layer as well as providing other services such as ensuring that the data packets arrive accurately and reliably between sender and receiver. The transport layer most often uses TCP or User Datagram Protocol (UDP).
    - The transport layer here is the TCP protocol for reliability (assured delivery and packet error checking), fragmentation and ordering.
    - TLS protocol might be used, as bare MQTT sends connection credentials in plain text format and does not include any measures for security or authentication.

## 8.3.1) MQTT Protocol:

### 8.3.1.1) HTTP VS MQTT:

- MQTT was designed for the Internet of Things (although it wasn't called that at the time) whereas HTTP was created to make documents available across the internet. They both run over TCP connections, and are both client-server in architecture, but MQTT allows messages to pass in both directions between clients and servers whereas HTTP servers only respond to requests from clients.

  Thus, HTTP is mainly a **pull protocol** in which someone loads information on a Web server and users use HTTP to pull the information from the server at their convenience. While MQTT is a "PUSH" system in which the producers push data to brokers.

- MQTT protocol is considered a lightweight protocol in which its header size is 2 bytes while HTTP header size is 8 bytes.

- MQTT is designed for low power consumption as it is used for low power IOT devices while HTTP is a heavyweight protocol since HTTP utilizes heavy system resources as explained above, this also leads to heavy power consumption.

### 8.3.1.2) How MQTT protocol works:

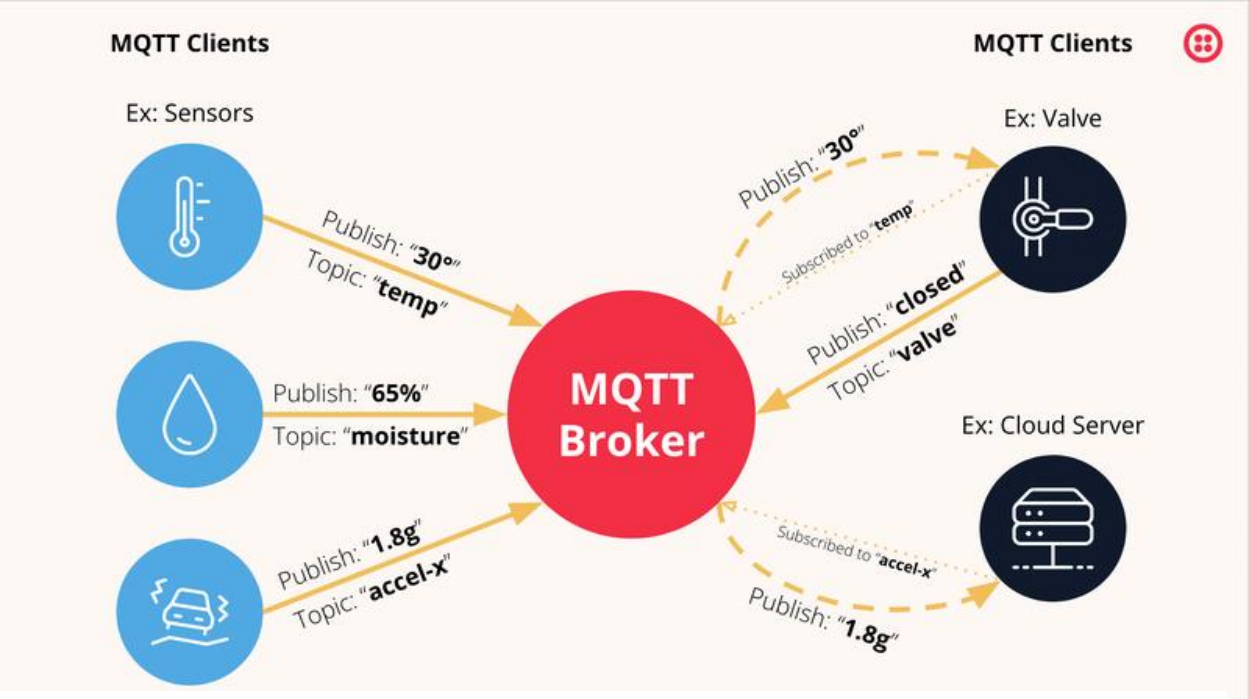
MQTT communication works as a publish and subscribe system. Devices publish messages on a specific topic. All devices that are subscribed to that topic receive the message.


### 8.3.1.3) MQTT Basic Concepts:

- **Messages:** Messages are the information that you want to exchange between your devices. It can be a message like a command or data like sensor readings, for example.
- **Topics:** Topics are the way you register interest for incoming messages or how you specify where you want to publish the message.

- **Publish/Subscribe:** In a publish and subscribe system, a device can publish a message on a topic, or it can be subscribed to a particular topic to receive messages
  - For example **Device 1** publishes on a topic.
  - **Device 2** is subscribed to the same topic that **device 1** is publishing in.


  - So, **device 2** receives the message.
- **Broker:** The **MQTT broker** is responsible for **receiving** all messages, **filtering** the messages, **deciding** who is interested in them, and then **publishing** the message to all subscribed clients.




*Example on how MQTT works.*

### 8.3.2) TLS Protocol:

- TLS is a protocol used for data encryption and authentication which means it can prove the identity of both server and the user. Mutual authentication is made in case of mTLS.
- Encryption is done using asymmetric encryption which means there are private and public keys so when one is used to encrypt the other is used to decrypt and vice versa. Hence, the server encrypts using the private key and when the message reaches the user they decrypt it using the public key.

Initially when the user requests to visit a site, the server then responds with the **server certificate** which includes the public key of the server. The client issues the server a challenge by encrypting a secret number using the public key in the certificate and asking the server to decrypt it. Only someone with access to the corresponding private key will be able to decrypt the number successfully. This in effect proves that the server has the private key (which means the server is the owner of this certificate.

If this site is Google for instance, what if this server claims to be Google generating another private and public keys, to make this authentication possible, Certificate Authority must exist. A Certificate Authority is a trusted third party that can "sign" certificates adding their stamp of approval. This allows the verifier to know that they mutually trust somebody. A signature is done by using the trusted third party's private key to encrypt a known value, which is then added to the certificate. A verifier can then use the signer's public certificate to decrypt that known value, and verify that it matches the expectation.

The signed certificate by the CA is called x.509 certificate.

To make a new X.509 certificate you need to create a Certificate Signing Request (CSR) and give it to a Certificate Authority (CA). The CSR is a digital document that contains your public key and other identifying information. When you send a CSR to a CA it first validates that the identifying information you've supplied is correct, for example you may be asked to prove ownership of a domain by responding to an email. Once your identity has been verified, the CA creates a certificate and signs it with a private key. Anyone can now validate your certificate by checking its digital signature with the CA's public key.

If the private key of the CA is stolen, then the one stealing it can create their own bogus certificates with any name they like and sign them as the original CA. This would obviously compromise security for lots of sites, and also destroy the CA's reputation in a way it would likely never recover from.

For that reason, intermediate CAs are created, they are different entities being signed by the CA private key. Thus, those intermediate CAs can sign other servers. Therefore, a **chain of trust** has been created.

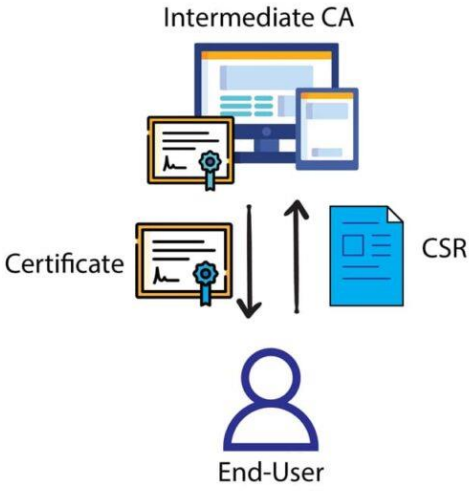



*Figure 78 CSR to Intermediate CA*

- The certificate of the original CA is called the root certificate, the ones below it are called intermediate certificate and the server requesting to be signed are called End-Entity Certificate (also known as the leaf certificate or server certificate)

### 8.3.2.1) The way an Intermediate CA is created is simple:

1. First, the Root CA signs the Intermediate CA's certificate, the Intermediate Certificate, with its own private key, thus making that Intermediate Certificate trusted.
2. The Intermediate CA now has the ability to use its Intermediate Certificate to sign end-user certificates.
    - Intermediate CAs can also sign other Intermediate CA's Intermediate Certificates, creating more links in the Chain of Trust leading back to the Root Certificate.
- When the user requests to visit a server, the server sends its server certificate and all intermediate certificates. At the user side , they start to check each one from bottom to top till reaching the root CA. the root certificate is built in the user's browser so the server would not need to send it.

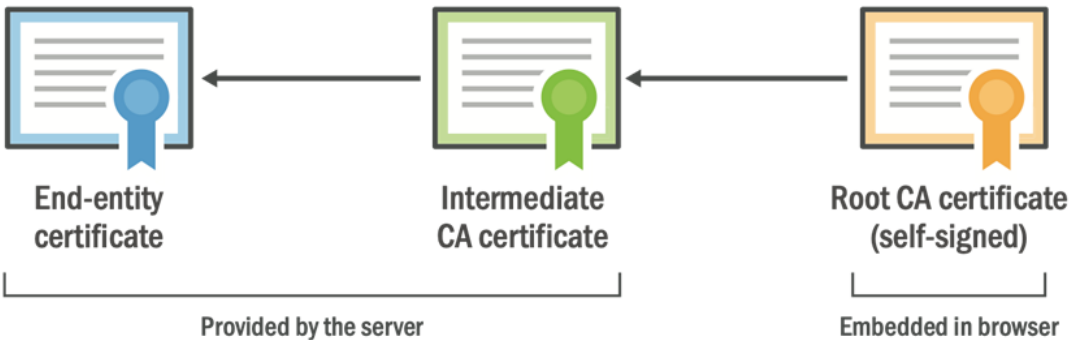



*Figure 79 figure showing where certificates are stored*

## 8.4) AWS IOT Core Connection:

The first method of connecting the two Raspberry Pis is through AWS cloud which has an AWS IOT core service implemented within it. AWS IOT core behaves as a broker managing received messages from the devices (known as things).

### 8.4.1) Steps for setting up Raspberry Pi:

1) Installing AWS IOT SDK.

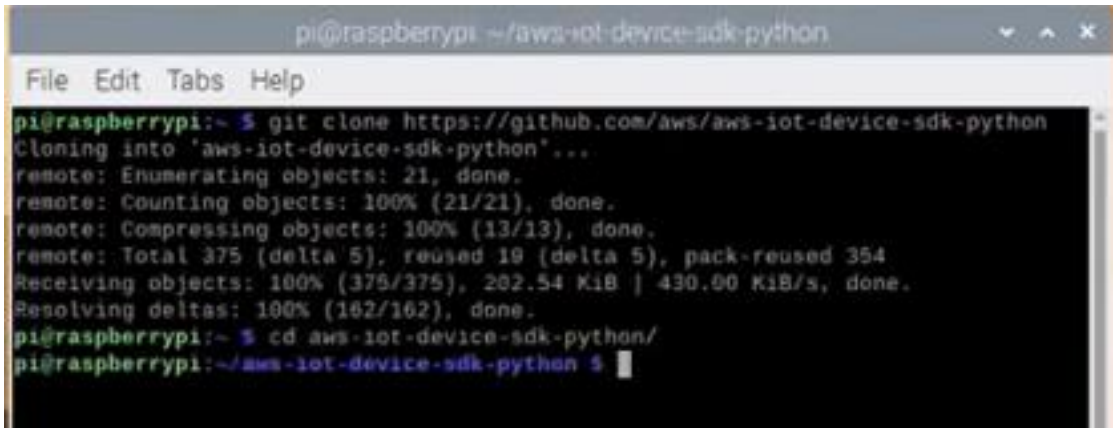



*Figure 80 installing required libraries for AWS*


- AWS IOT SDK is successfully installed.

2) Installing Paho MQTT library.

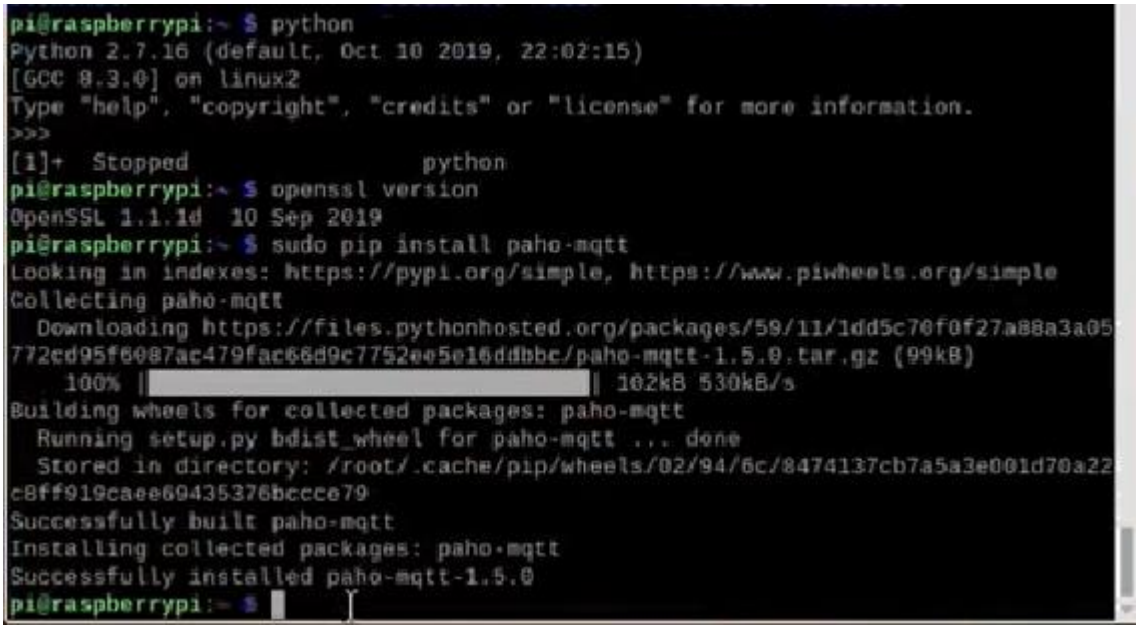
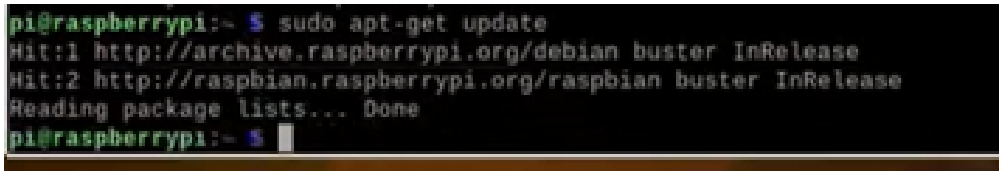



**Figure 81 Installing Paho.mqtt libraries**




## 8.4.2) Steps for setting up AWS IOT core:

1- Creating AWS account.

2- Registering the Raspberry pi as a thing in AWS IOT core.

- Registering a device on AWS IOT enables us to generate the certificates for TLS connection as well as enabling the access to various AWS IOT functions.
- The registration also eases the management of many devices if they exist. In our project there are three Raspberry Pis (one in the Garage and the other two are for the vehicles).

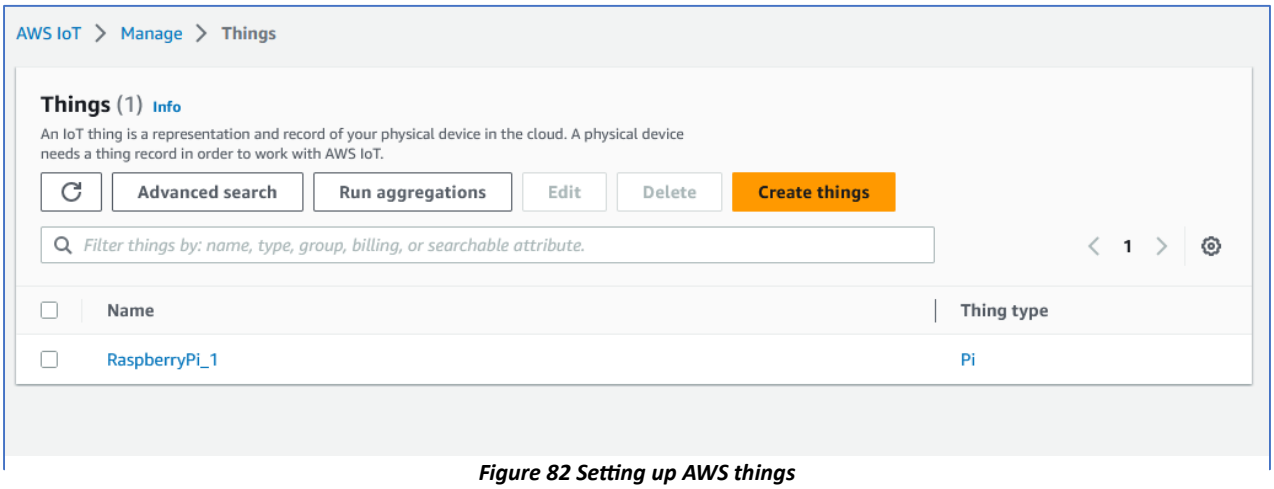
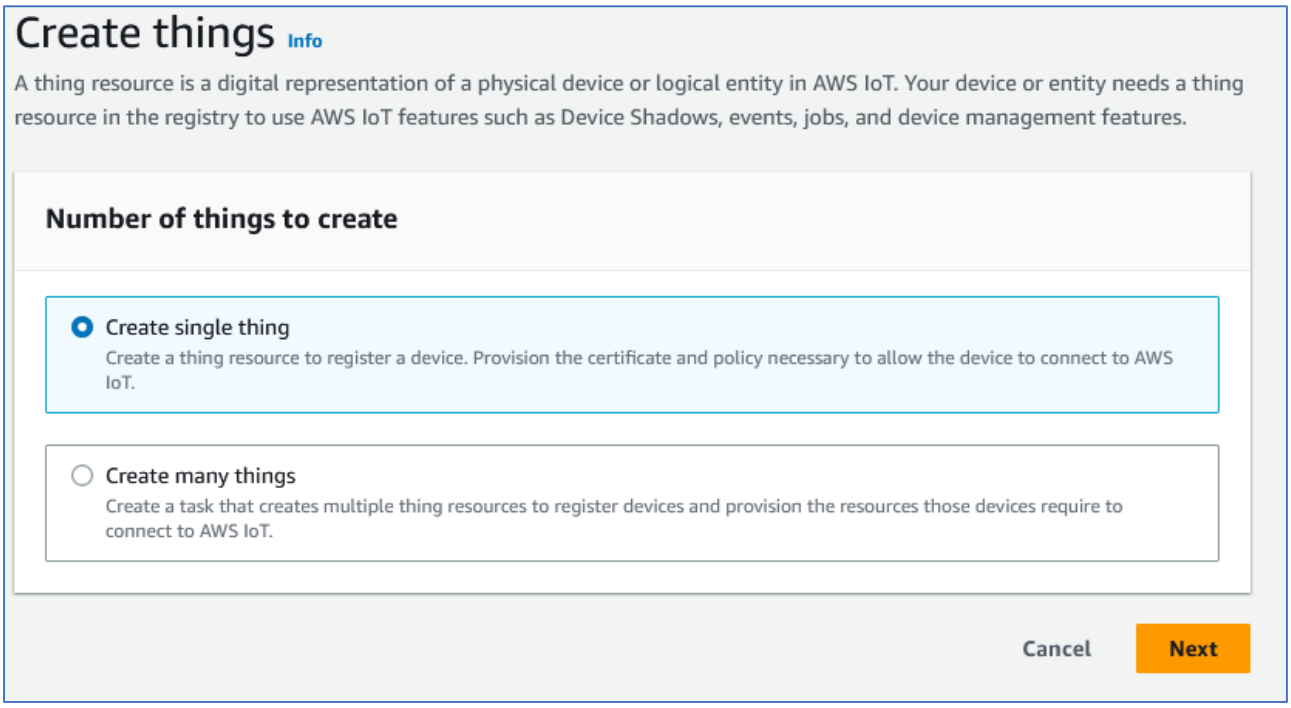
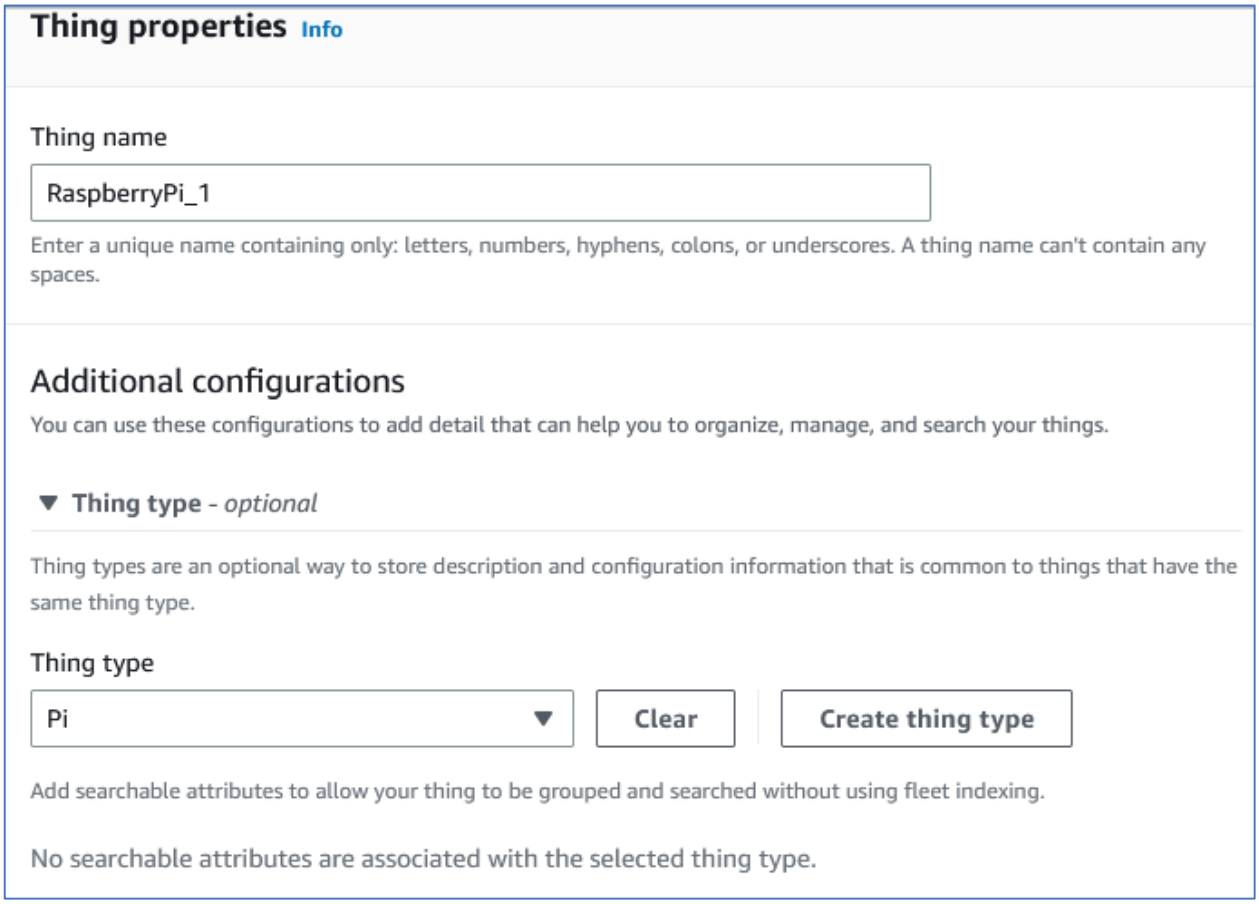
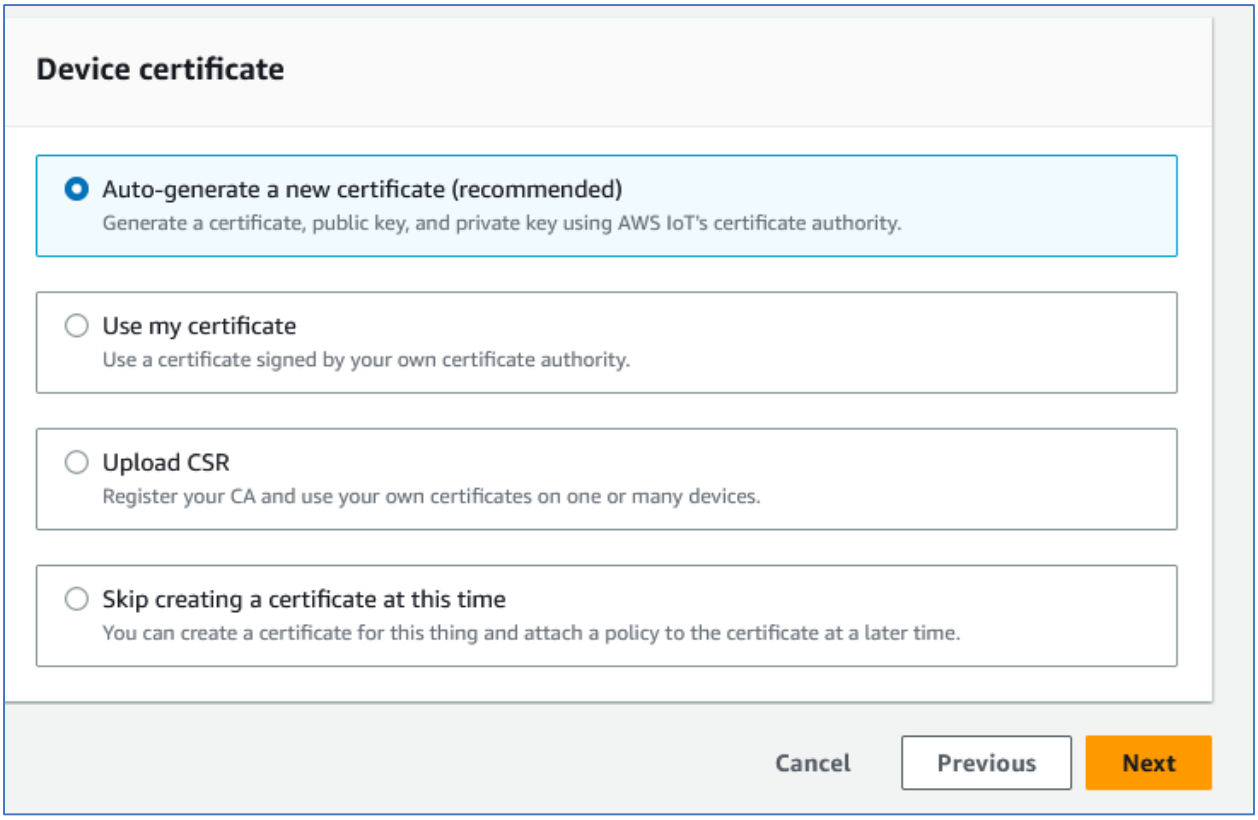
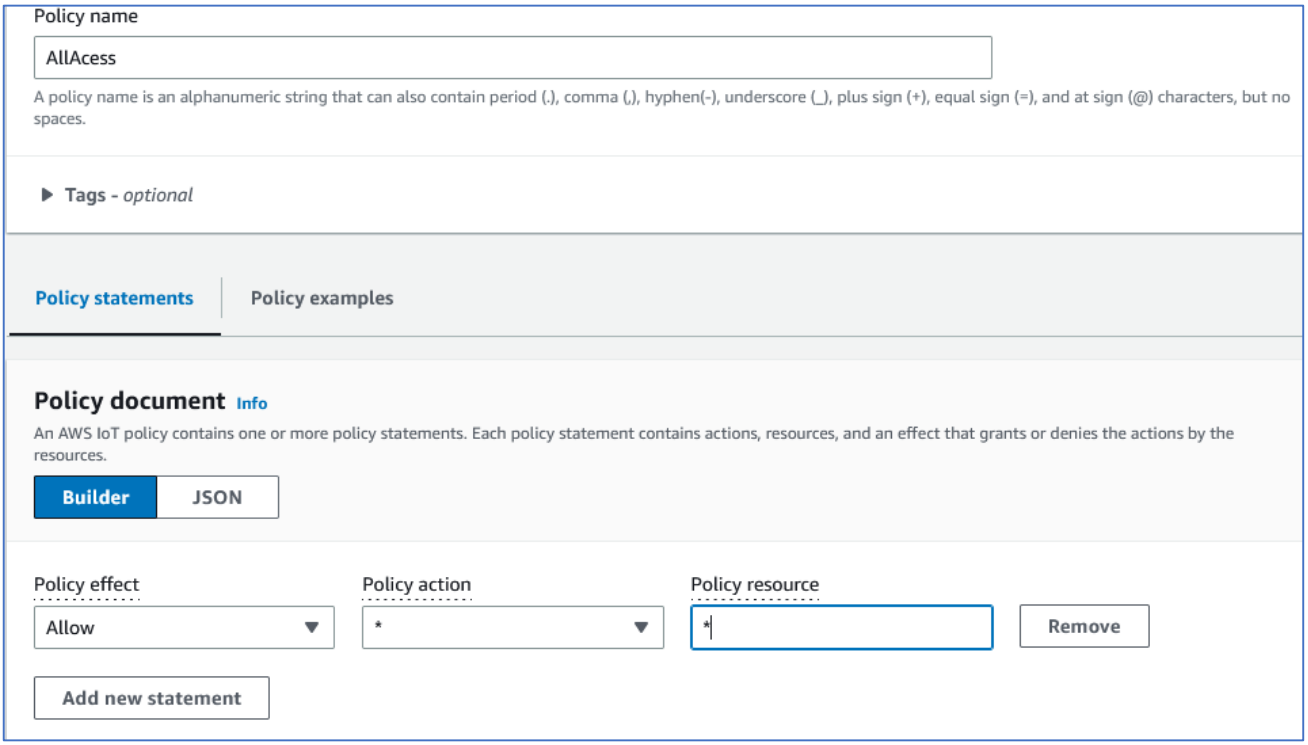



***Figure 82 Setting up AWS things***

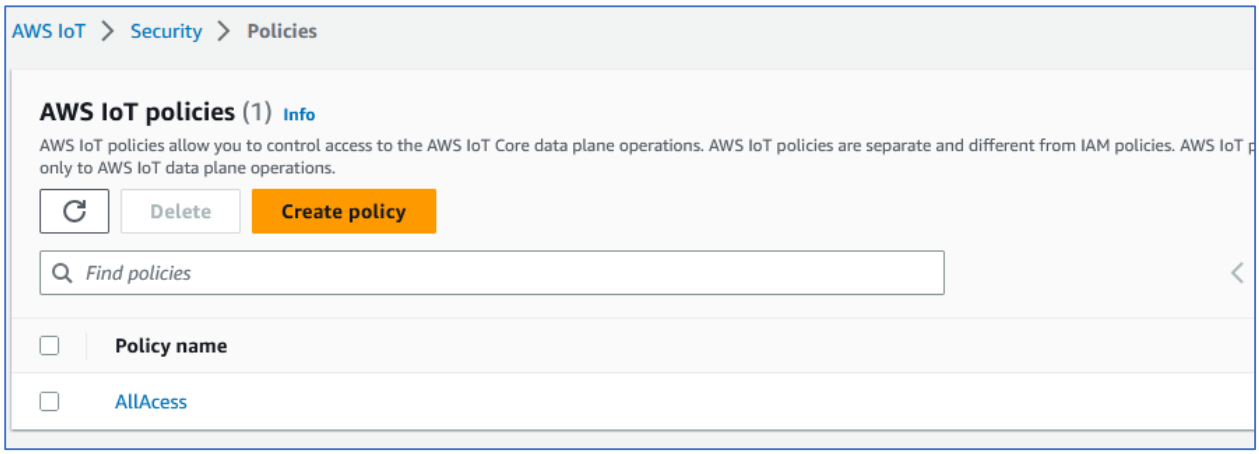

Create things Info
A thing resource is a digital representation of a physical device or logical entity in AWS IoT. Your device or entity needs a thing resource in the registry to use AWS IoT features such as Device Shadows, events, jobs, and device management features.
Number of things to create
Create single thing
Create a thing resource to register a device. Provision the certificate and policy necessary to allow the device to connect to AWS IoT.
Create many things
Create a task that creates multiple thing resources to register devices and provision the resources those devices require to connect to AWS IoT.
Cancel
Next

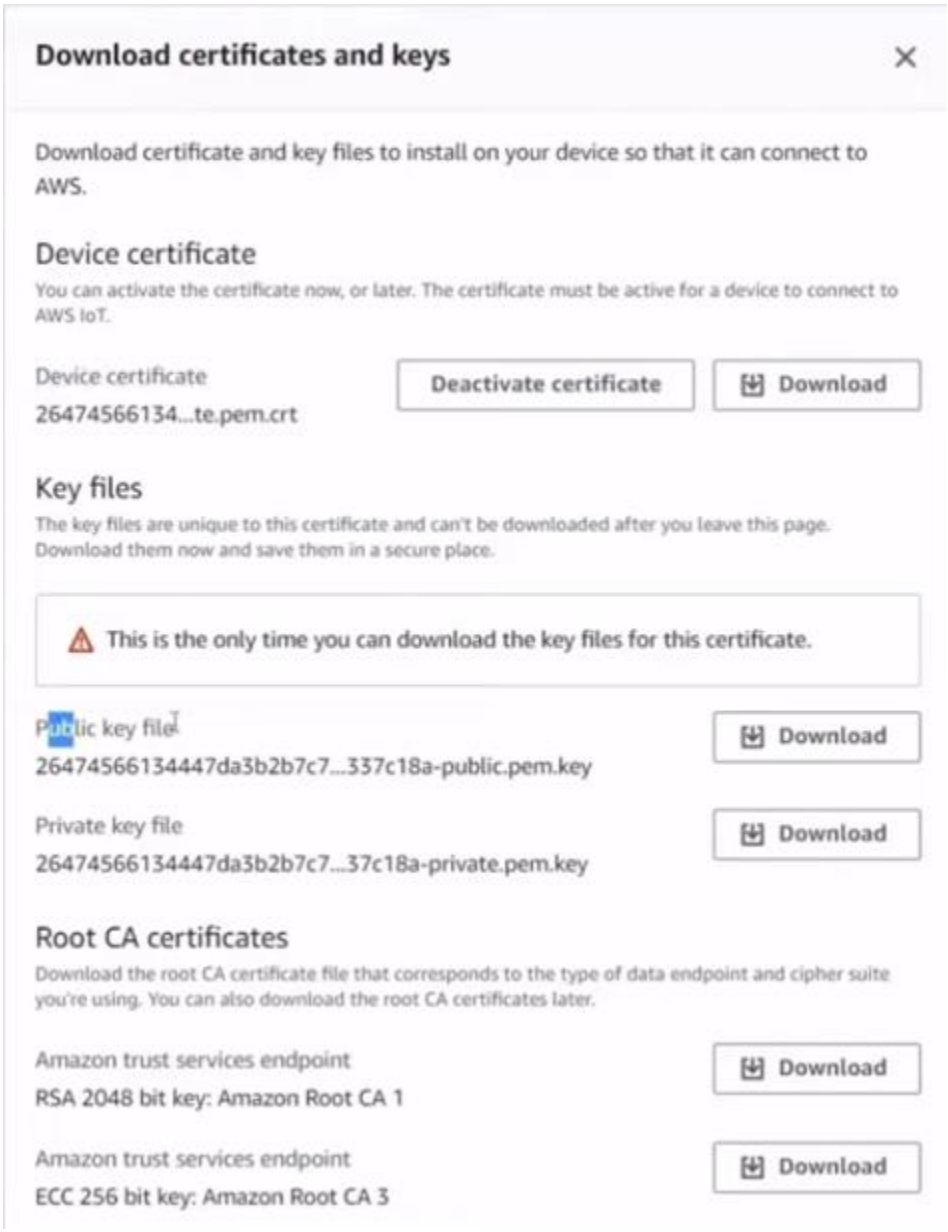

Thing properties Info
Thing name
RaspberryPi_1
Enter a unique name containing only: letters, numbers, hyphens, colons, or underscores. A thing name can't contain any spaces.
Additional configurations
You can use these configurations to add detail that can help you to organize, manage, and search your things.
Thing type - optional
Thing types are an optional way to store description and configuration information that is common to things that have the same thing type.
Thing type
Pi
Clear
Create thing type
Add searchable attributes to allow your thing to be grouped and searched without using fleet indexing.
No searchable attributes are associated with the selected thing type.


Device certificate
Auto-generate a new certificate (recommended)
Generate a certificate, public key, and private key using AWS IoT's certificate authority.
Use my certificate
Use a certificate signed by your own certificate authority.
Upload CSR
Register your CA and use your own certificates on one or many devices.
Skip creating a certificate at this time
You can create a certificate for this thing and attach a policy to the certificate at a later time.
Cancel
Previous
Next

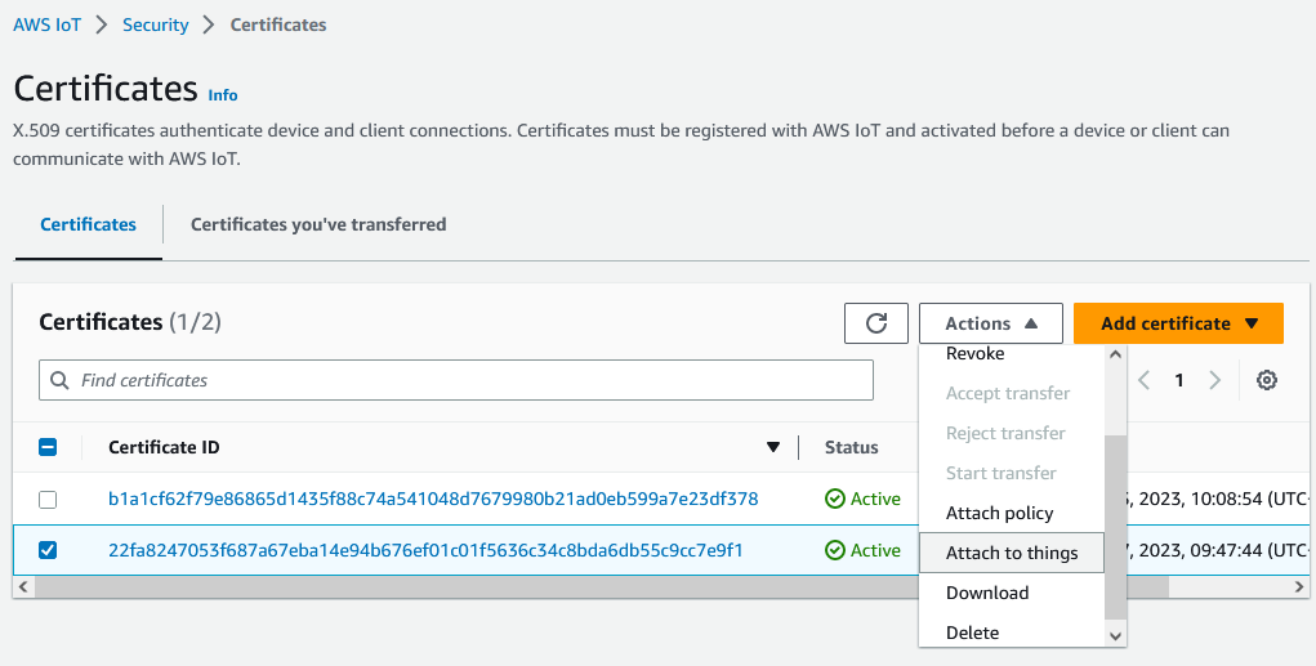

Policy name
AllAcess
A policy name is an alphanumeric string that can also contain period (.), comma (,), hyphen(-), underscore (_), plus sign (+), equal sign (=), and at sign (@) characters, but no spaces.
Tags - optional
Policy statements
Policy examples
Policy document Info
An AWS IoT policy contains one or more policy statements. Each policy statement contains actions, resources, and an effect that grants or denies the actions by the resources.
Builder
JSON
Policy effect
Allow
Policy action
*
Policy resource
*
Remove
Add new statement

- A policy is needed because:
    - when a thing is first created, all the actions are being disabled from the device, but we need it to connect and send and receive messages. Hence, we will allow all the possible actions to the device.
    - The policy resource is also set to all (*) to allow the access to all existing AWS resources if needed.






- Downloading the required certificates (Device Certificate, Private key, Amazon Root CA1).




- Attaching the policy made and the thing created to the certificate just made.

3) Creating a folder in the Raspberry pi and adding the certificates.

## 8.5) LAN Network (Edge) Connection:

The second way for connecting the Raspberry Pis is through a local broker which is Mosquitto broker.

## 8.6) Paho MQTT Client Library:

The core of the client library is the client class which provides all of the functions to publish messages and subscribe to topics.

### 8.6.1) Main Client Methods

The Paho MQTT client class has several methods. The main ones are:

- connect(host, port=1883) : used to establish a connection to a broker
  - There are 4 default Parameters for the connect method but we will assign two of them only (IP of the broker and port number to publish and subscribe messages to it)
  - The connect method is a blocking function which means that your script will stop while the connection is being established.
  - When a client issues a connect request to a broker that request should receive an acknowledgment.

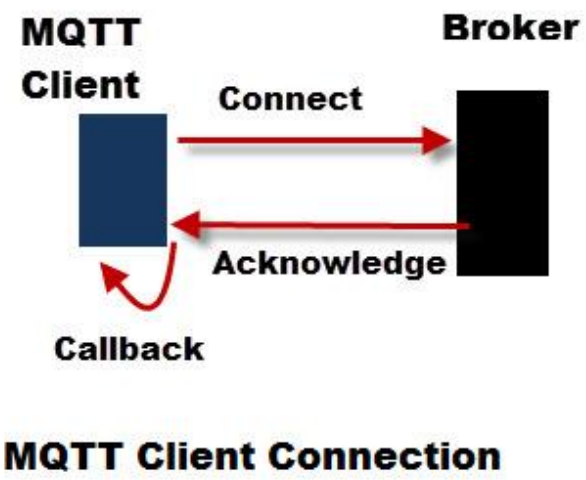



**Figure 83 CONNACK message**

- subscribe(topic): used to listen to specific Topic receiving at broker.
- Publish (topic, payload): used to publish messages to the broker

Each of these methods is associated with a **callback.**

### 8.6.1.1) Callbacks:

Callbacks are functions that are called in response to an event.

The events and callbacks for the Paho MQTT client are as follows:

- **Event** Connection acknowledged **Triggers** the **on_connect** callback.
- **Event** Disconnection acknowledged **Triggers** the **on_disconnect** callback.
- **Event** Message Received **Triggers** the **on_message** callback.

The Paho client is designed to only use the callback functions if they exist, and doesn't provide any **default** callback functions.

To use a callback you need to do two things:

1. Create the callback function
2. Assign the function to the callback.

**Callbacks and the Client Loop:**

Callbacks are dependent on the client loop as without the loop the callbacks aren't triggered.

- **client. loop_start()**

- **What this loop does and why is it necessary?**

  When new messages arrive at the Python MQTT client they are placed in a receive buffer.

  The messages sit in this receive buffer waiting to be read by the client program.

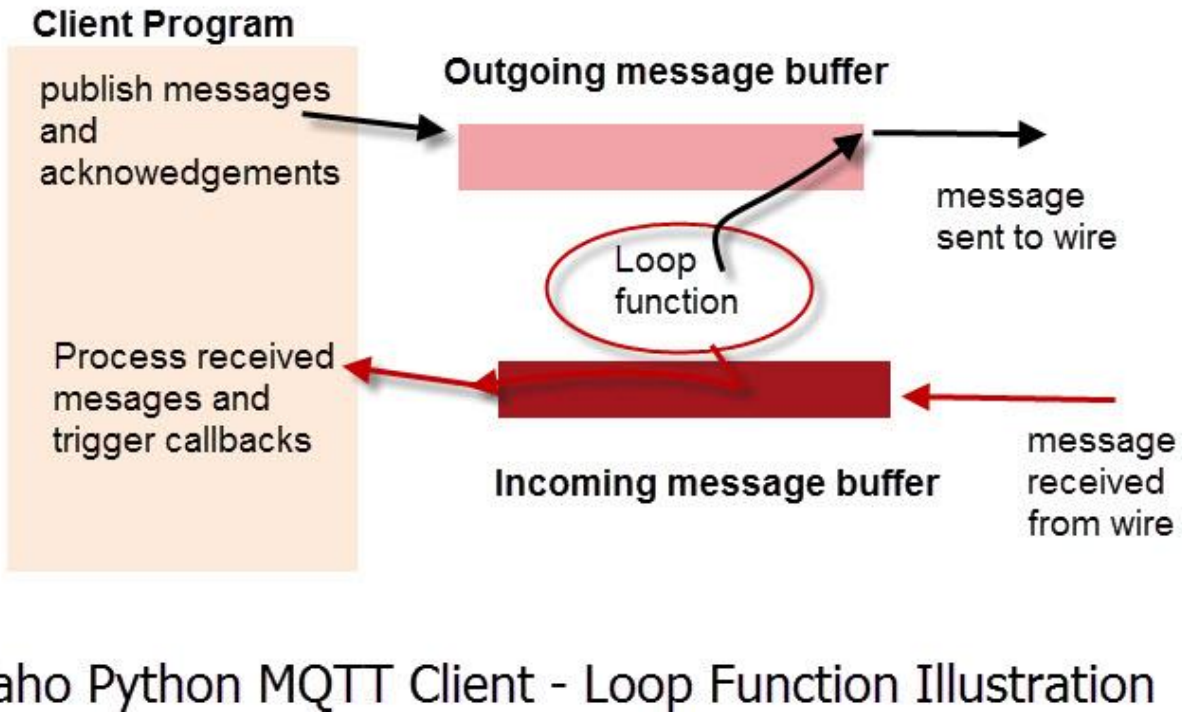



*Figure 84 Illustration of how loop function works*

The loop() function is a built in function that will read the receive and send buffers, and process any messages it finds.

On the receive side it looks at the messages, and depending on the message type, it will trigger the appropriate callback function.

For example if it sees a CONNACK message it triggers the on_connect() callback.

Now instead of manually reading the receive buffer you just need to process the callbacks.

Outgoing messages and message acknowledgements are placed in the send buffer.

The loop function will read this buffer and send any messages it finds.

The loop_start() starts a new thread, that calls the loop method at regular intervals for you. It also handles re-connects automatically.

## 8.7) Car and Garage scripts:

### 8.7.1) Garage script:

```
import paho.mqtt.client as paho
import ssl
import string
import json
import serial
from time import sleep
from gpiozero import LED
import RPi.GPIO as GPIO # Import Raspberry Pi GPIO library

LED PIN = 31
```

- Importing the necessary libraries and setting a pin to 31 which is going to be used to give a signal for the AVR existing in the garage that a parking slot has been reserved.

```
connflag = False
available_slots=1

def on_connect(client, userdata, flags, rc):                # func for
    global connflag
    if rc==0 and connflag== False:
        print ("Connected to Edge")
    elif rc==0 and connflag== True:
        print ("Connected to AWS")
    connflag=True
    print("Connection returned result: " + str(rc) )

```

- On_connect() function is used as a call back function which is called when a connection is established either with the Edge or AWS.

```
def on_message(client, userdata, message):
    global available_slots
    global counter

    decoded_message=str(message.payload.decode("utf-8"))
    dic_message= json.loads(decoded_message)
    print("received message= ",decoded_message)


    if message.topic== "reservation":
        if available_slots> 0:

            print("Parking slots are available")
            GPIO.output(LED_PIN, GPIO.LOW)
            sleep(0.5)

            GPIO.output(LED_PIN, GPIO.HIGH)

            #ser.write(chr(available_slots).encode(encoding="ascii"))
            msg={"ack":"The Parking slot has been reserved"}
            msg_JSON=json.dumps(msg)
            sleep(0.1)
            Garage.publish("reservation_ack",msg_JSON)
        elif available_slots==0:
            msg={"Nack":"Unfortunately, all parking slots are reserved'
            print("Number of available slots in Garage: ",available_slo
            msg_JSON=json.dumps(msg)
            sleep(0.1)
            Garage.publish("reservation_Nack",msg_JSON)

```

- If the Garage received a message, it should check if there are available slots or not, if there are available slots, it will give a signal indicating a received request and it will publish an acknowledgement to the vehicle.

```
def on_disconnect(client, userdata, rc):
    print("Disconnected from Edge")
    Garage.tls_set(caPath, certfile=certPath, keyfile=keyPath, cert_req
    Garage.connect(awshost, awsport, keepalive=60)
```

- If the vehicle disconnected from Edge network, it will try to reconnect with AWS, since disconnecting means out of range.

```
Garage= paho.Client()
Garage.on_connect = on_connect   #assigning functions to callbacks
Garage.on_message = on_message
Garage.on_disconnect=on_disconnect
ser = serial.Serial ("/dev/ttyS0", 9600)

######################
awshost = "a2p87hu0j8wq89-ats.iot.us-east-1.amazonaws.com"      # Endp
awsport = 8883                                                 # Port no.
clientId = "RaspberryPi_1"                                        # Thing
thingName = "RaspberryPi_1"                                       # Thing
caPath = "/home/pi/AWS/AmazonRootCA1.pem"
certPath = "/home/pi/AWS/certificate.pem.crt"
keyPath = "/home/pi/AWS/private.pem.key"

```

- Setting the parameters for AWS.
- AWS endpoints are given for specific account. Each account has several device endpoints that are unique to the account and support specific IoT functions. This particular one is the AWS IoT device data endpoints support a publish/subscribe protocol that is designed for the communication needs of IoT devices.
- AWS endpoint can be found in the settings.

```
Edgehost = "192.168.1.5"
Edgeport = 1883

Garage.loop_start()
try:
    Garage.connect(Edgehost, Edgeport, keepalive=60)
    print('trying')
#Garage.tls_set(caPath, certfile=certPath, keyfile=keyPath, cert_reqs=s
#Garage.connect(awshost, awsport, keepalive=60)     # connect to aws se
except:
    connflag = True
    Garage.tls_set(caPath, certfile=certPath, keyfile=keyPath, cert_rec
    Garage.connect(awshost, awsport, keepalive=60)
```

- It will first try to connect to the edge, if it’s unavailable, it will try to connect to amazon without giving errors.

```
sleep(2) # wait
Garage.subscribe("reservation",qos=0)

GPIO.setwarnings(False) # Ignore warning for now
GPIO.setmode(GPIO.BOARD) # Use physical pin numbering
GPIO.setup(37, GPIO.IN, pull_up_down=GPIO.PUD_DOWN) # Set pin 10 to be
GPIO.setup(LED_PIN, GPIO.OUT)
GPIO.setup(LED1, GPIO.OUT)
GPIO.output(LED_PIN, GPIO.HIGH)
```

- The Garage will subscribe to reservation topic to keep listening for any reservation request.
- Setting pin 37 as input to take a signal from the AVR if no available slots are available.

```
while 1:
    if GPIO.input(37)==1:
        available_slots=0
        print(available_slots)
    elif GPIO.input(37)==0:
        available_slots=1
```

- If pin 37 receives high, then no slots will be available
- If pin 37 receives low , then available_slots=1 indicating there are available slots in the garage.

### 8.7.2) Car script:

```
import RPi.GPIO as GPIO # Import Raspberry Pi GPIO library
import paho.mqtt.client as paho
import ssl
import string
import json
import serial
from time import sleep
import bluetooth
```

- Importing the necessary libraries.

```
############################
server_sock=bluetooth.BluetoothSocket(bluetooth.RFCOMM)
port = 22
server_sock.bind(("",port))
server_sock.listen(1)
client_sock,address = server_sock.accept()
print ("Connection made with: ", address)

############################
```

- Initializing Bluetooth socket server and starting listening for Bluetooth clients and accepting the coming request.

```
connflag = False
dic_message={}
flag=0
string=""
def on_connect(client, userdata, flags, rc):
    global connflag
    if rc==0 and connflag== False:
        print ("Connected to Edge")
    elif rc==0 and connflag== True:
        print ("Connected to AWS")
    connflag=True
    print("Connection returned result: " + str(rc) )
```

- On_connect() function is used as a call back function which is called when a connection is established either with the Edge or AWS.

```
def on_message(client, userdata, message):
    global dic_message
    global flag
    global string
    if flag != 1:
        flag+=1
    elif message.topic=="reservation_Nack":
        decoded_message=str(message.payload.decode("utf-8"))
        dic_message= json.loads(decoded_message)
        print("received message= ",dic_message["Nack"])
        string= str(dic_message["Nack"])
        client_sock.send(string)
        sleep(2)
```

```
    elif message.topic=="reservation_ack":
        decoded_message=str(message.payload.decode("utf-8"))
        dic_message= json.loads(decoded_message)
        print("received message= ",dic_message["ack"])
        string= str(dic_message["ack"])
        print(string)

        #if flag==0:
        client_sock.send(string)
        sleep(2)
```

- When an ACK or NACK is received at the vehicle, it is sent again via Bluetooth to the mobile application to notify the user if their request is accepted or all the parking slots have been reserved.

```
def on_disconnect(client, userdata, rc):
    print("Disconnected from Edge")
    Car_1.tls_set(caPath, certfile=certPath, keyfile=keyPath, cert_re
    Car_1.connect(awshost, awsport, keepalive=60)
```

```
def button_callback():
    global flag
    flag=0
    print("Button was pushed!")
    Car_1.subscribe("reservation_ack",qos=0)
    Car_1.subscribe("reservation_Nack",qos=0)
    msg={ "required_slots":1 }
    msg_id={"ID":1}
    msg_JSON=json.dumps(msg)
    msg_id_JSON=json.dumps(msg_id)
    Car_1.publish("reservation",msg_JSON)
    Car_1.publish("Car_ID",msg_id_JSON)
```

- This function is concerned with reserving a parking slot whenever its called and publishing the reservation request to the garage.
- The car ID is also published, so that the gate at the Garage is opened if it detects this ID.

```
76  Car_1= paho.Client(clean_session=True)
77  Car_1.on_connect = on_connect   #assigning functions to callbacks
78  Car_1.on_message = on_message
79  Car_1.on_disconnect=on_disconnect
80  #ser = serial.Serial ("/dev/ttyS0", 9600)
81  ######################
82  awshost = "a2p87hu0j8wq89-ats.iot.us-east-1.amazonaws.com"      # Endp
83  awsport = 8883                                                 # Port no.
84  clientId = "RaspberryPi_1"                                        # Thing
85  thingName = "RaspberryPi_1"                                       # Thing
86  caPath = "/home/pi/AWS/AmazonRootCA1.pem"
87  certPath = "/home/pi/AWS/certificate.pem.crt"
88  keyPath = "/home/pi/AWS/private.pem.key"
89
90  Edgehost = "192.168.137.1"
91  Edgeport = 1883
```

```
93  try:
94      Car_1.connect(Edgehost, Edgeport, keepalive=60)
95  #Garage.tls_set(caPath, certfile=certPath, keyfile=keyPath, cert_reqs=
96  #Garage.connect(awshost, awsport, keepalive=60)     # connect to aws s
97  except:
98      connflag = True
99      Car_1.tls_set(caPath, certfile=certPath, keyfile=keyPath, cert_req
100     Car_1.connect(awshost, awsport, keepalive=60)
101
102 Car_1.loop_start()
103 sleep(4)
```

```
117 while True:
118     |
119     recvdata = client_sock.recv(1024)
120     print ("Information received: %s" % recvdata)
121     recvdata=recvdata.decode()
122     if (recvdata == "Q"):
123         button_callback()
124         sleep(1)
125
126     elif (recvdata == "A"):
127         client_sock.close()
128         server_sock.close()
129         break
```

- The script keeps receiving from the Bluetooth, when the received text is “Q”, the button_callback() fn is called and whenever “A” is received, the sockets closed and the script ends.

# CHAPTER 9
# GPS Tracking for Accident Detection

## 9.1) Introduction:

This part is concerned with Tracking the location of the driver in the vehicle on real-time basis so that if in case an accident occurred, the location of the vehicle in the instant of accident is sent to the nearest hospital. Hence, further procedures can be taken from the hospital side like sending an ambulance.

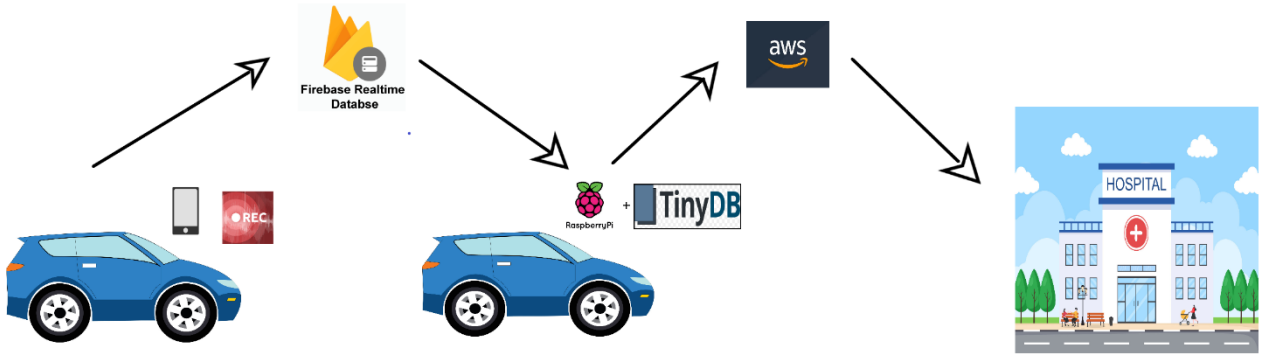



*Figure 85 showing the design of how GPS works*

The above figure descrGibes the logic behind the implementation of this scenario.

## 9.2) Components used:

- Cell phone of the user: It is going to be used instead of the GPS Module as it gives better accuracy for the location.
- Sensor logger application: It is an application used for recording the GPS information and sending it to a specific URL (which is the firebase database URL).
- Firebase database: The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in real-time to every connected client.
- TinyDB in Raspberry pi:
    - It is **a document-oriented database written in pure Python with no external dependencies**. It is designed to be easy and fun to use by providing a simple and clean API.
    - Its main function is to store the hospitals and their corresponding location.

- AWS: It is used using MQTTs protocol to send the location of the user to the nearest hospital in case an accident occurs.
- 3 Virtual machines: These virtual machines represents the hospitals for testing.
    - Hosp_1: Smouha international Hospital
    - Hosp_2: Gamal Abdul Nasser Hospital
    - Hosp_3: Maleka Nazly Hospital

## 9.3) How it works:

First, the cell phone of the user has an application called sensor logger installed on it, this application keeps tracking the devices' location even in the background mode, it then sends the information of the location including longitude and latitude as HTTP POST message to the firebase database,

Second, a python script is running on the raspberry pi checking the last location added to the database.

Once an accident occurs, a function calculating the distance between the last attained location and all the hospitals is called.

The distance is calculated using Haversine formula, The haversine formula calculates the shortest distance between two points, whose latitudes and longitudes are known, in a sphere.

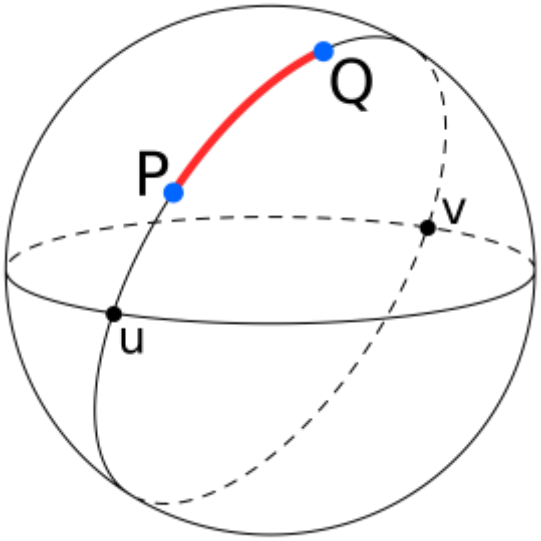



**Figure 86 2 points on sphere**

The hospital with the least calculated distance between the user and that hospital receives a message from the vehicle with its location.

## 9.4) Output of the python script while running:

```
pi@raspberrypi-3:~/GPS $ python main.py
None
Connected to AWS
Connection returned result: 0
Press enter to quit

{'deviceId': '1a498bf6-1caa-4532-b0ac-0380d7ed466d', 'messageId': 7, 'payload':
[{'name': 'location', 'time': 1689045523109000000, 'values': {'altitude': 19.299
999237060547, 'bearing': 289.5161437988281, 'bearingAccuracy': 0, 'horizontalAcc
uracy': 19.065000534057617, 'latitude': 31.2099197, 'longitude': 29.8802683, 'sp
eed': 0.9058736562728882, 'speedAccuracy': 0, 'verticalAccuracy': 1.210988521575
9277}}], 'sessionId': '14161797-7be8-4866-b20e-ec057a90c15f'}
[{'name': 'location', 'time': 1689045523109000000, 'values': {'altitude': 19.299
999237060547, 'bearing': 289.5161437988281, 'bearingAccuracy': 0, 'horizontalAcc
uracy': 19.065000534057617, 'latitude': 31.2099197, 'longitude': 29.8802683, 'sp
eed': 0.9058736562728882, 'speedAccuracy': 0, 'verticalAccuracy': 1.210988521575
9277}}]
{'altitude': 19.299999237060547, 'bearing': 289.5161437988281, 'bearingAccuracy'
: 0, 'horizontalAccuracy': 19.065000534057617, 'latitude': 31.2099197, 'longitud
e': 29.8802683, 'speed': 0.9058736562728882, 'speedAccuracy': 0, 'verticalAccura
cy': 1.2109885215759277}
longitude: 29.8802683
latitude: 31.2099197
```

**Figure 87 Script running**

```
{'deviceId': '1a498bf6-1caa-4532-b0ac-0380d7ed466d', 'messageId': 8, 'payload':
 [{'name': 'location', 'time': 1689045529140924000, 'values': {'altitude': 19.2
99999237060547, 'bearing': 277.4417724609375, 'bearingAccuracy': 0, 'horizontal
Accuracy': 19.707000732421875, 'latitude': 31.2099251, 'longitude': 29.8802413,
 'speed': 0.3479469120502472, 'speedAccuracy': 0, 'verticalAccuracy': 1.1721307
039260864}}], 'sessionId': '14161797-7be8-4866-b20e-ec057a90c15f'}
[{'name': 'location', 'time': 1689045529140924000, 'values': {'altitude': 19.29
9999237060547, 'bearing': 277.4417724609375, 'bearingAccuracy': 0, 'horizontalA
ccuracy': 19.707000732421875, 'latitude': 31.2099251, 'longitude': 29.8802413,
'speed': 0.3479469120502472, 'speedAccuracy': 0, 'verticalAccuracy': 1.17213070
39260864}}]
{'altitude': 19.299999237060547, 'bearing': 277.4417724609375, 'bearingAccuracy
': 0, 'horizontalAccuracy': 19.707000732421875, 'latitude': 31.2099251, 'longit
ude': 29.8802413, 'speed': 0.3479469120502472, 'speedAccuracy': 0, 'verticalAcc
uracy': 1.1721307039260864}
longitude: 29.8802413
latitude: 31.2099251
Button was pushed!
29.8802413 31.2099251
Maleka Nazly Hospital
```

At virtual machine terminal:

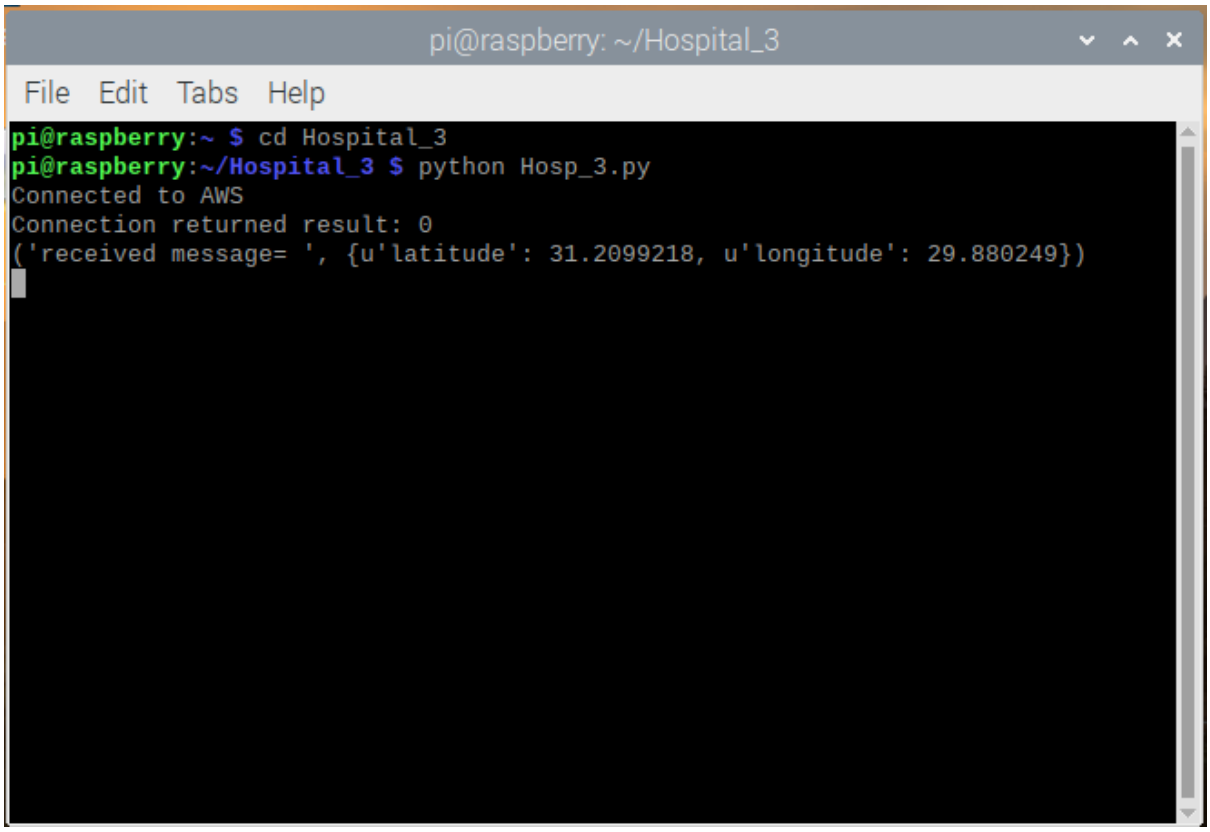



**Figure 88 Output at Virtual machine**

Hospitals_DB content stored at the Raspberry pi:

```
pi@raspberrypi-3:~/GPS $ python
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from tinydb import TinDB
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: cannot import name 'TinDB' from 'tinydb' (/home/pi/.local/lib/pytho
n3.9/site-packages/tinydb/__init__.py)
>>> from tinydb import TinyDB
>>> TinyDB('Hospitals_DB.json')
<TinyDB tables=['_default'], tables_count=1, default_table_documents_count=3, al
l_tables_documents_count=['_default=3']>
>>> db=TinyDB('Hospitals_DB.json')
>>> db.all()
[{'Hospital name': 'smouha international hospital', 'thingName': 'Smouha_interna
tional_hospital', 'longitude': 29.935054961367847, 'latitude': 31.20974804458930
4}, {'Hospital name': 'Gamal Abdel Nasser Hospital', 'thingName': 'Abd_Elnasser_
Hospital', 'longitude': 29.926221800000004, 'latitude': 31.20518116788626}, {'Ho
spital name': 'Maleka Nazly Hospital', 'thingName': 'Nazly_Hospital', 'longitude
': 29.88409061534196, 'latitude': 31.206355477664413}]
>>>
```

**Figure 85 Database stored in the RPi**

## 9.5) Python Script:

```
import queue
import json
import RPi.GPIO as GPIO # Import Raspberry Pi GPIO library
from tinydb import TinyDB, Query
from math import radians, cos, sin, asin, sqrt
import paho.mqtt.client as paho
import ssl

import firebase_admin
from firebase_admin import credentials
from firebase_admin import db
from time import sleep
import ast
```

**Figure 86 Python Script of GPS**

- Importing necessary libraries for running the script.

```
q1= queue.LifoQueue()
def listener(child_added):

    print(child_added.data)  # new data at /reference/event.path. None


    if child_added.data!= None:
        print(child_added.data["payload"])
        payload = child_added.data["payload"]
        Str = str(payload[0])
        res = ast.literal_eval(Str)
        print(res["values"])
        Dic_Data = ast.literal_eval(str(res["values"]))
        longitude1= Dic_Data["longitude"]
        latitude1= Dic_Data["latitude"]

        q1.put(longitude1)
        q1.put(latitude1)
        print("longitude:",Dic_Data["longitude"])
        print("latitude:", Dic_Data["latitude"])
```

- This listener function is running in another thread and its main function is detecting if a new child is being added to the firebase database, if a child is added, it's called as callback function performing the actions inside it and extracting the longitude and latitude from that new child and put it in a lifo queue in order to be able to access it out of the thread.

```
def on_connect(client, userdata, flags, rc):                # func
    if rc==0:
        print ("Connected to AWS")
    print("Connection returned result: " + str(rc) )
```

- The on_connect function is called after the connection with AWS is established and its function is to make sure that the connection is established successfully.

```
def Distance_Calculation(longitude1,latitude1,longitude2,latitude2):
    lon1 = radians(longitude1)
    lon2 = radians(longitude2)
    lat1 = radians(latitude1)
    lat2 = radians(latitude2)
    # Haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat / 2) ** 2 + cos(lat1) * cos(lat2) * sin(dlon / 2) ** 2

    c = 2 * asin(sqrt(a))

    # Radius of earth in kilometers. Use 3956 for miles
    r = 6371
    return(c * r)
```

- The Distance_Calculation function is called to calculate the distance between the vehicle and each hospital when being called.

```
def button_callback(channel):
    print("Button was pushed!")
    latitude1=q1.get()
    longitude1=q1.get()
    Distances=[]
    for Hospital in Hospitals_db:
        longitude2=Hospital["longitude"]
        latitude2=Hospital["latitude"]
        Distances.append(Distance_Calculation(longitude1,latitude1,long
    Min_distance_pos= Distances.index(min(Distances))+1
    dest= Hospitals_db.get(doc_id=Min_distance_pos)
    print(longitude1,latitude1)
    msg={ "longitude":longitude1,"latitude": latitude1 }
    msg_JSON=json.dumps(msg)
    GPS.publish(dest["Hospital name"],msg_JSON)
    print(dest["Hospital name"])
```

- Whenever a high voltage come the input pin 18 of the Raspberry pi, it detects an accident and enters this function which iterates through the database and calculates the distance and extracts the minimum distance and sends the location to the hospital corresponding to that distance

- Whenever a high voltage come the input pin 18 of the Raspberry pi, it detects an accident and enters this function which iterates through the database and calculates the distance and extracts the minimum distance and sends the location to the hospital corresponding to that distance.

```
cred = credentials.Certificate("gps-data-a987e-firebase-adminsdk-elp1b·

firebase_admin.initialize_app(cred, {'databaseURL': "https://gps-data-a

Hospitals_db = TinyDB('Hospitals_DB.json')

ref2 = db.reference("/a")
ref2.delete()
ref = db.reference("/a").listen(listener)
########################################
GPS= paho.Client()
GPS.on_connect = on_connect   #assigning functions to callbacks
GPS.on_message = on_message
```

- Connecting to the firebase database.
- Connecting to the TinyDB which is already created within the Raspberry pi.
- At the beginning the firebase database is empty from any previous sessions and then set the listener to listen to that database.
- The last three lines are concerned with initializing the MQTT protocol by creating an instance client and assigning the call back functions.

```
awshost = "a2p87hu0j8wq89-ats.iot.us-east-1.amazonaws.com"      # Endp
awsport = 8883                                                 # Port no.
clientId = "RaspberryPi_1"                                     # Thing_
thingName = "RaspberryPi_1"                                    # Thing_
caPath = "/home/pi/GPS/AmazonRootCA1.pem"
certPath = "/home/pi/GPS/certificate.pem.crt"
keyPath = "/home/pi/GPS/private.pem.key"

GPS.tls_set(caPath, certfile=certPath, keyfile=keyPath, cert_reqs=ssl.(
GPS.connect(awshost, awsport, keepalive=60)

GPS.loop_start()
sleep(1)
```

- Setting the parameters for AWS.
- Connecting to AWS with all previous parameters.

- Starting the loop_start method.

```
108 GPIO.setwarnings(False) # Ignore warning for now
109 GPIO.setmode(GPIO.BOARD) # Use physical pin numbering
110 GPIO.setup(18, GPIO.IN, pull_up_down=GPIO.PUD_DOWN) # Set pin 10 to be
111 GPIO.add_event_detect(18,GPIO.RISING,callback=button_callback,bouncetir
112 message = input("Press enter to quit\n\n") # Run until someone presses
113 GPIO.cleanup() # Clean up
114
```

- Setting pin 18 as input pin and assigning the call back function.

# CHAPTER 10
# Future improvement

# 10.1) Implementing Advanced Self-Parking Technology

- **Introduction:**

In the world of connected vehicles, innovation knows no bounds. One groundbreaking feature that has captured the imagination of both drivers and researchers alike is self-parking technology. As vehicles become smarter and more connected through V2X (Vehicle-to-Everything) systems, the integration of parallel parking capabilities using ultrasonic sensors has emerged as a promising avenue for enhancing convenience and safety.

- **4 Ultrasonic Sensors: Pioneering the Parallel Parking Process**

When it comes to self-parking, accurate and reliable sensor technology is at the heart of the operation. In our endeavor to implement a robust self-parking unit, we have turned to ultrasonic sensors as the key component for enabling precise maneuvering and obstacle detection. These sensors utilize sound waves to measure distances and provide real-time feedback on the vehicle's surroundings, ensuring a safe and seamless parking experience.

- **Research and Development:**

Before diving into implementation, extensive research and development were undertaken to identify the optimal sensor configuration for our self-parking unit. By exploring various ultrasonic sensor models and assessing their compatibility with our V2X system, we aimed to strike the perfect balance between accuracy, reliability, and cost-effectiveness.

- **Hardware Integration:**

Once the ultrasonic sensors were carefully selected, the next critical step was their seamless integration into both the vehicle and garage units. Each vehicle unit was equipped with two ultrasonic sensors positioned at the front, while two additional sensors were installed at the rear. This configuration ensured comprehensive coverage and precise measurement of distances from nearby obstacles during the parking process.

- **The Parallel Parking Process:**

When a driver initiates the self-parking feature, the system springs into action. Using the four ultrasonic sensors, it analyzes the surroundings and identifies suitable parallel parking spaces. The sensors continuously emit ultrasonic waves, which bounce back upon encountering an obstacle, allowing the system to calculate the precise distance between the vehicle and any potential obstructions.

Once a parking space is detected, the self-parking unit takes over control of the steering wheel and accelerates the vehicle at a safe speed. The ultrasonic sensors work in harmony, providing real-time feedback to guide the vehicle's movement. By accurately measuring the distance from nearby vehicles or obstacles, the system ensures precise positioning during the parking maneuver.

- **Safety and Collision Avoidance:**

Safety is paramount in any self-parking system, and our implementation is no exception. The four ultrasonic sensors play a pivotal role in collision avoidance. They continuously monitor the vehicle's surroundings, providing timely warnings and initiating necessary corrective actions to prevent any potential accidents. By intelligently interpreting the sensor data, the self-parking unit can identify any imminent risks and respond accordingly, ensuring the safety of the vehicle, its occupants, and the surrounding environment.

- **Benefits:**

The implementation of the self-parking unit offers several key benefits:

a) Improved user experience: The self-parking feature will provide drivers with a seamless and user-friendly parking experience, enhancing customer satisfaction.

b) Reduced parking time: The automated parallel parking technology will minimize the time required for parking, leading to increased efficiency and productivity for drivers.

c) Enhanced safety: By utilizing advanced sensors and algorithms, the self-parking unit will help mitigate the risk of collisions and improve overall road safety.

d) Environmental impact: Efficient parking will contribute to reduced fuel consumption and emissions, aligning with our organization's commitment to sustainability.

## 10.2) Using more powerful toolkit for computer vision applications instead of Raspberry Pi

The Jetson Nano is a powerful toolkit for machine learning projects. It is small, affordable, and easy to use. It is powered by NVIDIA, a leading manufacturer of graphics processing units (GPUs). GPUs are designed to accelerate the processing of mathematical operations, which are essential for machine learning tasks.

The Raspberry Pi is another popular toolkit for machine learning projects. However, it is not as powerful as the Jetson Nano. The Raspberry Pi uses a general-purpose processor, which is not as well-suited for machine learning tasks as a GPU.

## 10.3) Implement another model for License plate recognition

Using OCR to read license plates

Optical character recognition (OCR) is a technology that can be used to read text from images. This technology can be used to read license plates, which are typically made up of a combination of letters and numbers.

To read a license plate using OCR, an image of the license plate is first captured. This image is then processed by an OCR engine, which identifies the letters and numbers in the image. The OCR engine then outputs the text that it has identified.

## 10.4) Implement a model to detect the empty slots in the garage

To implement a model to detect empty slots of the garage using a camera, we need to follow these steps:

1. Collect a dataset of images of empty and occupied slots. This dataset can be collected by taking pictures of the garage at different times of day.
2. Train a machine learning model on the dataset. The machine learning model can be trained to identify empty and occupied slots by using a technique called image classification.
3. Deploy the model on a camera. The model can be deployed on a camera so that it can be used to detect empty and occupied slots in real time.

## 10.5) Implement a model to make object detections using YOLO

We can use YOLO (You Only Look Once) to detect pedestrians and decide if the person is close to the car or not. YOLO is a popular object detection algorithm that is known for its speed and accuracy.

To implement a model using YOLO, you will need to follow these steps:

1. Collect a dataset of images of pedestrians and cars. This dataset can be collected by taking pictures of pedestrians and cars in different settings.
2. Train a YOLO model on the dataset. The YOLO model can be trained using a variety of software frameworks, such as Darknet and TensorFlow.
3. Deploy the YOLO model on a camera. The YOLO model can be deployed on a camera so that it can be used to detect pedestrians and cars in real time.