Electronics & Communication Department,
Faculty of Engineering,
Cairo University.

Graduation Project
2023

Graduation Project

Auto Emergency Braking System

Presented by:

Ahmed Saeed Abdelraouf Ahmed

Amira Sherif Kamel Mohamed

Doaa Mohamed Shafiy Mubarak

Hassan Sayed Hassan Sayed

Mennatullah Gamal Eid Mohammed

Omneiia Atef Mohy Eldein Hussien

Under supervision of:

Prof. Omar Mohamed Nasr

A Thesis Presented for the Degree of
Bachelor of Electronics and Communication

SWIFT ACT
YOU DREAM . WE ACT

Acknowledgment:

We gained a lot of technical and communication skills needed for the working environment while working on our graduation project with Swift-Act. It increased our knowledge and our working experience. However, none of this would have been possible without the help of many great individuals to whom we would like to express our sincere gratitude and appreciation.

First, we would like to show our special gratitude and thanks to ***prof. Omar Nasr*** for their involvement in every phase throughout the project. Moreover, we would like to express our sincere gratitude for their eagerness to help us reach our goal and deliver a high-quality final product. All of this shows that we were fortunate to have hir as our advisor.

Second, we would like to thank our sponsors, **Swift-Act**.

Next, we would like to thank ***Eng. Ashraf Amgad and Eng. Amr Abd-Elnabi*** for their continuous encouragement and motivation. In addition, we would like to thank them for their help and guidance with the whole project steps.

In addition, we want to thank the teaching staff of Faculty of Engineering, Cairo University who taught us everything we need to be able to finish this project. We are grateful that we met some of the greatest minds in the faculty; we learned a lot from them on, both academic and personal level.

Finally, yet important, we want to send a very special thanks to our families who supported us through thick, thin, and tolerated us in stressful times. Without them and their continuous encouragement, we would have never reached the place where we are now.

At the end, we would like to thank other unnamed who helped us in various ways to have a good experience and to finish our project.

ABSTRACT

The automotive industry is changing rapidly to fulfill user requirements. One of these requirements, is the increase in gadgets inside the car. To achieve such a requirement, now a day's accidents are increasing more and more, so safety has acquired a priority. Improper usage of brakes is also one of the problems for accident. The project idea is to improve the safety parameters regarding to brakes. Sudden recognition of any object in front panics the driver, at that situation normal drivers fail to use brakes correctly this leads accident, taking the driver reaction time into account we will try to assist the driver by doing this we can avoid accidents and hence can increase safety.

In our project, we model a scenario, that have cars and pedestrians moving on the road. This scenario is managed by two sensors and a camera.

In addition, the major cause of death in our world is car accidents. Nearly around 1.5 billion people die due to car accidents and the majority are happening just due to simple factors that are drowsiness and distraction of drivers. Most people travel long distances without any sleep and using mobile phones while driving results in the issue of tiredness and as result drowsiness and distraction. This can be avoided just by alerting the driver when there is any such case of occurrence. So, we are proposing a system that can alert the driver using an alarm or auto-brake when the driver falls asleep. The aim of this project is to build a driver distraction and drowsiness detection system that will detect whether a person’s eyes are closed or not and distracted or not.

The two flows are then directed to an application that acts as the Frame, which is responsible for prioritizing the flows and choosing which flow to pass first.

The frames are then routed to the host machine, which would process the camera frames detecting the objects in it. The processed camera frames are then sent to the Simulink to take actions based on the passed data.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1. Introduction

Vehicle technology has increased rapidly in recent years, particularly in relation to braking systems and sensing systems. Automatic braking may be a safety technology that automatically activates the vehicle's brake, to the point, when necessary. Systems can vary from pre-charging brakes to slowing the vehicle to reduce damage. Nowadays, some advanced and updated systems completely take over and stop the vehicle before a collision happen. The precise capabilities of their car's automatic braking system. Regardless of a vehicle's autonomous technologies, drivers should remain conscious of their surroundings and maintain control in the least times. the automatic braking or brake assist is an integral component of crash avoidance technologies, including front crash prevention systems, back over prevention systems, and cross-traffic alert systems. Each automaker may have a special name for such technologies, but the rock bottom line is that the brake assist is supposed to attenuate accidents.

Additional hardware that allows brake pressure to be increased above pedal demand as well as to be reduced, combined with additional software control algorithms and sensors allow traction control (TC), electronic brake fore distribution (EBD), brake assist (BAS) and electronic stability control (ESC) functions to be added. In parallel to the development of braking technologies, sensors have been developed that can detect physical obstacles, other vehicles or pedestrians around the vehicle. Many luxury, mid-size and small cars in Europe, and in Japan even very small cars (Daihatsu Move), are now fitted with an adaptive cruise control (ACC) system that is capable of measuring and maintaining a driver-preset headway to the vehicle ahead by automatic modulation of the engine control, and if required, automatically applying brakes up to a maximum deceleration of 0.3g Tas per ISO standard) If no vehicle is ahead, the vehicle maintains the desired "set-speed", ACC can be ordered as an option for new vehicles: At least three heavy truck manufacturers offer this feature on their vehicles, Theoretically, a vehicle equipped with modern braking technology and adaptive cruise control is equipped with all of the necessary hardware to allow a simple (braking only no steering) collision: avoidance system that would be capable of detecting when a collision is likely to occur and applying emergency braking to avoid it. Collision mitigation systems are already on the market, providing limited braking capability Integrated safety systems based on these principles can be broadly divided into three categories

Collision avoidance - Sensors defect a potential collision and take action to avoid it entirely, taking control away from the driver, In the context of braking this is likely to include applying emergency braking sufficiently early that the vehicle can be brought to a standstill before a collision occurs. In future, this could also include steering actions independent of the driver This category is likely to have the highest potential benefits but is the highest risk approach because a false activation of the system has the potential to increase the risk to another road

Collision mitigation braking systems (CMBS)-Sensors detect a potential collision but take no immediate action to avoid it. Once the sensing system has detected that the collision has become inevitable regardless of braking or steering actions then emergency braking is automatically applied (independent of driver action) to reduce the collision speed, and hence injury severity, of the collision. This type of system has lower potential benefits but is lower risk because it will not take control away from the driver until a point very close to a collision where the sensing system is likely to be more reliable. Such a system may also trigger actions related to secondary safety such as the pre-arming or optimization of restraints.

Forward collision warning- Sensors detect a potential collision and take action to warn the driver. This is the least risky option since false detection of a collision only has impacts on the driver's reaction to, and perception of, the system. This type of system could also be used to optimize restraints. This type of system has been sold on some EU vehicles since 1999. The original focus of this project was to assess the technical requirements, costs and benefits relating to collision mitigation braking systems in their current form, however, as the project has developed the scope has expanded to consider, the potential benefits that could arise from automatic emergency braking systems (AEBS) that include avoidance capabilities.

1.1. Problem Definition

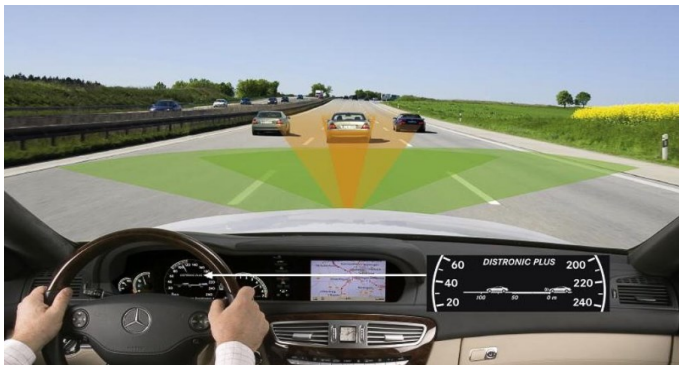


Figure 1: Driver View.

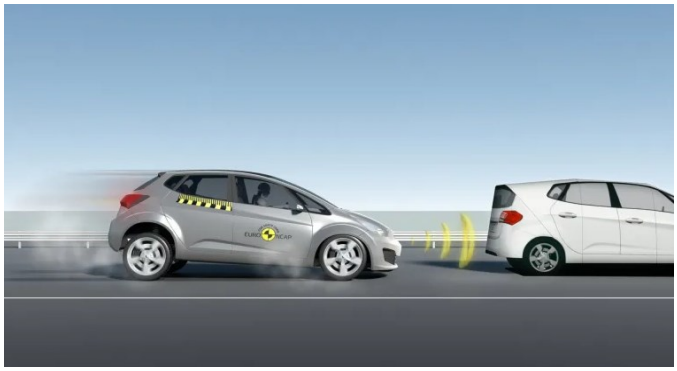


Figure 2: Car brakes suddenly

The objective of this project is to design the automatic braking system in order to avoid the accident. To develop a safety vehicle braking system using RADAR sensor and to design a vehicle with less human attention to the driving. This project is necessary to be attached to every vehicle. Mainly it is used when drive the vehicles in night time. Mostly the accident occurred in the night time due to long travel the driver may get tired. So the driver may hit the front side vehicle or road side trees. By using this project, the vehicle is stopped by automatic braking system. So we can avoid the accident.

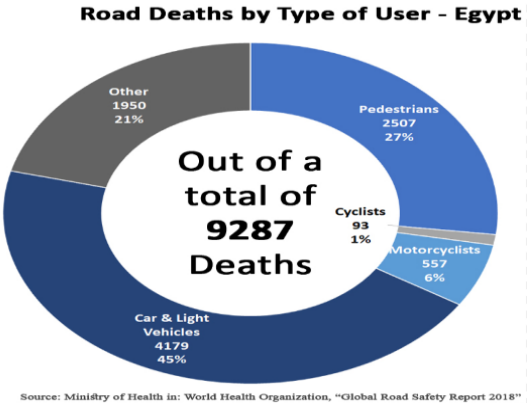


Figure 3: Accidents on roads statistics in Egypt

Therefore, the importance of an ABS inside the car is important.

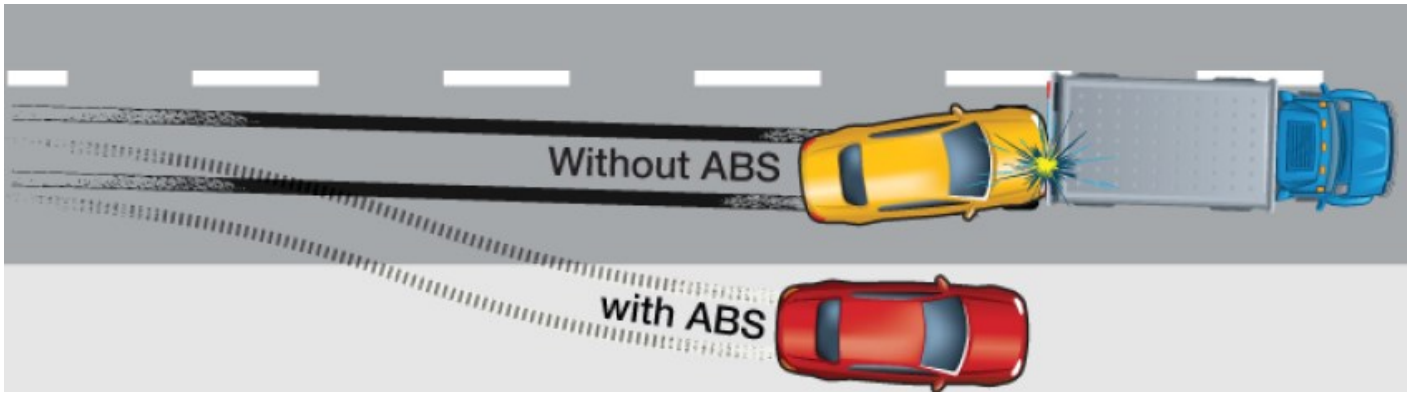


Figure 4: With and without ABS

1.2. Proposed Solution

We propose the usage of Radars and Communication protocols "UART", "CAN" & "SPI" as a communication method between ECUs in the vehicle. The automotive requirements for ADAS are 20-100 Mbps bandwidth per camera and a latency of <1ms. These requirements can be fulfilled in our proposed solution.

1.3. Project Description

Active brake safety technology is mainly composed of three modules, including control module (ECU), distance measurement module and brake module. The core of the ranging module includes microwave radar, face recognition technology and video system, etc., which can provide safe, accurate, real-time images and road condition information on the road ahead.

First use radar to measure the distance to the vehicle or obstacle in front, and then use the data analysis module to compare the measured distance with the warning distance and safety distance. If the distance is less than the threshold of warning distance, a buzzer will be heard by the driver and surrounding cars. When the distance is less than the safety distance, even if the driver does not have time to step on the brake pedal, the system will start to see if the other lanes are free to go to if so, the car will turn right or left based on the best lane to move through. It starts by checking the right lane then the left lane accordingly, if this isn't an option and there are obstacles in other lanes the system will make the car brake automatically, thus escorting the safe travel.

-Measured active brake

After referring to the relevant experimental standards of international safety testing agencies and combining some common accident cases on the road, we divided the active emergency braking test into two major items: the anti-collision braking test and the pedestrian detection test, and the vehicle AEB system is considered separately The ability to recognize the vehicle ahead and pedestrians crossing the road, and the ability to control the speed of the vehicle.

In addition, many brands or models currently have high-end driving assistance systems, such as Tesla's Autopilot, etc. These systems integrate multiple driving assistance functions including AEB, and can reach

the second-level autonomous driving level in accordance with relevant industry standards. It should be noted that these systems can only be used as an aid and supplement to the driver's operation during driving. In order to achieve the uniformity of test standards and ensure fairness and impartiality, we only test the Active Emergency Braking System (AEB) and will not turn on other driving assistance systems.

-Anti-collision braking test

In this test session, we use an inflatable model as the "target vehicle" in a rear-end collision, that is, the party being rear-end collision. The test vehicle will drive from directly behind the model at a speed of 10-30km/h to investigate the whether the car can recognize obstacles ahead, issue warnings and apply braking.

-Pedestrian detection test

In this test session, we set up three scenes: 1. The dummy is standing still in front of the vehicle's driving route to check whether the vehicle can recognize stationary pedestrians and issue warnings and apply braking; 2. If the first scene passes successfully, it will enter the test of the second scene, that is, the dummy props cross the vehicle movement route horizontally, simulate the scene of pedestrian crossing the road, and examine the detection range and reaction speed of the system.

If the test car successfully passes the tests of the first two scenes, then the third additional scene will be tested, the "ghost probe" scene. The difficulty of this scene is relatively high, so it serves as a "plus point" for the entire test. The test speed of all scenes is 30km/h, each scene is tested 5 times, and the evaluation conclusion is based on whether the system can avoid or effectively reduce the collision.

1.4. Vision and Mission

1.4.1. Vision

Our vision is to contribute with our project in the automotive industry and to keep up with the global shift toward the automotive industry field.

1.4.2. Mission

Our mission is to produce an outstanding project that gives better results and shows superiority over other projects made in this topic.

1.5. Domain of Application

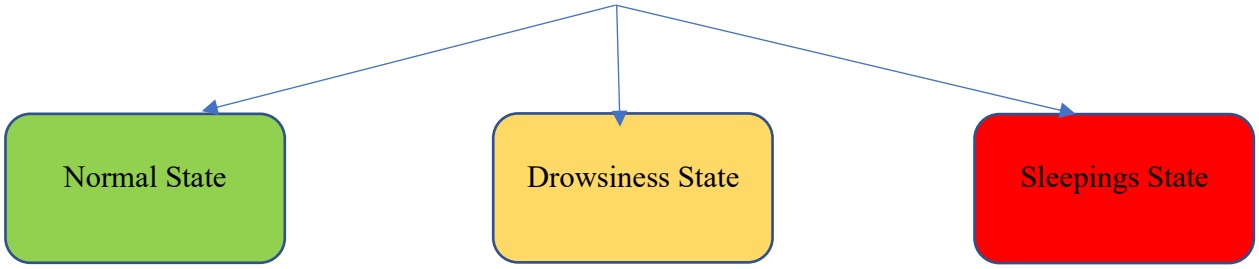
The domain of our application is the automotive industry domain. The automotive industry is a huge one with a vast amount of money moving. Moreover, in-vehicle networking is a challenging topic nowadays as will be discussed later in our thesis; that is why we chose this specific topic for our graduation project.

1.6. Summary of Approach

Our project has passed through several phases throughout this academic year. First, we started researching in the automotive industry from the embedded development point of view and studying the current market and its limitations, along with that we studied the SPI Protocol, UART Protocol, CAN Protocol, Radars, and the amendments designed for the automotive industry. The following step was to find papers on the

calculation of collision time based on mathematical equations, then we begin with small scenarios to implement our algorithm and try sending and receiving by UART protocol from the discovery board then we split the tasks to be handled by RTOS, therefore we have 5 tasks managed by RTOS which are: sending UART, receive UART, define region of operation, make decision and finally a CAN task which handles the data sent from the Raspberry Pi which is connected to a USB camera that scans the driver's face by rate 2 frames per second based on this data the algorithm starts from this points:

We defined three region of operations:



If we are in the normal state, we will be checking the environment surrounding the car continuously until it detects an object in the range we will be specifying in the algorithm chapter of which the ranges will be dividing them into 3 ranges:

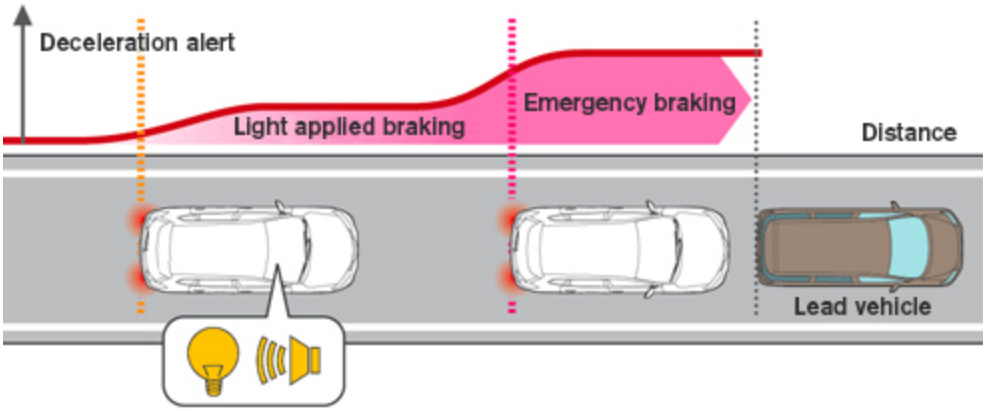


Figure 5: Regions of operation.

The four ranges are 1: Normal state, 2: Warning state, 3: Partial brake & 4: Full brake force.

Normally, we are in region 1 until a car or a human appears in the region of warning a buzzer will be heard to alarm the driver, hereby we enter the 2nd region, if a car or a human in the region of emergency braking the brake will be maximum to force the car to stop immediately.

If we are in the warning state, a buzzer will be heard to alarm the driver if detected drowsy.

If we are in the sleeping state, the car will choose the nearest lane to the pavement to park and the system will stop until the driver starts it all over again.

The CAN task is given higher priority than other tasks.

Chapter 2

Methodology

2. Methodology

2.1. About PreScan Program:

Prescan refers to a software platform used for the development, testing, and validation of advanced driver-assistance systems (ADAS) and autonomous vehicles. It is a simulation and virtual prototyping tool that allows engineers and researchers to create virtual environments, vehicles, and sensors to simulate real-world driving scenarios.

Prescan enables automotive companies and researchers to test their ADAS and autonomous driving algorithms and technologies in a virtual environment before implementing them in real-world vehicles. It provides a realistic simulation of various driving conditions, such as urban, rural, and highway scenarios, and allows users to assess the performance, safety, and reliability of their systems.

The software incorporates accurate physics-based modelling, sensor simulations (such as LiDAR, radar, and cameras), and realistic vehicle dynamics to replicate real-world driving conditions. It can simulate complex interactions between vehicles, pedestrians, and other objects in the environment, enabling users to evaluate the behavior of their autonomous systems and fine-tune them accordingly.

By utilizing Prescan, automotive manufacturers and researchers can save time and costs associated with physical testing and can iterate and optimize their algorithms in a controlled and repeatable virtual environment. It aids in the development and validation of ADAS features like collision avoidance, lane keeping, adaptive cruise control, and autonomous driving capabilities.

Prescan is one of several simulation tools available in the market for the development of autonomous vehicles and plays a crucial role in advancing the technology and ensuring its safety and reliability.

2.2. Sensors and actuators

2.2.1. Lidar

Lidar, which stands for Light Detection and Ranging, is a remote sensing technology that uses lasers to measure distances and create precise 3D models of objects and environments. Lidar sensors emit laser pulses that bounce off objects and return to the sensor, allowing it to calculate the distance and shape of the object. Lidar technology has become increasingly popular in recent years due to its ability to provide highly accurate and detailed information about the physical world.

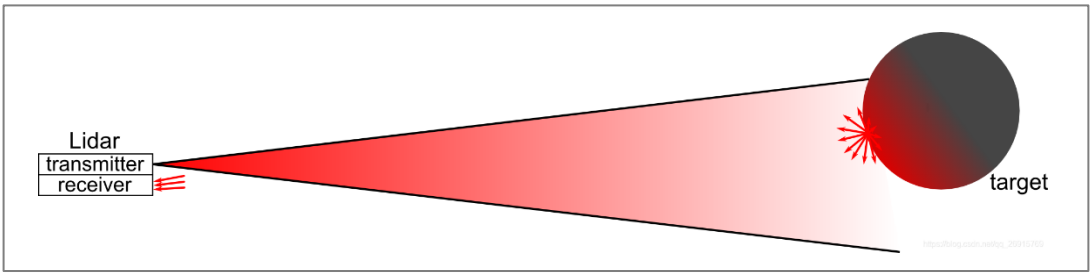


Figure 6: Operating Principle of Lidar

2.2.1.1. Lidar Applications

Lidar technology has a wide range of applications across various fields, including automotive, forestry, archaeology, and meteorology. Here are some more details on the applications of Lidar:

1. Autonomous Vehicles: Lidar sensors are a critical component in the development of autonomous vehicles. They help these vehicles "see" their surroundings by emitting pulses of light and measuring the time it takes for the light to bounce back. This information is then used to create a 3D map of the environment, which allows the vehicle's onboard computer to navigate safely.
2. Geographical Surveying: Lidar technology is used extensively in geographical surveying to create high-resolution 3D maps of the terrain. This information is used to study the topography of the land, map floodplains, and identify areas of potential geological hazards.
3. Forestry Management: Lidar sensors are used in forestry management to estimate forest biomass, map forest structure, and measure tree height. This information is critical in managing forests sustainably and can help predict forest fires and potential damage to infrastructure caused by fallen trees.
4. Archaeology: Lidar technology is used in archaeology to create high-resolution 3D models of archaeological sites. This information is used to study the landscape, identify previously undiscovered sites, and preserve cultural heritage.
5. Meteorology: Lidar sensors are used in meteorology to measure the atmospheric composition, including the concentration of gases, aerosols, and clouds. This information is used to study climate change, atmospheric chemistry, and weather forecasting.
6. Industrial and Manufacturing: Lidar technology is used in industrial and manufacturing settings to inspect and verify the quality of products and materials. It can also be used in quality control and automation processes, such as robot navigation and precision measurements

2.2.1.2. How Lidar works

Lidar sensors work by emitting laser pulses that bounce off objects and return to the sensor. The sensor can then calculate the distance to the object based on the time it takes for the pulse to return. Lidar sensors can also measure the intensity of the returning pulse, which can provide information about the reflectivity of the object. By emitting laser pulses at multiple angles, Lidar sensors can create a detailed 3D map of an object or environment.

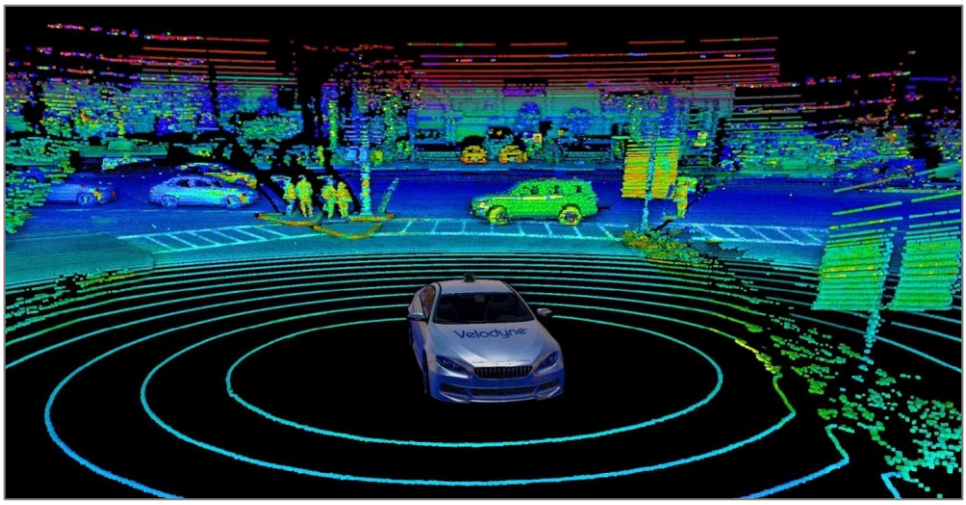


Figure 7: Lidar 3D map

2.2.1.3. Lidar Eye Safety

Eye safety is an important consideration when working with Lidar sensors. The intense light emitted by Lidar sensors can cause damage to the eyes; if proper safety precautions are not taken. To protect against eye damage, Lidar sensors are typically equipped with safety features such as automatic shut-off mechanisms and warning systems that alert operators when the sensor is operating.

2.2.1.4. Lidar Sensor in Autonomous Vehicles

Lidar sensors are an essential component to autonomous vehicles, allowing them to accurately perceive their surroundings and navigate roads safely. In self-driving cars, Lidar sensors are typically mounted on the roof or front grille of the vehicle and emit laser pulses in all directions. These pulses bounce off nearby objects and return to the sensor, allowing the car's onboard computer to create a detailed 3D map of its surroundings.

2.2.1.5. Lidar Resolution

Lidar sensors are capable of producing high-resolution 3D maps of objects and environments. The resolution of a Lidar sensor is determined by the number of laser pulses emitted by the sensor per second, as well as the size and spacing of the laser beams. Higher resolution Lidar sensors are capable of producing more detailed 3D maps, but may also be more expensive and require more processing power to analyze the data. Lidar resolution is an important consideration when choosing a sensor for a specific application, as it can impact the accuracy and precision of the resulting 3D models.

2.2.2. Radar

Radar, which stands for Radio Detection and Ranging, is a remote sensing technology that uses radio waves to detect and locate objects. Radar was first developed in the early 20th century and has since become an important tool in a range of applications, from military and aviation to weather forecasting

and traffic monitoring.

Radar system uses an antenna to transmit radio signals. It refers to electronic equipment that detects the presence of objects by using reflected electromagnetic energy. Under some conditions, radar system can measure the direction, height, distance, course, and speed of these objects. The frequency of electromagnetic energy used for radar is unaffected by darkness and also penetrates fog and clouds. This permits radar systems to determine the position of airplanes, ships, or other obstacles that are invisible to the naked eye because of distance, darkness, or weather.

Figure 8: RADAR

2.2.2.1. Radar Applications

Radar technology has a wide range of applications across various fields, including military, aviation, meteorology, maritime navigation, and automotive safety. Here are some more details on the applications of Radar technology:

1. Military Applications: Radar technology plays a crucial role in military applications, including early warning systems, ground-based surveillance, air defense, missile guidance, and target tracking. Military Radar systems can detect and track aircraft, missiles, ships, and ground vehicles, providing valuable intelligence to military commanders.
2. Aviation Applications: In aviation, Radar is used for air traffic control, weather monitoring, and aircraft navigation. Air traffic control systems use Radar to track the position of aircraft and ensure safe separation between planes. Weather Radar systems detect and track precipitation and severe weather events, allowing pilots to adjust their flight paths accordingly.
3. Meteorology Applications: Weather Radar systems are used extensively in meteorology to detect and track precipitation, thunderstorms, and other severe weather events. Doppler Radar, in particular, is used to measure the speed and direction of wind, which is critical for predicting the path of storms.
4. Maritime Navigation Applications: Radar technology is widely used in maritime navigation for ship detection, collision avoidance, and marine search and rescue operations. Marine Radars can detect ships, icebergs, and other objects in the water, providing valuable information to ships' crews and helping to prevent collisions.
5. Automotive Safety Applications: In recent years, Radar technology has become increasingly important in automotive safety systems, such as adaptive cruise control, blind spot detection,

and automatic emergency braking. Automotive Radars use radio waves to detect nearby objects and help cars navigate safely on the road.

6. Industrial and Security Applications: Radar technology is also used in a range of industrial and security applications, such as level measurement, perimeter security, and ground-penetrating Radar. Ground-penetrating Radar, in particular, is used to detect buried objects, such as pipes and cables, and is also used in archaeological and geological surveys.

2.2.2.2. How Radar work

Radar sensors are devices that change microwave echo signals into electrical signals through wireless sensing technology, which allows them to detect motion and determine an object's position, shape, motion characteristics, and trajectory. Unlike other sensors, radar sensors are not affected by light and darkness and can even detect obstructions like glass or "see" through walls. Compared to ultrasound sensors, radar sensors are capable of detecting longer distances and pose no risk to humans or animals.

One of the main advantages of radar sensors over other types of sensors is their ability to detect motion and velocity by analyzing an object's Doppler effect, which is the change in wave frequency. This allows a radar sensor to compute both an object's speed and direction. Additionally, by using multi-channel sensors, radar sensors can observe an object's movement from multiple perspectives, allowing for more accurate and complex measurements of movement. These different perspectives, along with previously collected measurements, are analyzed to determine an object's complex movements.

2.2.2.3. Radar system components

Radar systems are composed of several critical components that work together to enable the system to function effectively.

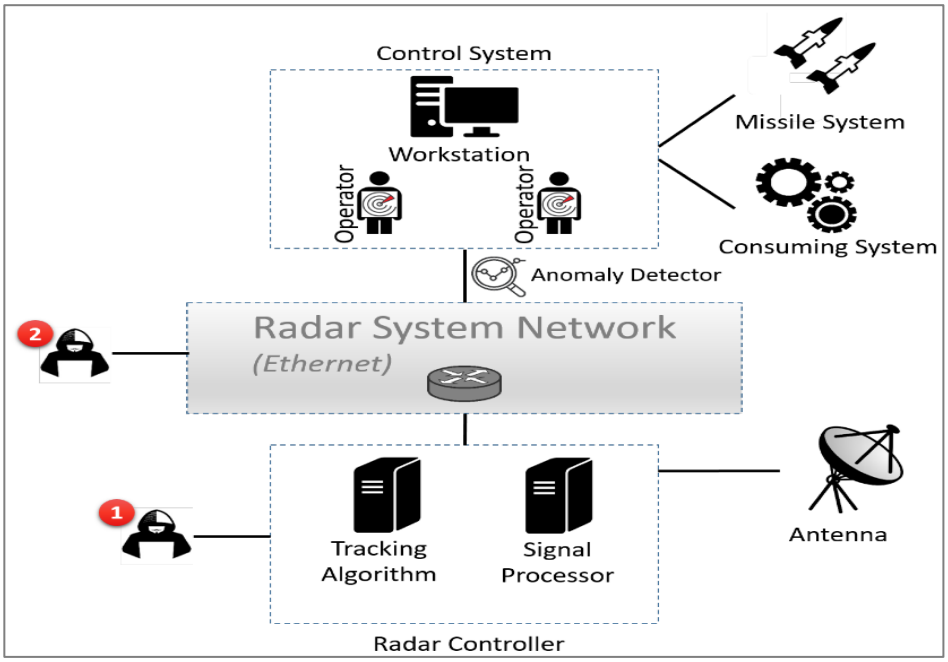


Figure 9: An example of a typical radar system architecture.

Here are some more details on the components of a Radar system:

- Antenna: The antenna is the component of a Radar system that emits and receives radio waves. It is usually a parabolic dish that is designed to focus the transmitted signal in a specific direction and to collect the reflected signal from the target.
- Transmitter: The transmitter generates the radio waves that are emitted by the antenna. It converts electrical energy into radio frequency energy and sends it to the antenna for transmission.
- Receiver: The receiver is responsible for detecting the reflected radio waves from the target. It amplifies the received signal and converts it into an electrical signal that can be processed by the Radar system.
- Duplexer: The duplexer is a component that allows the same antenna to be used for both transmitting and receiving signals. It ensures that the transmitted signal does not interfere with the received signal.
- Signal Processor: The signal processor is the component that analyzes the received signal and extracts information about the target, such as its position, speed, and direction. It also filters out unwanted noise and interference from the signal.
- Display: The display is the component that presents the information about the target to the operator in a usable format. It can be a screen, a series of lights, or an audio signal, depending on the type of Radar system.
- Power Supply: The power supply is responsible for providing the necessary electrical power to the Radar system components, including the transmitter, receiver, and signal processor.
- Cooling System: Some Radar systems generate a significant amount of heat during operation. A cooling system is required to maintain the temperature within the specified range and prevent damage to the Radar system components.

2.2.2.4. Radar sensors Types in autonomous vehicles

Radar is an essential technology for autonomous vehicles, providing accurate information about the vehicle's surroundings in real-time. There are several types of radar used in autonomous vehicles, each with its own advantages and disadvantages. Here are some of the most common types:

1. Short-Range Radar (SRR): SRR is used for close-range detection and operates at a frequency between 24 and 26 GHz. It provides accurate information about the distance and speed of nearby objects, making it ideal for use in parking assistance and collision avoidance systems.
2. Medium-Range Radar (MRR): MRR operates at a frequency between 76 and 81 GHz and is used to detect objects at a range of up to 150 meters. It provides information about the distance, speed, and direction of objects, making it ideal for use in adaptive cruise control and lane departure warning systems.
3. Long-Range Radar (LRR): LRR operates at a frequency between 77 and 81 GHz and is used to detect objects at a range of up to 250 meters. It provides accurate information about the distance, speed, and direction of objects, making it ideal for use in high-speed driving scenarios and highway

driving.

4. Solid-State Radar: Solid-state radar uses electronic components instead of traditional mechanical parts to emit and receive radar signals. It provides high-resolution imaging capabilities and is ideal for use in advanced driver assistance systems (ADAS) and autonomous vehicles.
5. FMCW Radar: Frequency-Modulated Continuous Wave (FMCW) radar uses a continuous wave signal that is modulated in frequency. It provides high-precision distance measurements and is ideal for use in advanced safety systems, such as collision avoidance and emergency braking.

LIDAR and RADAR Fusion: Some autonomous vehicles use a combination of LIDAR and RADAR technologies to provide a more comprehensive view of the vehicle's surroundings. This fusion of technologies provides accurate distance and positioning information and is ideal for use in complex driving scenarios

2.2.2.5. Automotive Radar

Automotive RADARs as core sensor (range, speed) of driver assistance systems: long range (LRR) for Adaptive Cruise Control, medium range (MRR) for cross traffic alert and lane change assist, short-range (SRR) for parking aid, obstacle/pedestrian detection

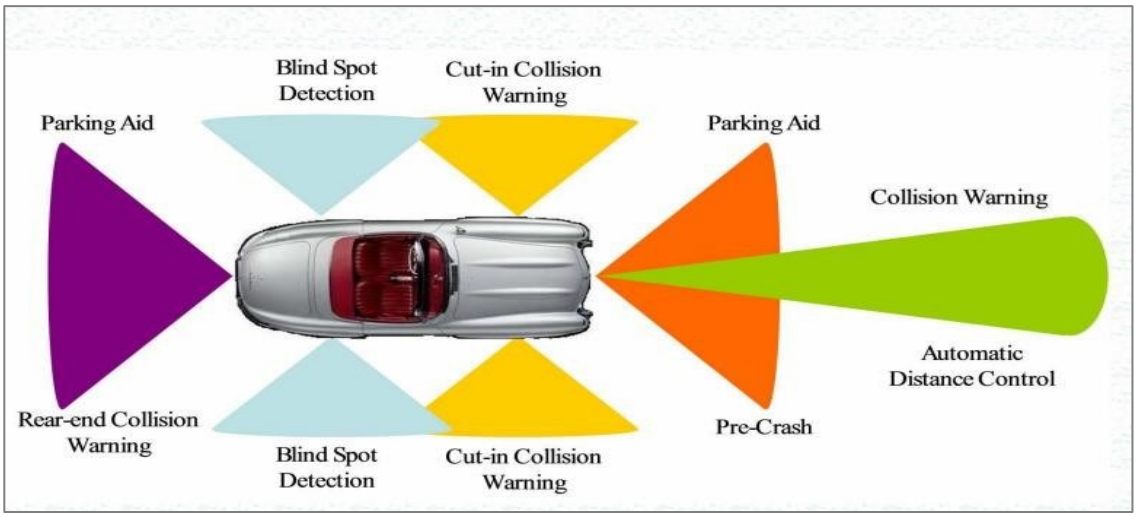


Figure 10: Automotive Radar.

- To other sensing technology RADAR is robust in harsh environments (bad light, badweather, extreme temperatures)
- Multiple RADAR channels required for additional angular information
- Data fusion in the digital domain with other on-board sensors

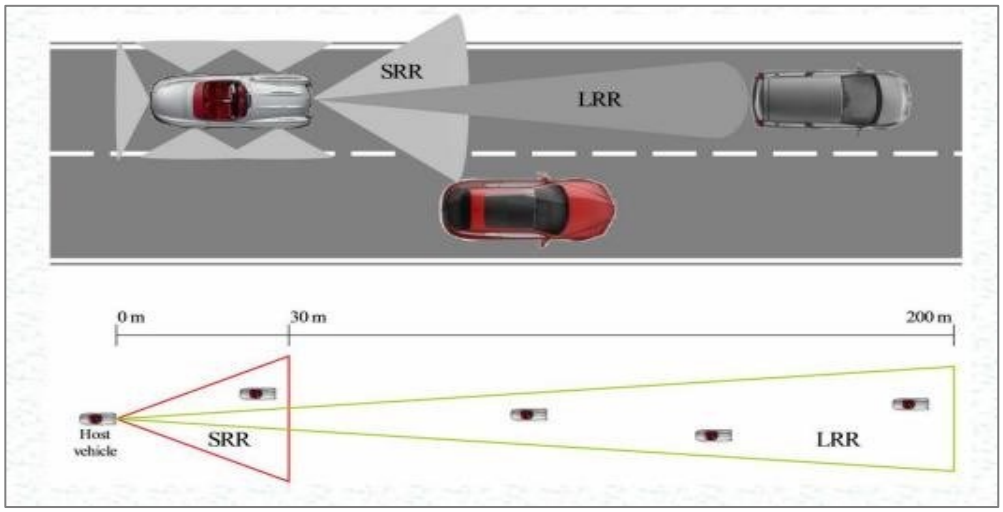


Figure 11: Automotive Radar

Main signal processing functions in automotive RADARs:

- Range estimation
- Doppler frequency estimation
- Long Range Radar (LRR)
- Observation area39:

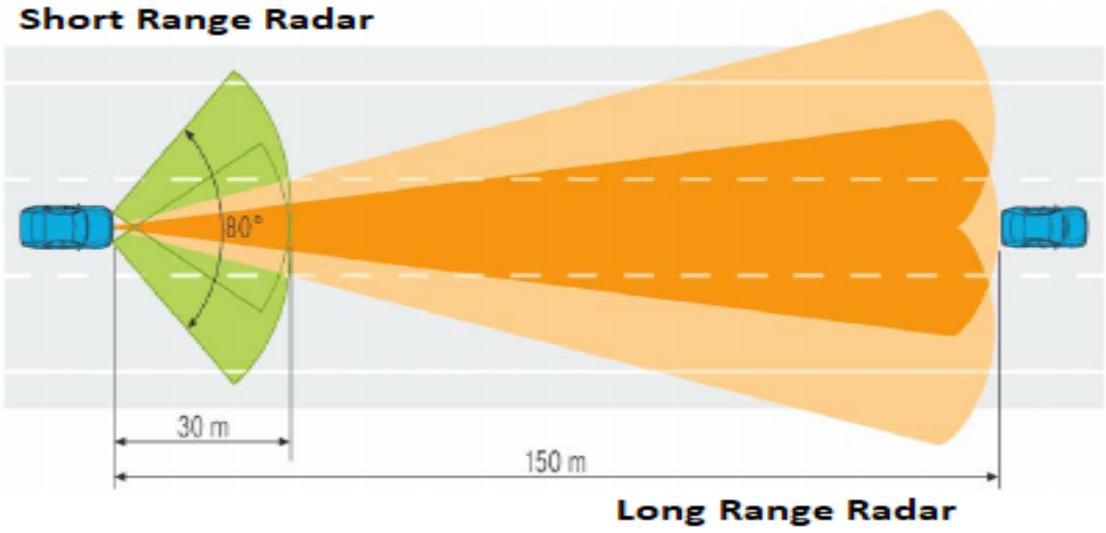


Figure 12: Long Range Radar (LRR)

Requirements for LRR RADAR:
Functionalities: Autonomous Cruise Control (ACC) Collision warning.

2.2.2.6. Doppler Radar

A Doppler radar [4] is a specialized radar that uses the Doppler effect to produce velocity dataabout objects at a distance. It does this by bouncing a microwave signal off a desired target and analyzing how the object's motion has altered the frequency of the returned signal. radar adapted to acquire detection information including velocity information of a target by transmitting and receivinga radio wave.

In known in-vehicle radars, detection is performed over a detection range in front of a vehicle,and the position and the velocity of a target such as another vehicle existing in the detection range aremeasured. In FM-CW radars, a transmit signal is transmitted which alternately has an up-modulation period in which frequency gradually increases and a down modulation period in which the frequency gradually decreases. If a reflected signal from a target is received, the position and the velocity of thetarget with respect to the position and the velocity of the vehicle are determined on the basis of a frequency spectrum of a beat signal due to the difference in frequency between the transmitted signaland the received signal.

Parameter	Value
Velocity resolution	$\Delta v_r = 2.25$ km/h
Range resolution	$\Delta R = 1$ m
Unambiguous radial velocity	$v_{max} = 250$ km/h
Maximum range	$R_{max} = 200$ m
Short measurement time	$T_{CPI} = 10$ ms

Figure 13: Requirements for LRR Radar

In the FM-CW radars described above, the relative velocity (the “Doppler velocity”) is measured on the basis of the frequency shift of the beat signal appearing in the frequency spectrum due to the Doppler effect, and the position of the target are determined every predetermined measurement interval and the moving velocity (the “differential velocity”) of the target is determinedfrom a change in the position. Each of these two types of velocity information has advantages and disadvantages, and use of only one of these two types of velocity information can cause a problem in the determination of the velocity.

The “Doppler velocity”, the “differential velocity”, and the “overall velocity” are described below.

2.2.2.7. Doppler Velocity

The Doppler velocity Vdop is determined as follows.

Vdop=c·fd/2fo(1) where c: velocity of light

fd: Doppler shift frequency fo: transmission frequency

Differential Velocity

The differential velocity Vdiff is determined as follows.

Vdiff=(dn−db)/T(2) where

dn: distance to a target obtained in a present measurement db: distance to a target obtained in a previous measurement

T: measurement intervals at which steps S1 to S10 shown in Figure 17 are performed repeatedly, that is, which the beam is scanned in the azimuth direction over the detection range once in each interval T.

2.2.2.8. Overall Velocity

The overall velocity refers to the relative velocity obtained this time by an overall determination on the basis of the relative velocity determined (output) in the previous measurement, and the Doppler velocity and the differential velocity described above.

The difference between the Doppler velocity and the differential velocity is small, the Doppler velocity is employed as the relative velocity of the target. Alternatively, when the difference between the Doppler velocity and the differential velocity is great, a value employed for an immediately previous relative velocity is employed as a current value of the relative velocity, while when the difference between the Doppler velocity and the differential velocity is small, the Doppler velocity is employed as the relative velocity of the target.

2.2.3. Lidar vs Radar

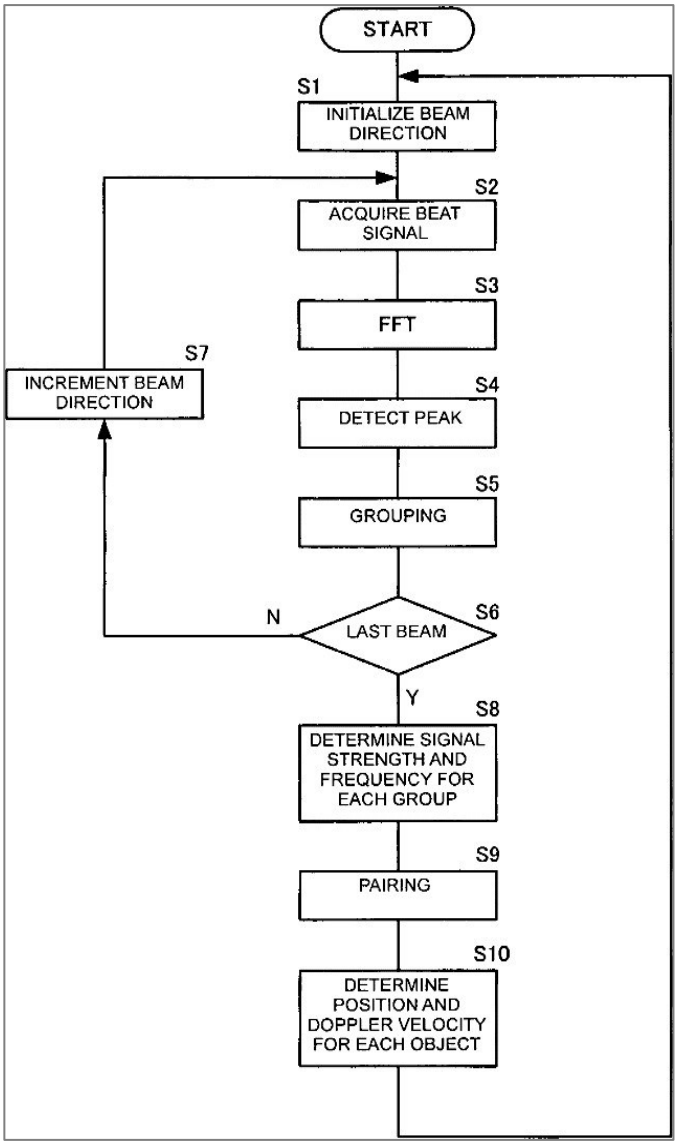


Figure 14: Time measurement

	Radar	***Lidar***
Wavelength	Radio waves (30 cm - 3mm)	Infrared light (micrometer range)
Applications	Aircraft anti-collision systems, air traffic control, radar astronomy	3D mapping, laser altimetry, contour mapping
Working principle	Radio waves are transmitted from a rotating or fixed antenna and the time of flight of the reflected signal is measured	An infrared laser beam is pointed at a surface and the time it takes for the laser to return to its source is measured

Resolution	Several meters at a distance of 100 meters	A few centimeters at a distance of 100 meters
Advantages	Can detect objects at long distances and through fog or clouds	Compact solution that enables a high level of accuracy for 3D mapping
Disadvantages	Lateral resolution is limited by the size of the antenna	Limited range and affected by environmental conditions such as dust and rain
Usage in autonomous vehicles	Used for object detection and collision avoidance	Used for mapping and localization

Table1: Lidar vs Radar.

2.3. Realistic case study for the suitable sensor

2.3.1. Mercedes Active Safety Braking system

Mercedes-Benz offers a radar-based assistance system in the shape of DISTRONIC PLUS (optional), which also incorporates Brake Assist PLUS.

DISTRONIC PLUS proximity control operates at speeds of between 0 and 200 km/h: it keeps the car a set distance behind the vehicle in front, applies the brakes as required and can even bring the car to a complete halt, depending on the traffic situation. If the gap to the vehicle in front narrows too quickly, the system gives the driver an audible warning and, as soon as this first warning signal sounds, automatically calculates the brake pressure required to prevent a collision in this situation.

This technology helps the driver to gauge the level of risk and, in combination with Brake Assist PLUS, makes the calculated brake boosting force available instantly, even if the driver does not press the brake pedal forcefully enough. Brake Assist PLUS allows controlled, targeted braking and, if necessary, increases the braking force right up to the point at which an emergency stop is performed, depending on the road speed and the distance to the vehicle in front.

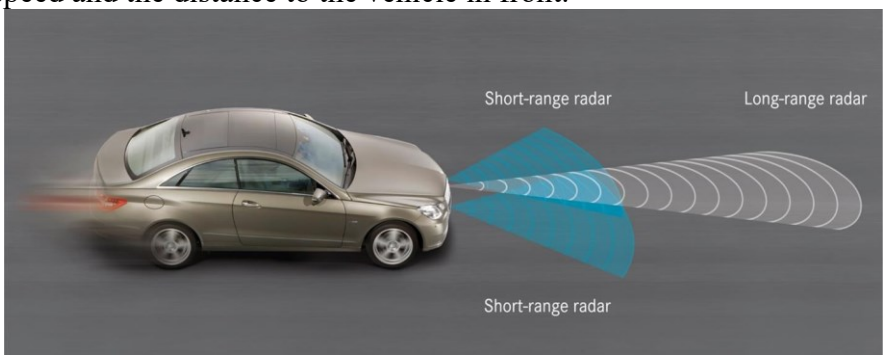


Figure 15: Short-range and long-range radar

When a potential accident situation is recognized, two wide-angle short-range sensors with an 80-degree beam width and a range of around 30 meters, located behind the front bumper, and a long- range

radar with a range of 200 meters, located in the radiator grille, are called upon to offer assistance.

In addition, the sensor system now also has medium-range detection capability, allowing monitoring of the area up to around 60 meters ahead of the car with a 60-degree beam width.

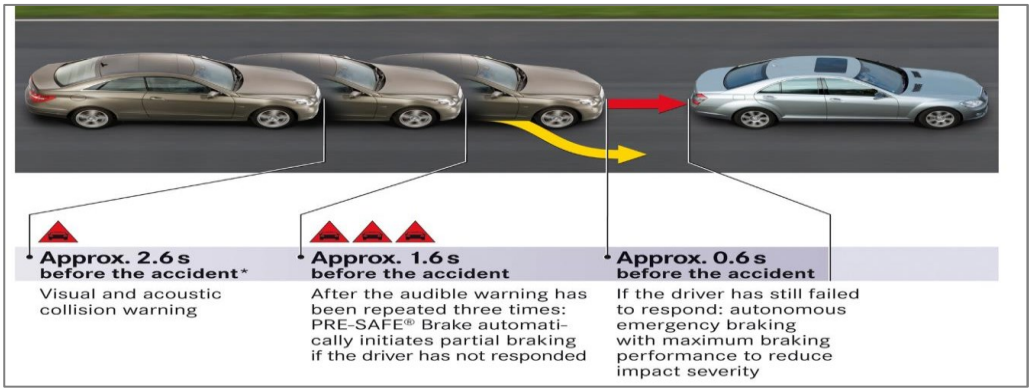


Figure 16: Time calculated by the system until the impact where the relative speed remains unchanged.

2.4. PreScan

2.4.1. About PreScan Program:

PreScan refers to a software platform used for the development, testing, and validation of advanced driver-assistance systems (ADAS) and autonomous vehicles. It is a simulation and virtual prototyping tool that allows engineers and researchers to create virtual environments, vehicles, and sensors to simulate real-world driving scenarios.

PreScan enables automotive companies and researchers to test their ADAS and autonomous driving algorithms and technologies in a virtual environment before implementing them in real-world vehicles. It provides a realistic simulation of various driving conditions, such as urban, rural, and highway scenarios, and allows users to assess the performance, safety, and reliability of their systems.

The software incorporates accurate physics-based modelling, sensor simulations (such as LiDAR, radar, and cameras), and realistic vehicle dynamics to replicate real-world driving conditions. It can simulate complex interactions between vehicles, pedestrians, and other objects in the environment, enabling users to evaluate the behavior of their autonomous systems and fine-tune them accordingly.

By utilizing PreScan, automotive manufacturers and researchers can save time and costs associated with physical testing and can iterate and optimize their algorithms in a controlled and repeatable virtual environment. It aids in the development and validation of ADAS features like collision avoidance, lane keeping, adaptive cruise control, and autonomous driving capabilities.

PreScan is one of several simulation tools available in the market for the development of autonomous vehicles and plays a crucial role in advancing the technology and ensuring its safety and reliability.

2.4.2. Radar sensor in PreScan

1. A RADAR system uses an antenna to transmit radio signals.
2. Models' atmospheric attenuation as function of frequency and rain type.
3. Optionally provides an externally steerable (from Simulink) scan pattern.

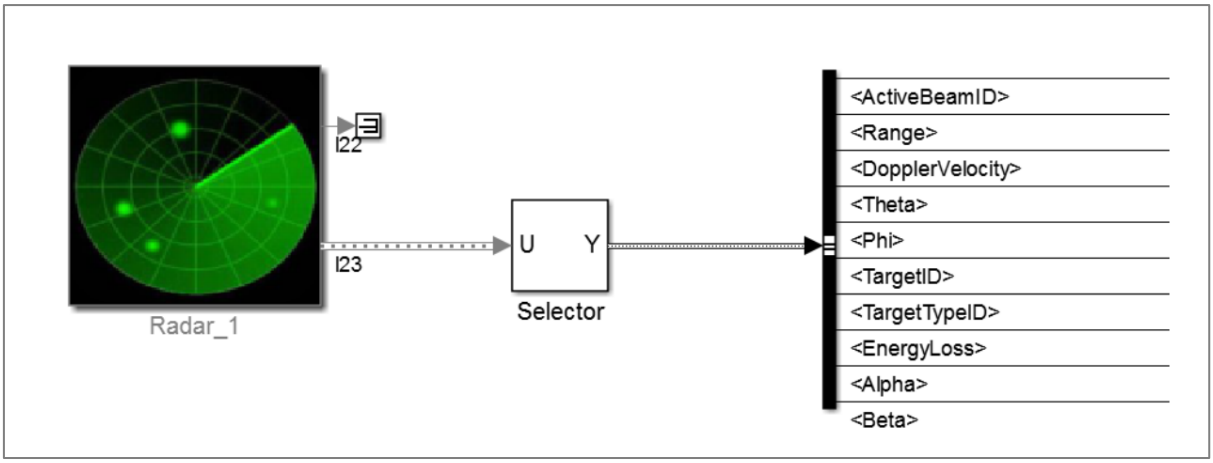


Figure 17: Radar sensor representation in Simulink.

The Radar sensor's block has two output ports. The first port writes out the number of detected objects. The second port contains a vector of output buses. The vector size is the maximum detectable object number of the sensor. Each output bus contains the following signals:

Signal name	Description
ActiveBeamID[-]	ID of the beam in the current simulation time step. Value is 0 when there's no detection.
Range[m]	Range at which the target object has been detected.
DopplerVelocity [ms-1]	Velocity of target point, relative to the sensor, along the line-of-sight between sensor and target point.
DopplerVelocityX/Y/Z [ms-1]	Velocity of target point, relative to the sensor, along the line-of-sight between sensor and target point, decomposed into X,Y,Z of the sensor's coordinate system.
Theta[deg]	Azimuth angle in the sensor coordinate system at which the target is detected.
Phi[deg]	Elevation angle in the sensor coordinate system at which the target is detected.
TargetID[-]	Numerical ID of the detected target.
TargetTypeID[-]	The Type ID of the detected object.
EnergyLoss[dB]	Ratio received power / transmitted power.
Alpha[deg]	Azimuthal incidence angle of the Radar beam on the target object.
Beta[deg]	Elevation incidence angle of the Radar beam on the target object.

Figure 18: Radar signals description.

- In position tab:

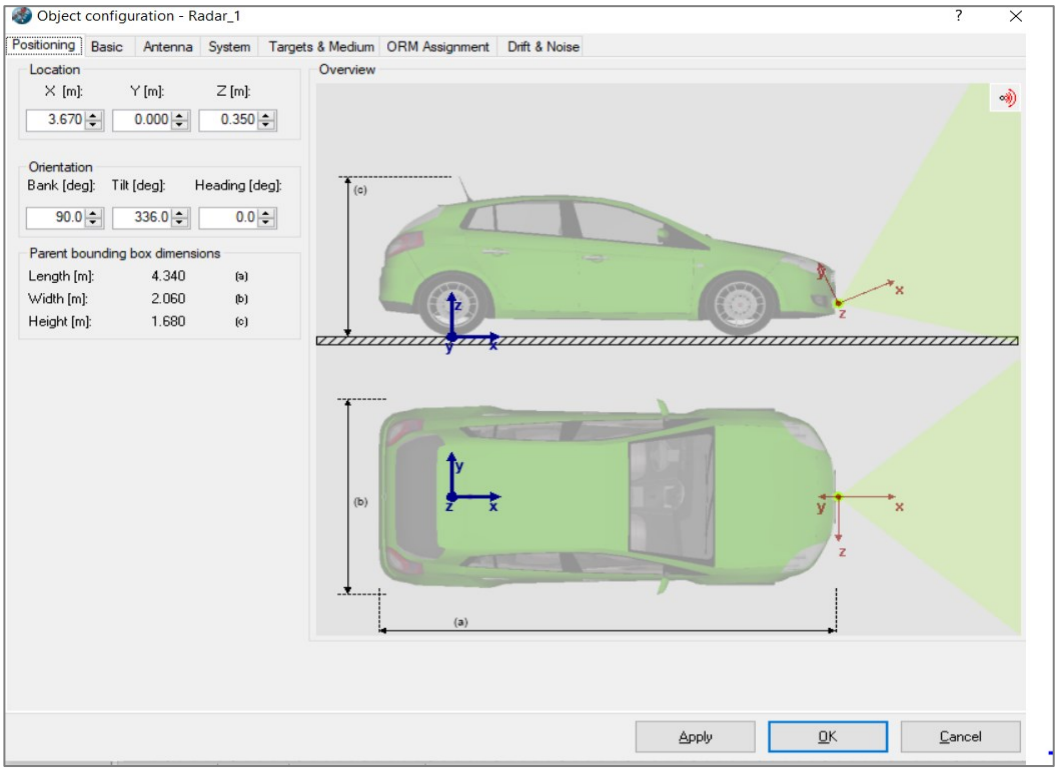


Figure 19: PreScan radar position tab

- **Location:**

  This is where we can place the location of the radar normally in x, y, z coordinates.
- **Orientation:**

  It is divided into three parts. The bank angle determines the rotation angle of the radar itself. So, if you set it to 90, you'll find that the Z-axis is where the Y-axis used to be. Secondly, the tilt determines the elevation angle, and from experiments, it seems that the best value to set it is to -24. Thirdly, the azimuth determines the azimuth angle, and the best value is to set it to 0, so that it's in the middle and covers the whole range evenly.
- **Parent bounding box size:**

  It tells you the size of the radar itself, how much space it takes up, and we don't really care about it.

- In Basic tab:

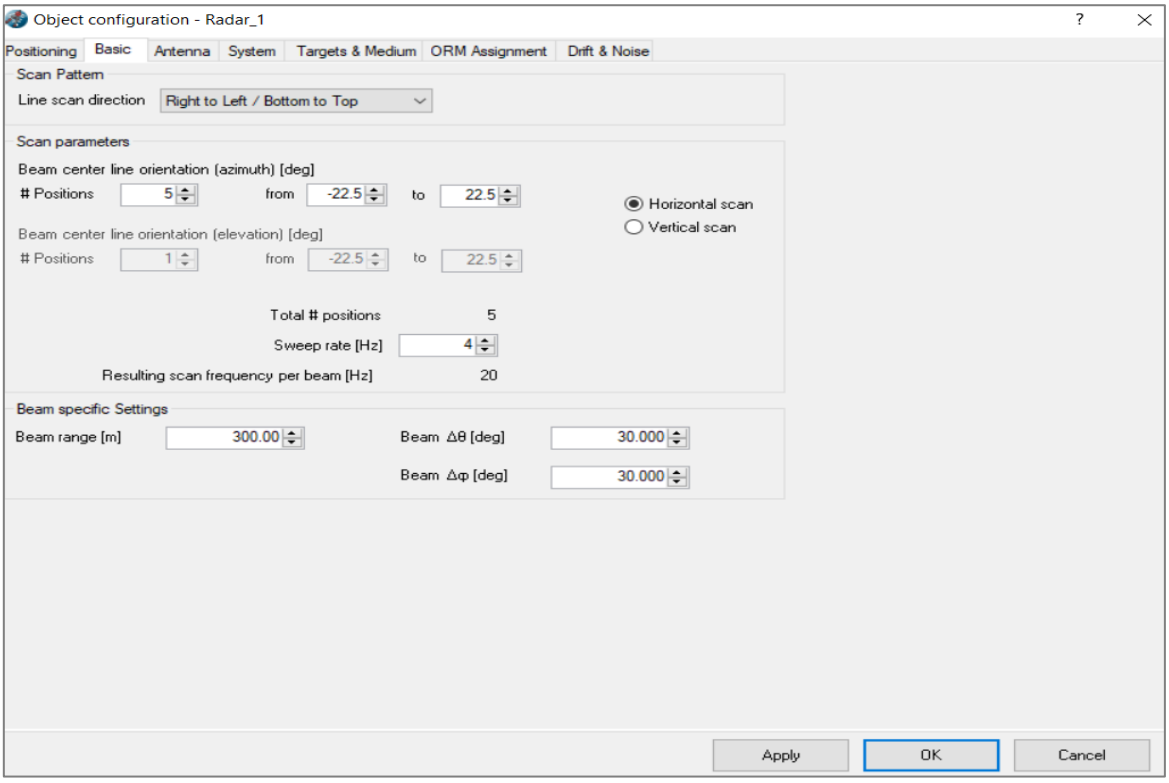


Figure 20: PreScan Radar Basic Tap

The Scan Pattern can be set to either Left to Right/Top to Bottom or Right to Left/Bottom to Top. The Scan parameters field includes information such as:

- Beam center line orientation in Azimuth and Elevation: is the angle measured from the first position of the beam center line to the last position of the beam centerline. Note that if only one position is initialized the beam center line is oriented with a zero degree.
- Number of positions in Azimuth and Elevation: these are integer values that set the number of positions of the radar center line beam. In each position a reading is taken. Based on the sweep rate, the resulting scan frequency per beam is calculated. Note that the radar has only one beam that scans through a number of positions. The scanning can be done in either horizontal or vertical. The Beam Specific Settings field includes information such as:
    - Beam range: the range of the beam.
    - Beam Δθ: the angles in degrees to define the azimuth beam field of view.
    - Beam Δφ: the angles in degrees to define the elevation beam field of view

  This is a very important task. Here, we are determining the scanning directions, and I believe it's best to set it from right to left/bottom to top. This is because we want to first, detect what's underneath like cars, individuals, and animals.

  As we mentioned earlier, the range of the radar is divided into several beams. Here, we will divide each

beam into half. The sweep rate is the rate at which, we set the scanning frequency for one beam which is equal the total number of beams multiplied by the sweep rate. For example, if we have 11 beams and set the sweep rate to 4, we will find that it scans each beam 44 times per second.

First, we need to perform horizontal scanning, not vertical. Secondly, we need to determine the number of beams from the number of positions. Thirdly, we need to determine the total range of the radar.

Below, we set the properties of each beam, and they are all the same. Beam range determines the range as a distance.

- **System Tab:**

The maximum number of object to output:

Suppose more than one target is seen by a single beam. The parameter max objects to output limits the number of data sets to be generated. Selection is made based on the nearest distance considerations (however, the datasets are not sorted by distance).

Based on the above explanation, it means that within the radar range, there are several beams and for each beam, I manually specify the number of objects that it can detect. The radar will detect the closest objects to it within that specified distance, and any objects beyond that distance cannot be detected in that particular beam, as previously mentioned.

System Resolution fields: **resolution cells** are the volumes in space that contribute to the echo received by the TIS at any one instant: within a single resolution cell all objects detected are clustered into a single quantified reading. Resolution cells exist in the range direction, in the azimuth angle and the elevation angle. *Figure 21* shows the effect of having a limited resolution in range and azimuth (situation in elevation is not depicted here, but is similar to the situation in azimuth): although the 2 targets (4/R4 & 1/R4) are physically at different locations, they both will be assigned to range R4. Of course, they will have different detection angles in azimuth (in this case -θ1 and θ3).

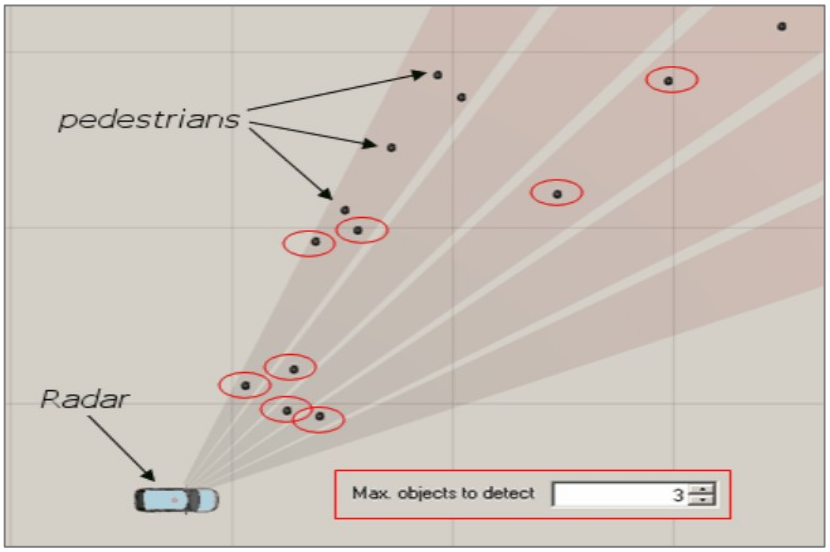


Figure 21: Pedestrians of radar

The allowable range for the values to be entered for the angular azimuth cell size is from 0 to 1.0 * (minimum of the beam width in azimuth of all beams), for the angular elevation cell size it can range

from 0 to 1.0 *(minimum of the beam height in elevation of all beams), and for the range cell size it is from 0 to 1.0 *(minimum of the range of all beams). By default, they all are set to 0 indicating there is no quantization behavior at all.

Here, you are literally dividing the entire range of the radar into a 2D plane. Like in *figure 22*, you can divide the x-axis for a defined range with equal distances. This distance is placed in the range cell size field, which represents R1, R2, R3, ...

Then, you can divide the y-axis for angles, specifying an origin, for example, in the center, and before and after it, other specified angles. Our task is to determine the angle of the azimuth and the elevation, with the range determined by the azimuth and elevation angular cell size.

So when I detect an object, I say that it's in R4, for example, and the angle is like theta -1, which is the first cell after the origin on the right side of the radar. And, of course, because I'm looking from above, this is the range of the azimuth angle.

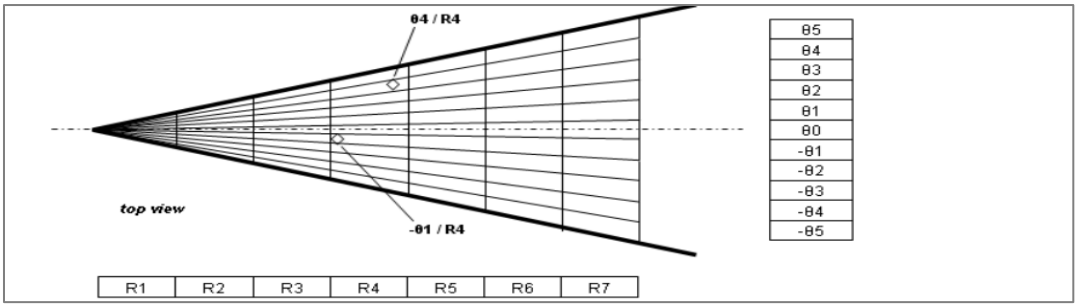


Figure 22: Radar system Resolution

- Radar in Simulink:

  Using "bus selector" component, we can show the data from radar in Simulink:

  - select the signal identifiers using this bus selector
  - put displays in each signal which will display screenshots of a number of things that we previously defined in PreScan.

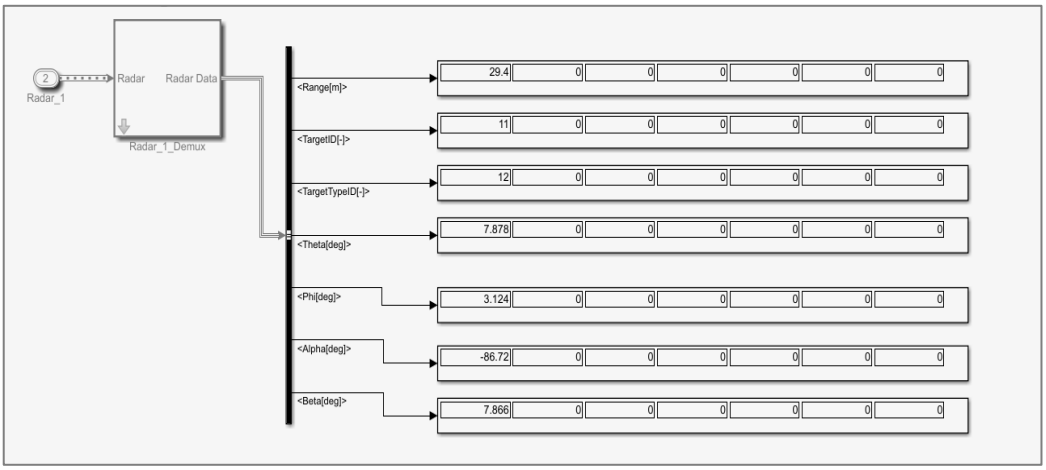


Figure 23: Radar in Simulink

Chapter 3

Simulink Blocks

3. Simulink Blocks

3.1. S Function:

A S-function is a computer language description of a Simulink block written in MATLAB®, C, C++ S-functions use a special calling syntax called the S-function API that enables you to interact with the Simulink engine.

This interaction is very similar to the interaction that takes place between the engine and built-in Simulink blocks.

Inside this block shown below. We Put our Algorithm which is translated to C code to brake the car.

```
%initializes the legacy code data structure
def = legacy_code('initialize');
%specifying the sfunction name of the block
def.SFunctionName = 'func_name';
%specifying the function return type, name and inputs
def.OutputFcnSpec = 'double y1 = func_name(double u1,double u2)';
%specifying the source files to be linked with the sfunction block
def.SourceFiles = {'main.c'};
%specifying the header files to be linked with the sfunction block
def.HeaderFiles = {'add_lib.h'};
%generates a S-function source file as specified by the Legacy Code Tool
%data structure, def.
legacy_code('sfcn_cmex_generate', def);
%Compiles and links the S-function generated by the Legacy Code Tool
%based on the data structure, def
legacy_code('compile', def);
```

There are three outputs of this block (Brake, Control and Warning).

- The brake varies from 0 to (max brake force) 150bar.
- The control signal takes values between 0 to 1. Zero means the driver has the control over the brake and one means the software of the car takes the control.
- The Warning signal takes values between 0 to 1. Zero means no warning and one means it's warning.

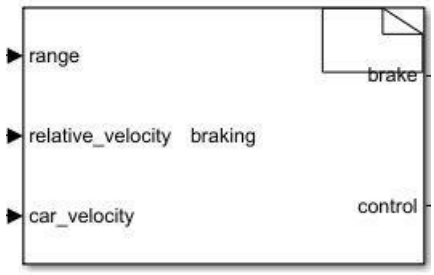


Figure 24 :S Function Block

The S-function block is a block in Simulink where you can add a function written in C language to perform some logic on its inputs and affect its outputs.

In the S-function we deal with inputs and outputs as pointers. You only specify the inputs and the outputs in the bock configurations and the MATLAB creates the function for you with the parameters you defined as constant parameters and the outputs as real ones.

The code in the S-function block is treated like any other block in the Simulink, so the code is executing from the beginning every time its sampled like how Simulink working in MATLAB. It's important to know that to make sure that your code is only depending on the current state (Current inputs) and cannot be dependent on the last state.

3.1.1. Algorithm in Simulink:

When we create an S-function block there is a (main.c) is created automatically in this file we can write and compile our code as shown in *figure 25*.

```
*control = 1;
*warning = 0;
if (*car_velocity > 25);
else
{

   float alt_relative_velocity = -1 * (*relative_velocity);
   float critical_time = calculate_critical_time(alt_relative_velocity); // calculate critical time
   float collosion_time = (*range)/(alt_relative_velocity);


   if ((alt_relative_velocity > 0) &&(0 < collosion_time) && (collosion_time <= critical_time))
   {
       *control = 2;
       *brake = 150;
       *warning = 1;
   }
   else if((alt_relative_velocity > 0) &&(0 < collosion_time) && (collosion_time <= critical_time + 0.3)){
      *warning = 1;
      *brake = 0.6 * 150; //40% brake

   }
   else if((alt_relative_velocity > 0) &&(0 < collosion_time) && (collosion_time <= critical_time + 3)){
       *warning = 1;
   }
   else if(*range>5){
       *control = 1;
       *warning = 0;
       *brake = 0;
```

Figure 25 : Algorithm in S-function

3.2. Controller:

As shown in *figure 26* if the TTC is less than or equal TCritical the control equals 1 to make the driver removes his legs of the accelerator by sending zero to the accelerator.

Take a process on control parameter in MUX Block: Throttle and Control.

To Determine the Throttle value.

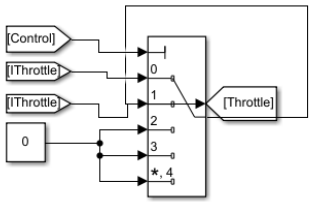


Figure 26 : Obtain the function of controller

3.3. Serial configuration Block:

Used to set the configuration of Serial Port that determine the speed of port that used in UART configuration, number of bits, parity and stop bit.

Figure 27 : Serial configuration Block

3.4. Rader Blocks:

3.4.1. Radar Blocks Process:

There are six Radars detecting two values long range Radar (LRR) & Short-range Radar (SRR).

Collect data and concatenate it into frame by using Concatenate frame Block that contain data about SRR Range, LRR Range, SRR Relative speed, LRR, Relative speed, Desired Velocity.

Send this data to Serial Send Comm5.

3.4.2. Car Dynamics:

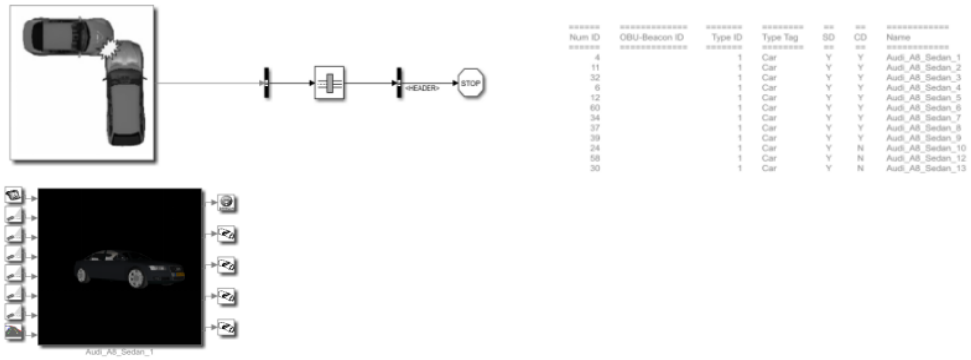


Figure 28: Car dynamic

Collision Detection Block: can be used to detect collisions between PreScan components. For Collision Detection the actual component shape is taken into account, so it is not a bounding box approach. The size of the vector is the maximum number of detected collisions specified in the General Settings dialog. Each result bus signal contains the two colliding objects IDs, if there is a corresponding collision detected. **Overview of all Objects Active in Simulation**: Text containing all experiment objects (which participate in the simulation) characteristics: Numerical ID, Type ID, Type Tag, SensorDetectable (SD), Collision Detectable (CD), and Name.

Bus Creator: This block creates a bus signal from its inputs.

Signal Conversion: Convert a signal to a new type without altering signal values.

If the incoming signal is not a bus, the 'Signal copy' option creates a contiguous segment of memory to store a copy of an input signal. If the incoming signal is a bus, the 'Signal copy' option outputs a copy of the incoming bus.

Signal Conversion: Convert a signal to a new type without altering signal values.

If the incoming signal is not a bus, the 'Signal copy' option creates a contiguous segment of memory to store a copy of an input signal. If the incoming signal is a bus, the 'Signal copy' option outputs a copy of the incoming

bus.

Bus Selector: This block accepts a bus as input which can be created from a Bus Creator, Bus Selector or a block that defines its output using a bus object. The left list box shows the signals in the input bus. Use the Select button to select the output signals. The right list box shows the selections. Use the Up, Down, or Remove button to reorder the selections. Check ' Output as bus ' to output a single bus signal.

In this block shown in *figure 29*, we can control our car movements. In our experiment we send brake and car speed from S-function block to this block.

We use from the block: -

- The throttle.
- V0 to control the speed
- The Brake

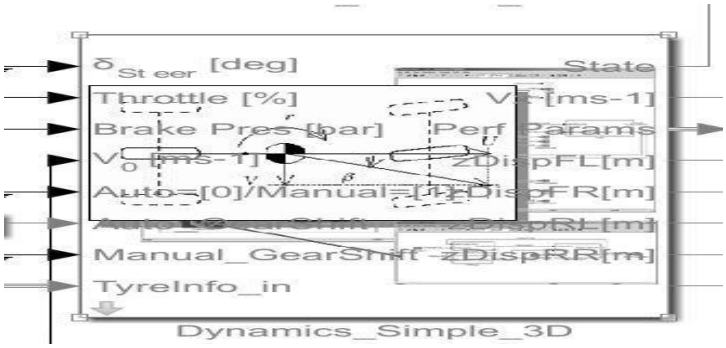


Figure 29 :Car Dynamics Block

The velocity we use ranges from 0 to 25 m/s.

The brake force we use ranges between 0 to 150bar.

3.4.3. Host Car configurations:

We use Audi_A8_Sedan with the following specifications shown in *figure 30*.
The maximum brake force of our module host car is 150bar.

3.4.4. Constrains:

The algorithm is designed to work only within the speeds lower than or equal to 90 km/h. If the speed is greater than 90 km/h, the driver has the full control as auto-braking is such a speed may cause a disaster.

Brake force:

the brake force is a key parameter that determines the level of braking applied to the wheels of a vehicle. ABS

is designed to prevent the wheels from locking up during braking, allowing the driver to maintain steering control and reduce the stopping distance. The brake force varies from one car to another. In our experiment using Audi_A8_Sedan the max force is equal to 150bar.

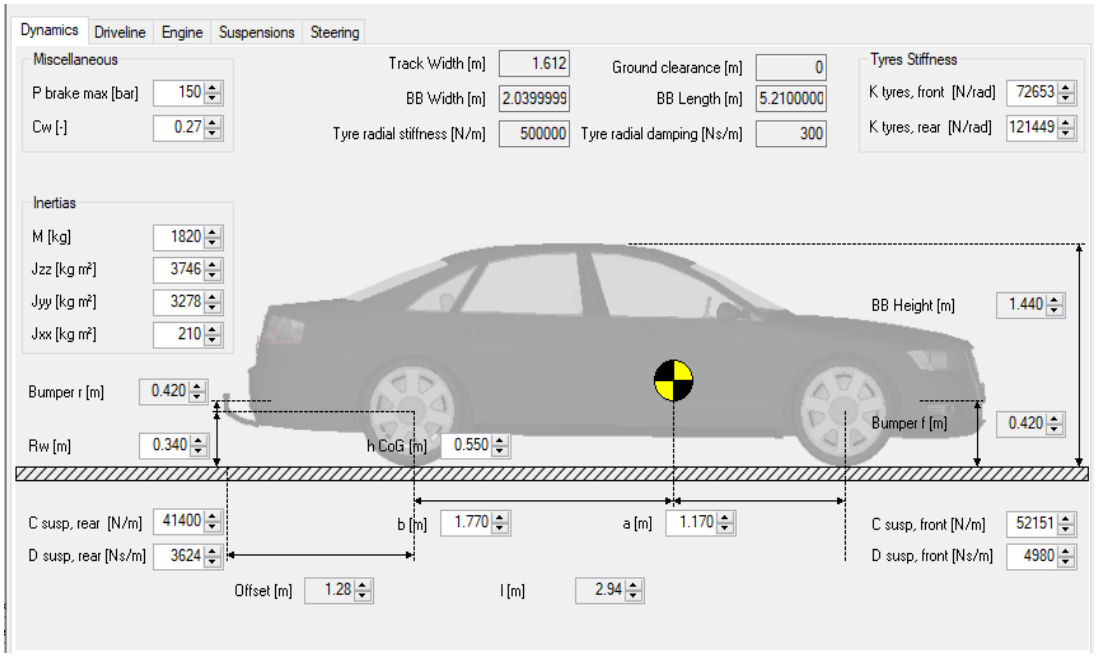


Figure 30: Car Configurations

3.4.5. Steering angle:

The steering angle is defined as the angle between the front of the vehicle and the steered wheel direction as shown in *figure 32*. The steering system has a maximum (minimum) steering angle of +30 (-30) degree. And by using it can divide the road into n number of equal distanced virtual zones as shown in *figure 31*.

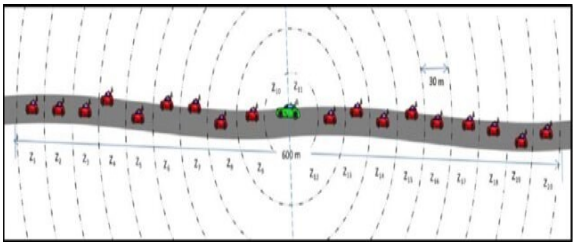


Figure 31: Virtual zone division for short-term. misbehavior detection.

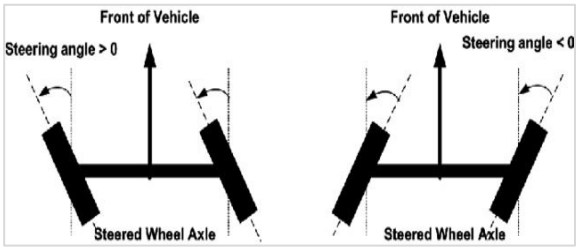


Figure 32 :Definition of Steering Angle

3.5. Lane Keeping Sensor block:

The Lane Marker Sensor provides an input port to the PreScan compilation sheet. Within the subsystem of an

actor, a Lane Marker Sensor Data block can be found that outputs data for use in your Simulink model. The output is always a Bus signal containing all scanned data. To identify the contents of the bus, select 'Update Diagram' from the context menu or the Edit menu of the compilation sheet. In order to extract information from the bus, a ‘Bus Selector’ from the ‘Simulink/Signal Routing’ library can be used.

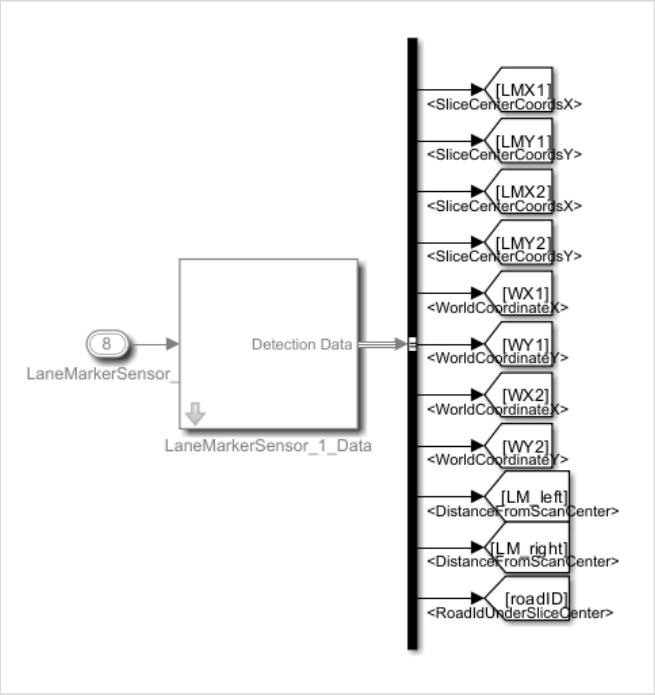


Figure 33: Simulink representation of the Lane Marker Sensor

3.6. UART Process Blocks:

Serial receive Comm 5: that receive data from Serial Send Comm5.

ASCII to string Block to convert ASCII Data to string.

Splitting Data Block: used to split UART Data to The desired output: Throttle, Brake force and Control. Output of this block is Control, Brake force, steering angle, lane ID, lane Keeping, Lane marker angle.

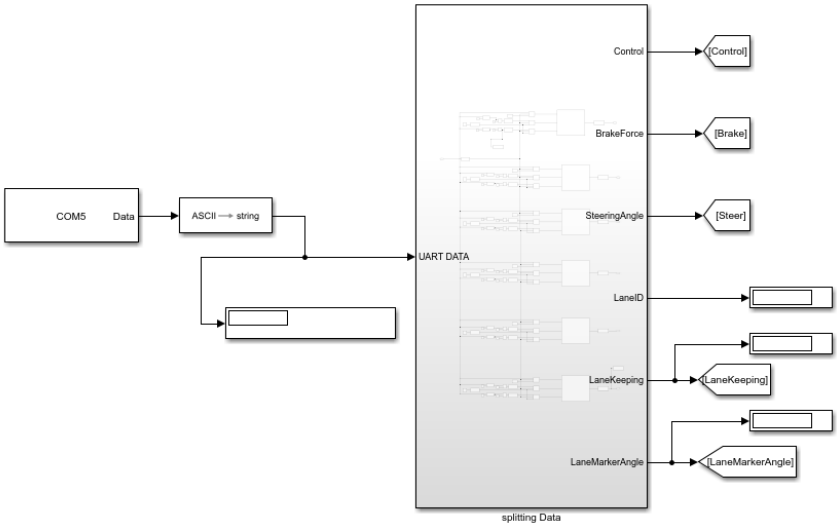


Figure 34: UART Blocks Process

3.7. Test Cases Scenarios

3.7.1. Fixed object

In this scenario we want to get our car to stop before a fixed object according to our auto braking system without the interface of the driver.

In this scenario our system can be applied on the whole range of the constrains that the speed can be in the range of 0 to 54 km/hr.

At this scenario our car velocity is maximum so that we can test our system according to corner cases. So: -

- The target car velocity is 0 km/hr.
- The host car velocity is 54 km/hr.

After the braking system is applied the distance between the host car and the target car at the maximum velocity is approximated 1 m and this will be illustrated more as shown in *figure 35*.

Figure 35 :Fixed object view

as shown the system succeeded to stop the car without crashing with variety of braking force according to the speed and the range between two cars and this will be shown more at the following scopes reading.

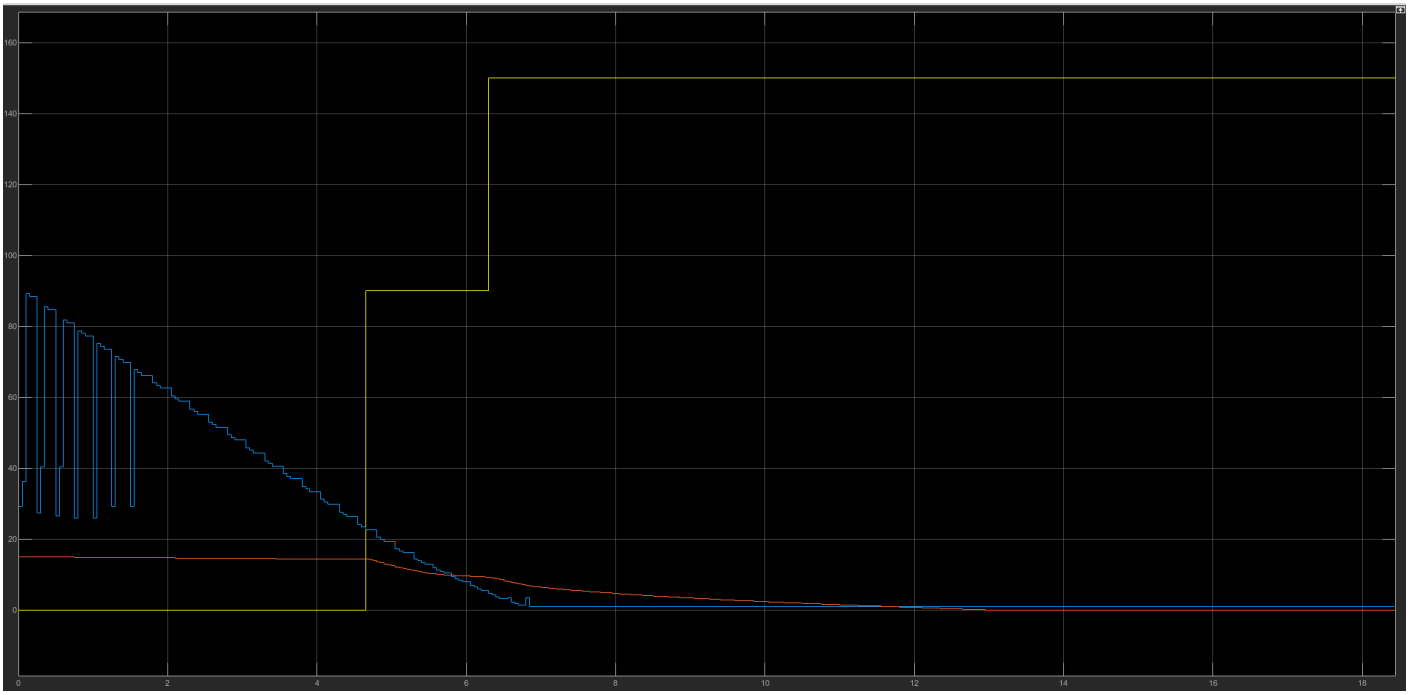


Figure 36: Fixed object range, velocity and brake force

As shown in *figure 36* at scopes that our car (the orange signal) started to move at 15 m/s that is equivalent to 54 km/hr. and the system started to take action when time to collision is less than critical time (as described before).

After the braking system take the control from the driver and started to control the car the velocity started to decrease gradually until it reached to zero consequently the range between the two cars started to decrease from 60 meter to 1 meter between the two cars.

Also, this will be illustrated more in the following reading scope of the braking:

as shown the brake force is applied at two records first with 40 % of the maximum braking force to decrease the velocity of the car gradually and then apply the maximum braking force which is 150 bar when the host car becomes closer to the target object (that in this scenario is the Toyota car).

This is the scope reading that collects all reading signals of the system.

- The orange signal indicates the car velocity signal.
- The yellow signal indicates the brake pressure signal.
- The blue signal indicates the range signal.

3.7.2. Moving Car

In this scenario we want to prevent our car from crashing into another moving car using relative velocity according to our auto braking system without the interface of the driver.

In this scenario our system can be applied on the whole range of the constrains that the speed can be in the range of 0 to 54 km/hr.

At this scenario our car velocity starts from maximum then after applying auto braking our velocity decreases according to the opposite moving car to save safe distance between the two cars. After that the other car keep moving and our car is slow down until the distance between the two cars becomes more than the critical one so that our car start to increase its velocity again. After a while, the other car stops suddenly then our system will take control again and fully stops our car preventing it from crashing.

System Slowing down:

According to the velocity of the opposite moving car our system will response to this easily as when the opposite car slows down and the range starts to slow down also, our system start take the control from the driver and start to slow down our car velocity when it finds that the driver did not take any action.

And when the range between two cars increase the system leave the control to the driver.

Figure 37: Moving car view

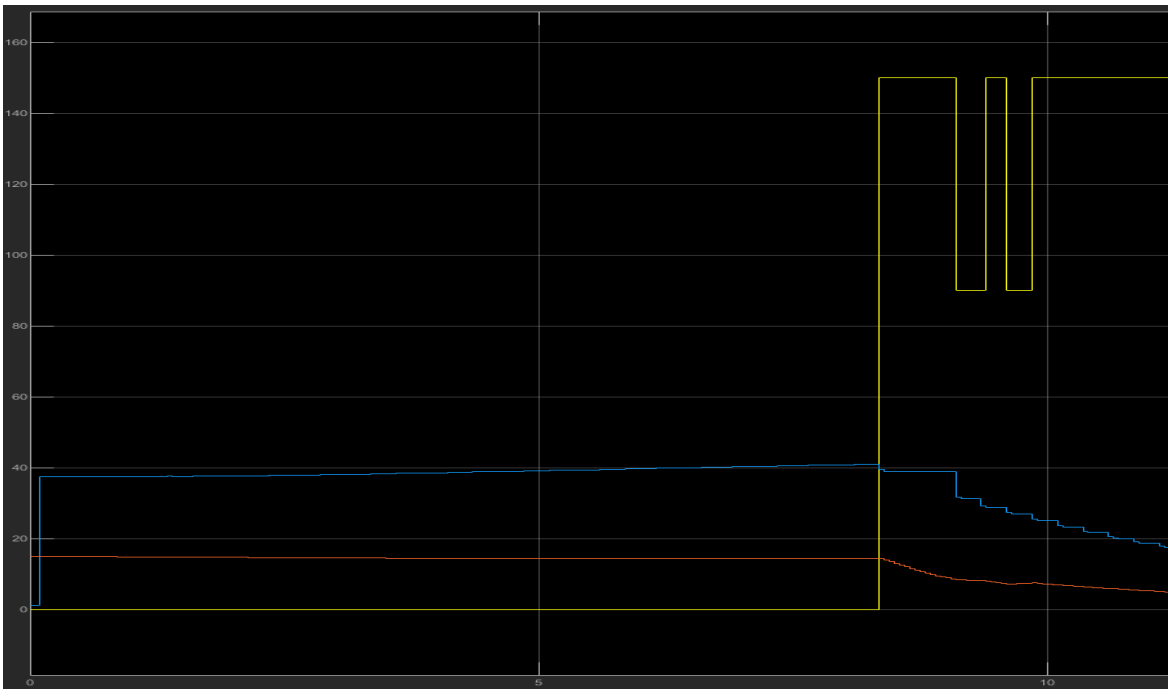


Figure 38: Moving car slows down

The radar range was decreased such that it only detects the lane and neglect the surrounding environment, therefore once the car entered the range of the radar it applied full brake immediately,

As shown in *figure 38* the range (blue signal) decreases then the car velocity (orange Signal) decrease, according to the TCC.

Also, the scope of the brake illustrates that the system starts with 50% of the braking force to slow down its velocity.

1. Increasing the range && increasing the Velocity:

As the range between the two cars increase the velocity of the host car increases, but this case where the car is relatively slow compared to the host car therefore, we find them moving close to each other simulating the movement of cars in a traffic jam.

3. Slows down to stop:

When the opposite car slows down to stop and there is no action from the driver, so the system takes the control again and start to slow down till it stops by applying the brake to be 150 bar.

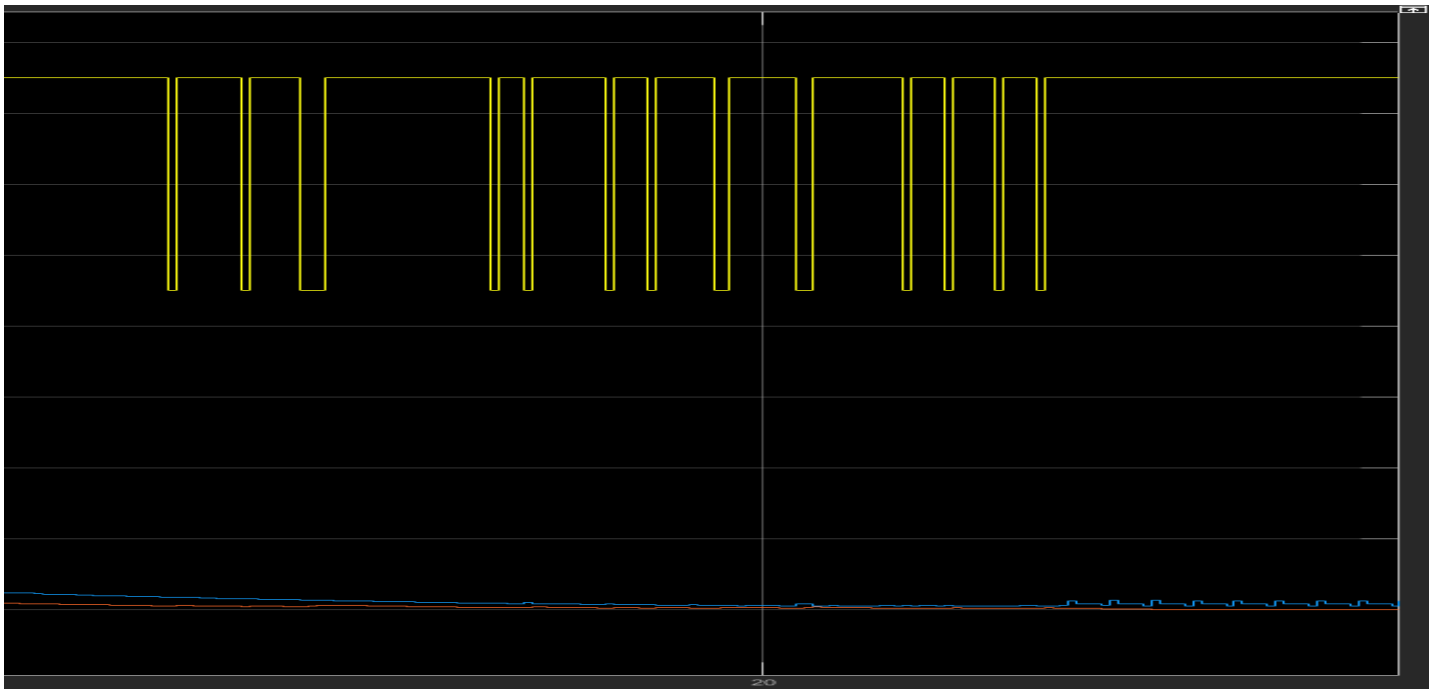


Figure 39: Cars slows down to stop

We can see the distance between the two cars increasing and decreasing till the car stops suddenly therefore the brake was applied to its maximum value.

3.7.3. Man crossing road

In this scenario we want to get our car to stop before a crossing man according to our autobraking system without the interface of the driver.

In this scenario our system can be applied on the whole range of the constrains that the speed can be in the range of 0 to 90 km/hr.

At this scenario our car velocity is 79.2 km/h.

So:

- The target male velocity is 7.2 km/h.
- The host car velocity is 79.2 km/hr.

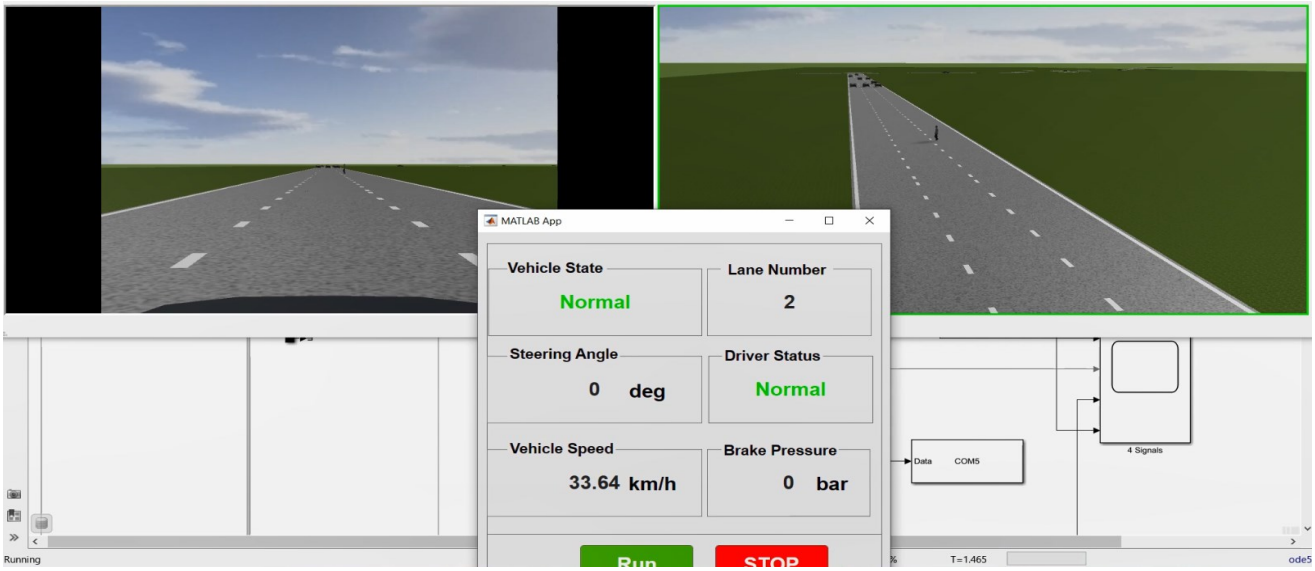


Figure 40: Man cross the road and stands in the middle.

When the person becomes at the warning region of the car, it applies the algorithm where the steering angle is applied to change the lane in order not to hit the person or stand still at the place where the person is standing.

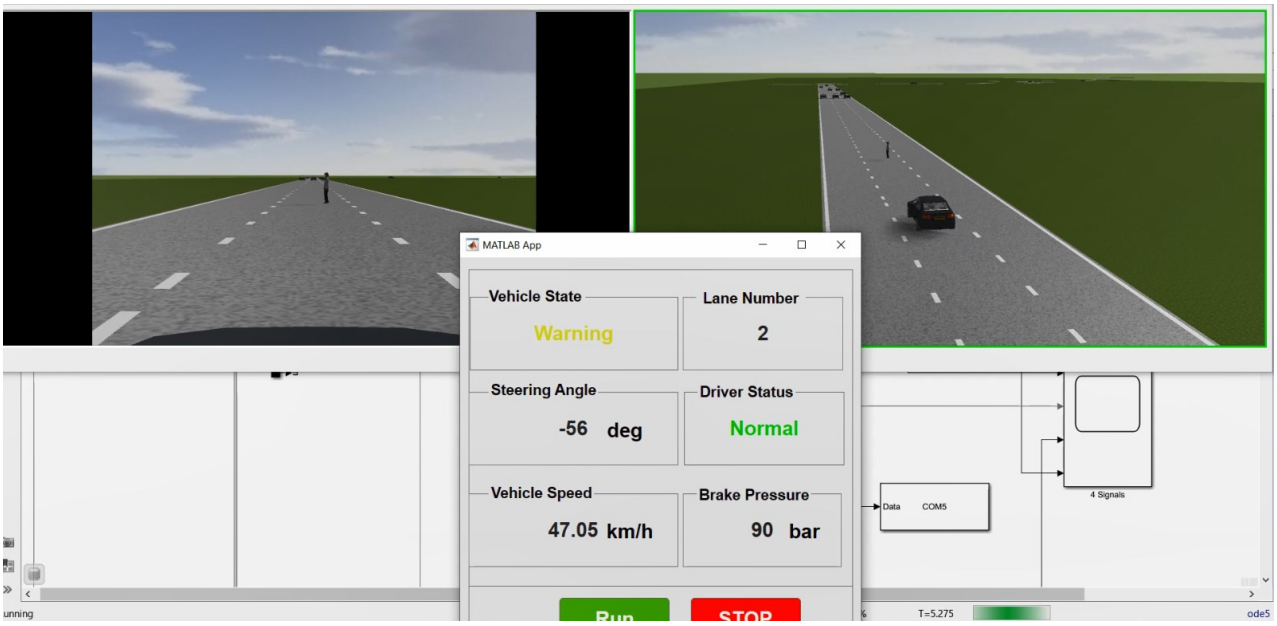


Figure 41: Changing the lane once the person became in the warning region with respect to the car.

Now, the car has changed the lane and has moved forward in its path.

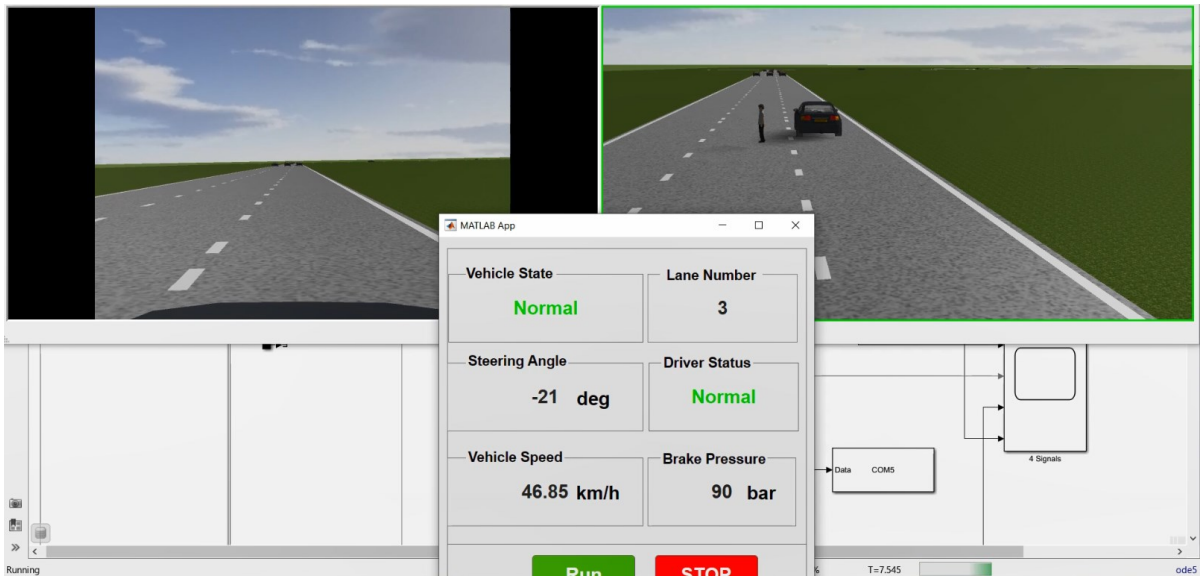


Figure 42: Car after changing the lane to avoid the person.

We used the scope to view our main signals.

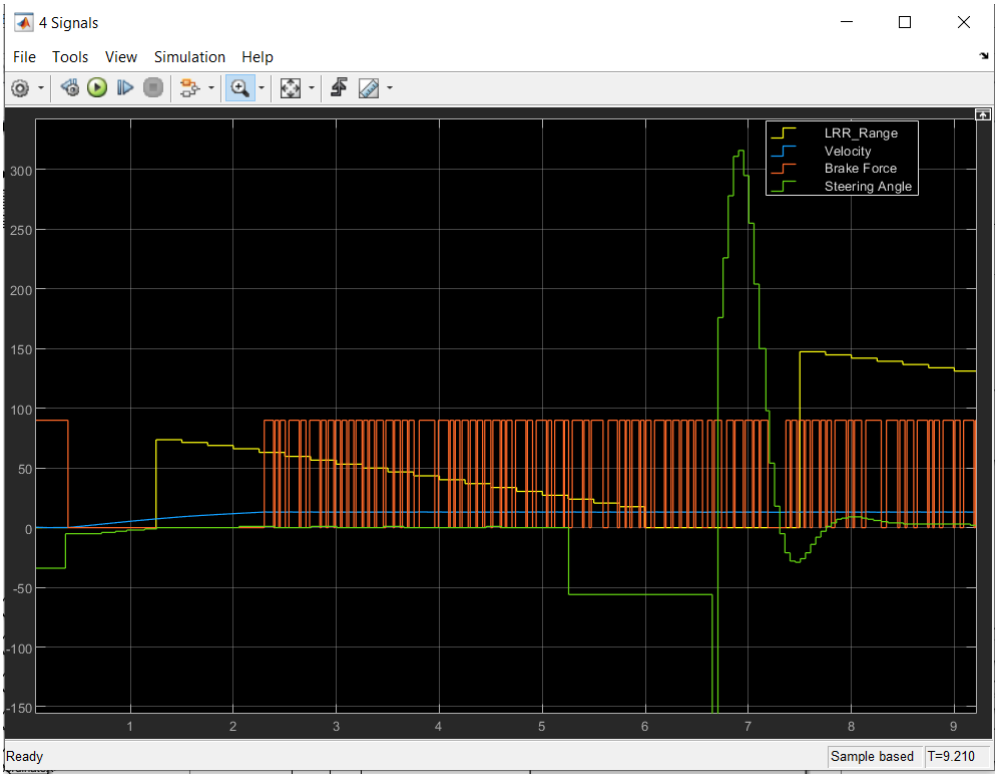


Figure 43: Range, velocity and brake force signals.

3.8. Ncap Results Vs Ours

3.8.1. Ncap demo experiment:

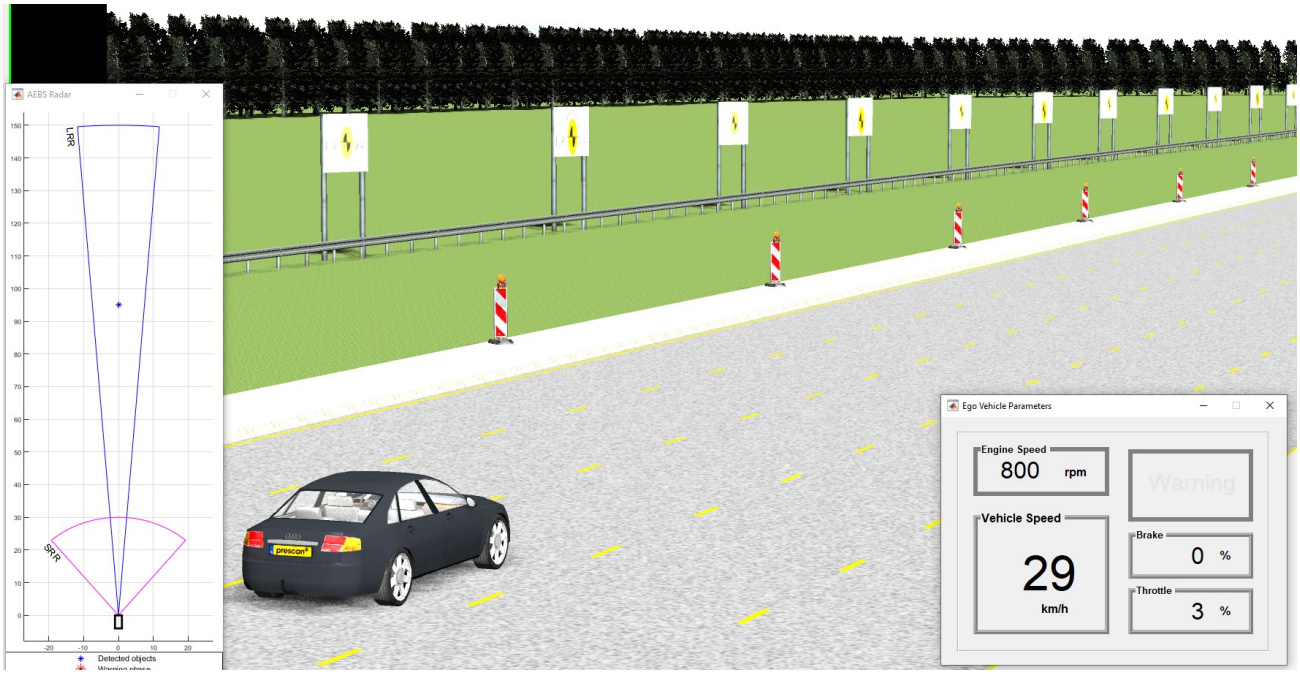


Figure 44: NCAP Model

As shown in the *Figure 8* the graphs at the beginning of detecting an object in NCAP Demo Experiment. The host car speed is 29 km/h and the range between host car and target car.

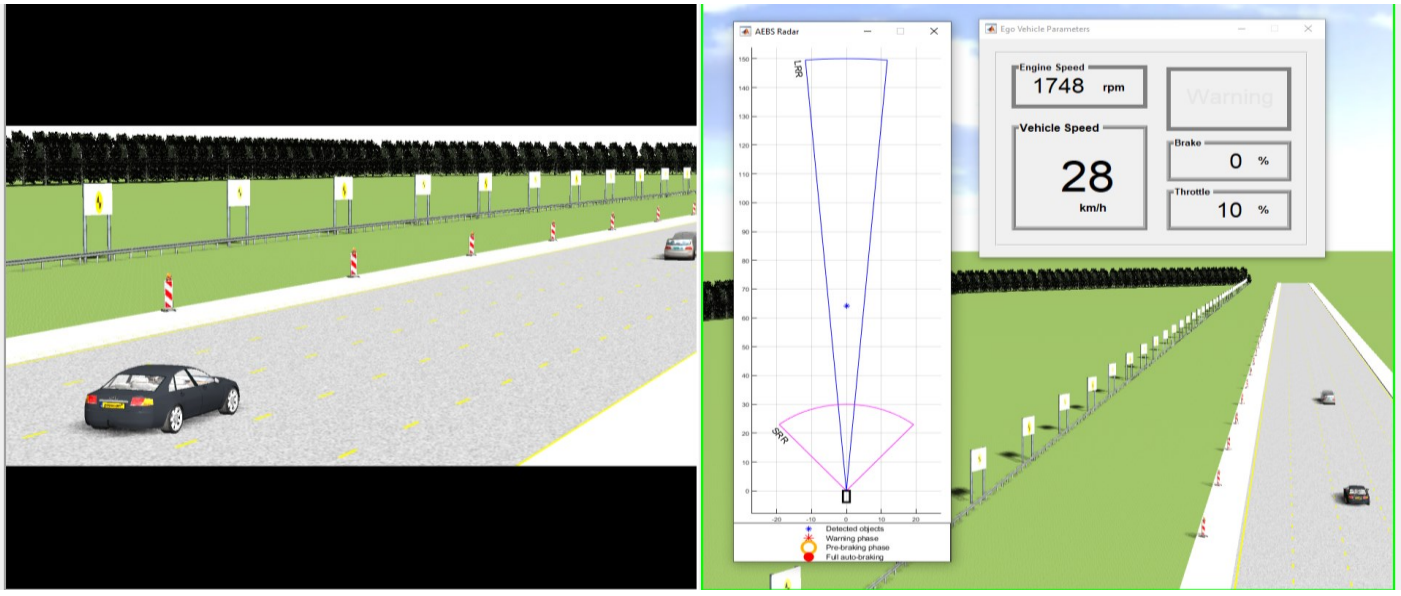


Figure 45: Car detected by Long Range RADAR(LRR)

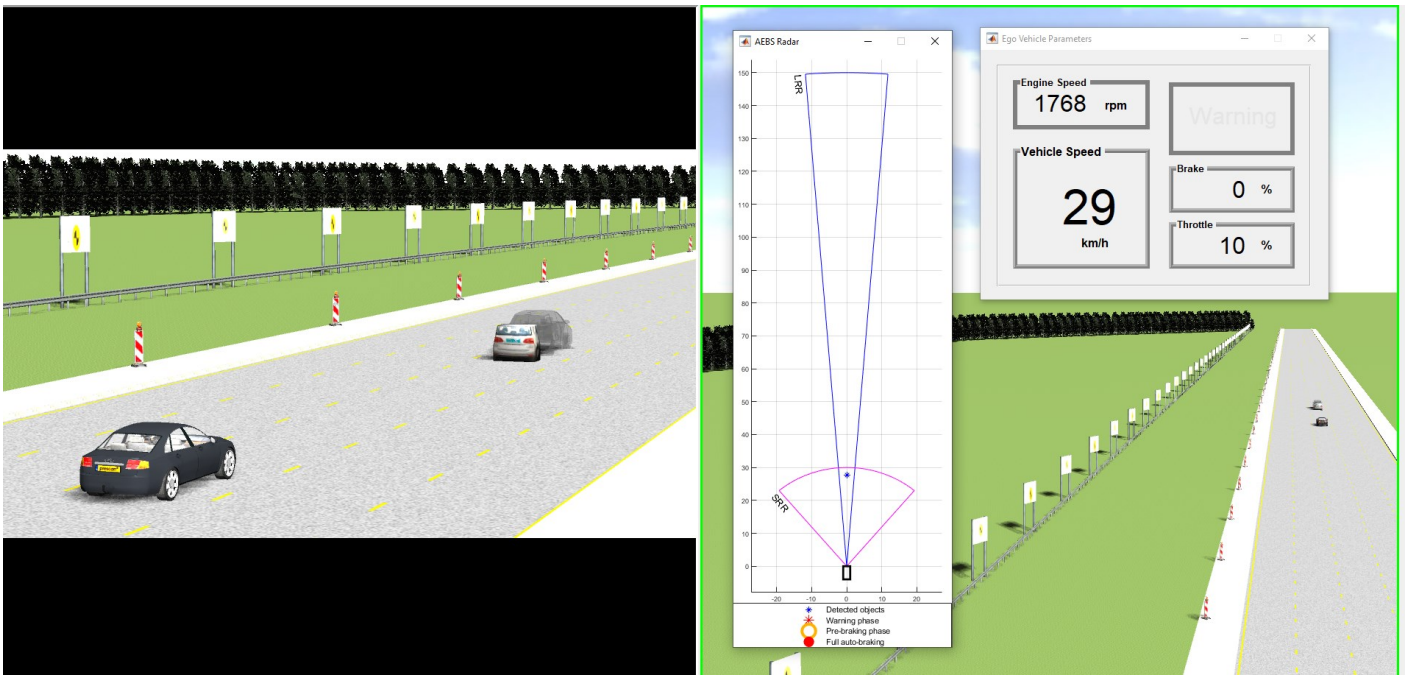


Figure 46: Car is detected by Short Range RADAR(SRR)

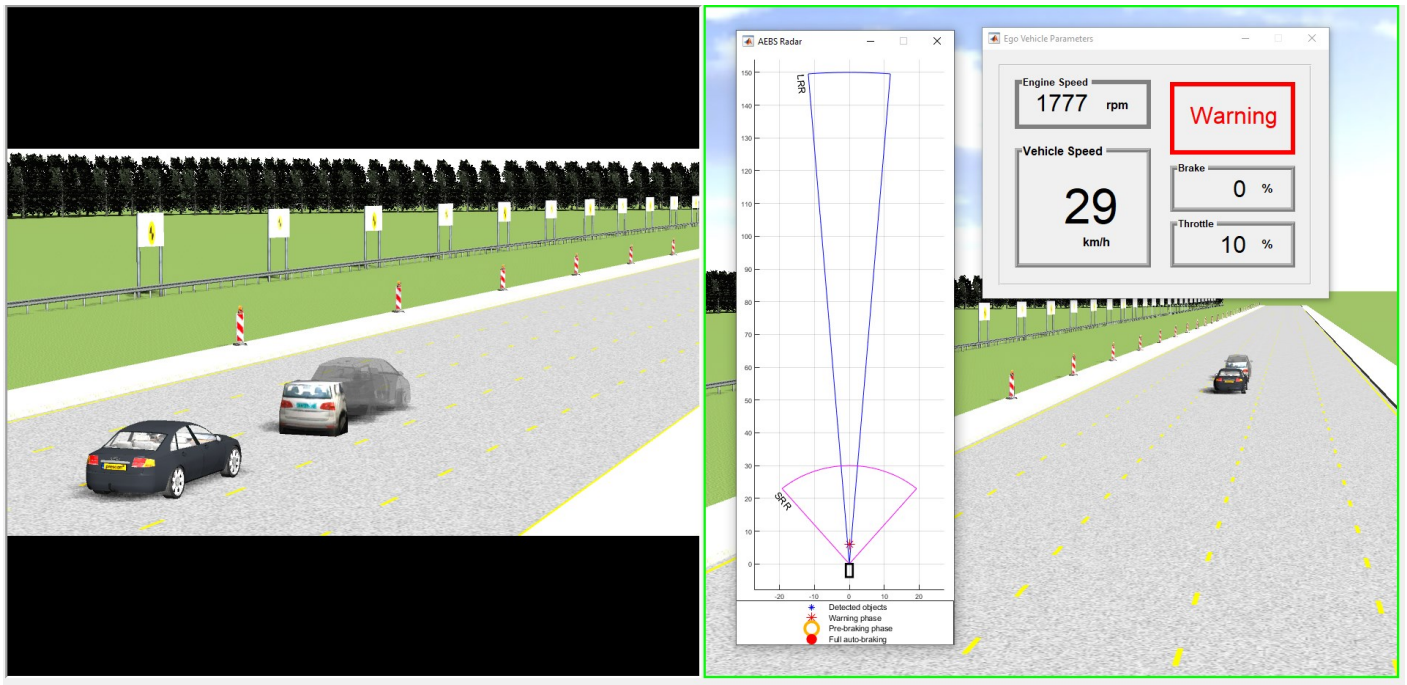


Figure 47: Car enters warning state.

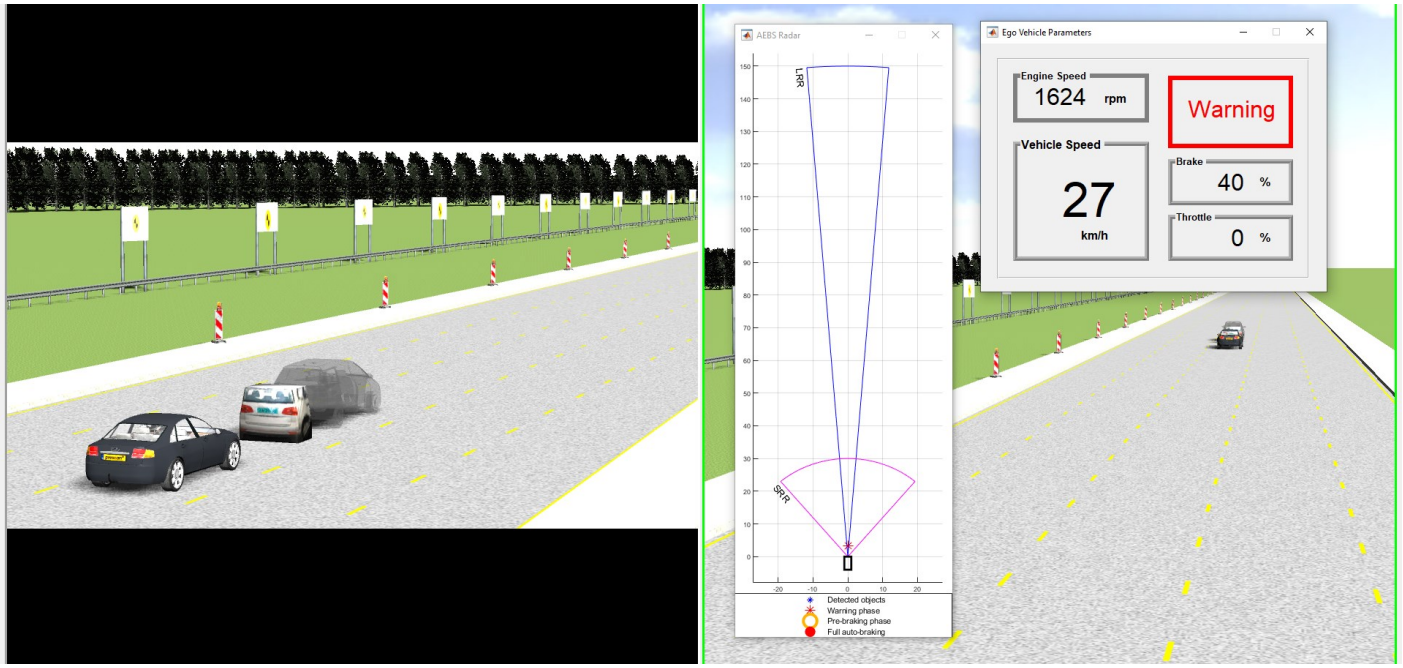


Figure 48: Applying 40% brake pressure.

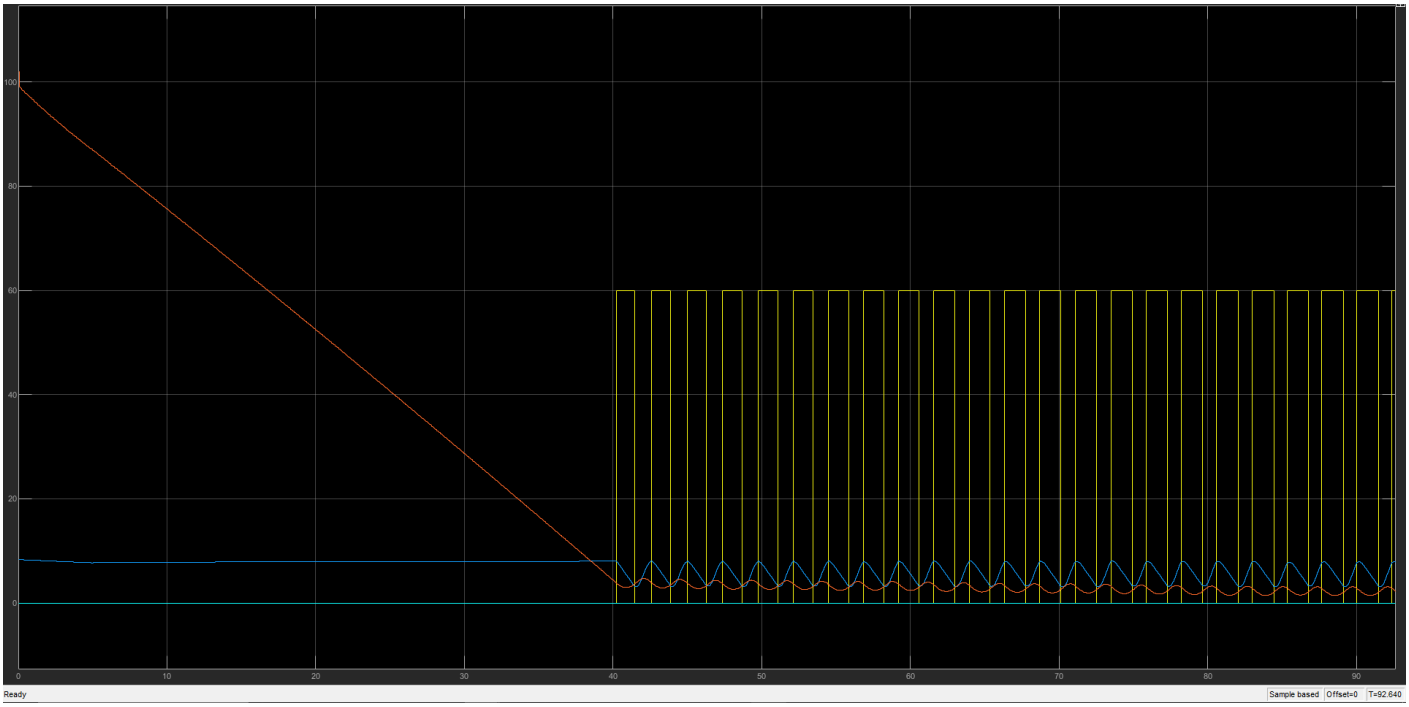


Figure 49: NCAP range, velocity and brake signals

3.9. Our generated scenario

We will have at first vehicle state, driver distraction and driver drowsiness will be normal.

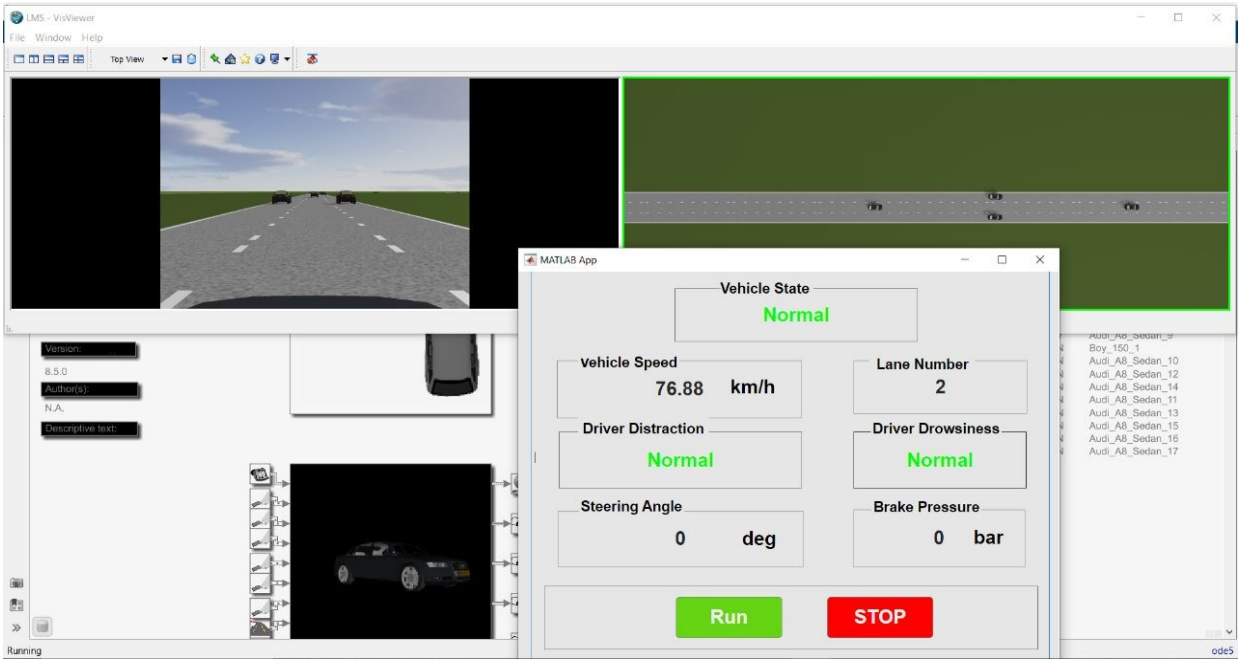


Figure 50: All statuses are displayed at the beginning of the simulation.

When a car is found to be in the range of the warning the car will keep its speed and write only a warning to alert the driver of a nearby object.

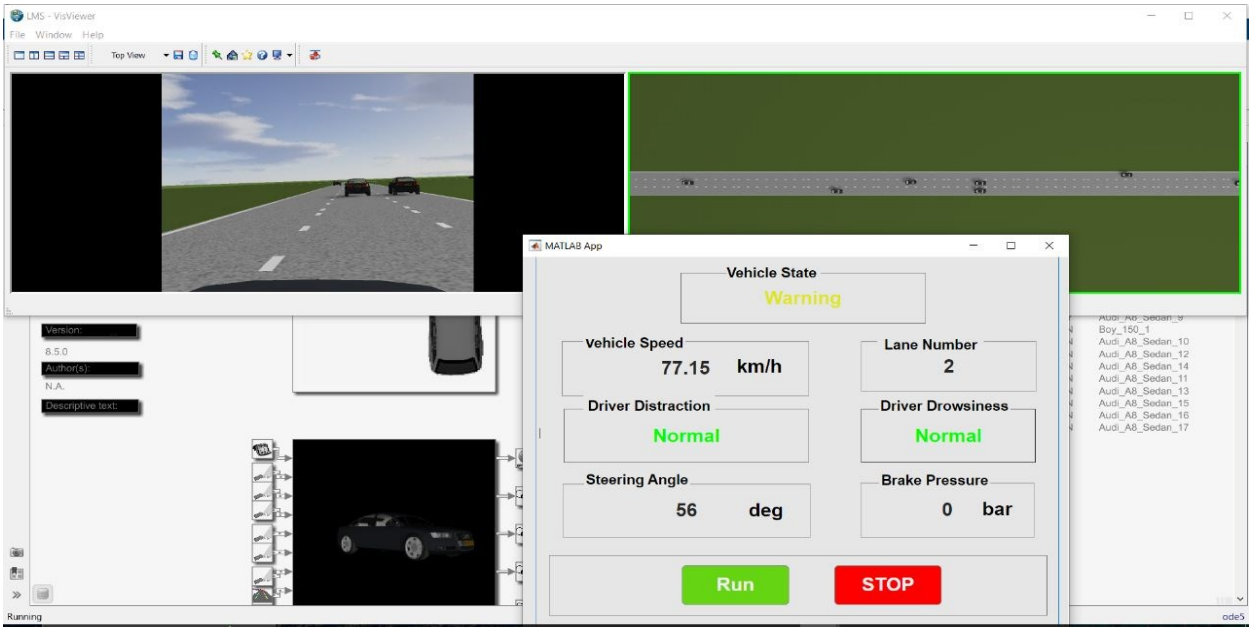


Figure 51: Warning status when an object was detected in the warning range.

When the driver is detected to be drowsy after 10 frames then the car will enter the warning status and a buzzer will be played to alert the driver.

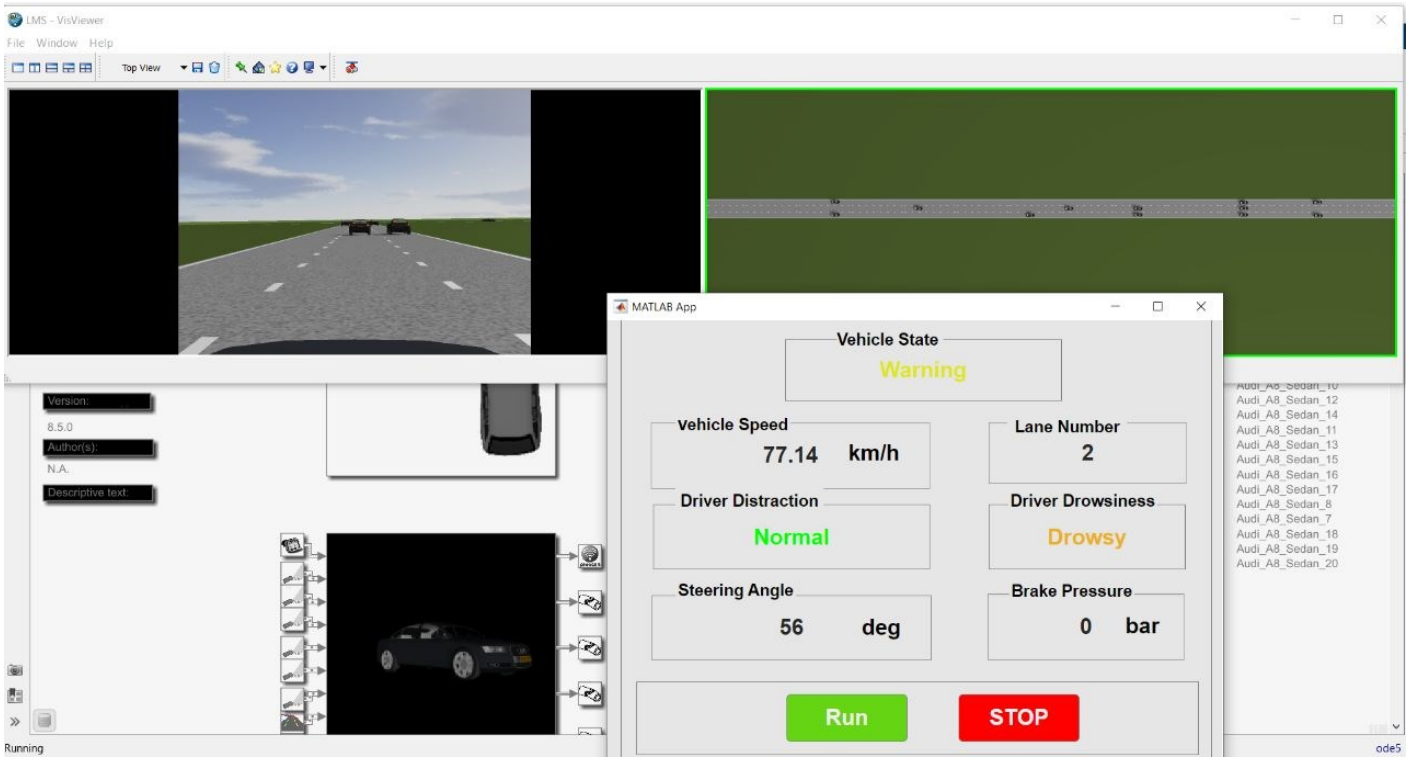


Figure 52: Drowsiness is detected and the vehicle state is changed.

The same action will be made for the distraction detection of the driver and the buzzer will be played for alerting.

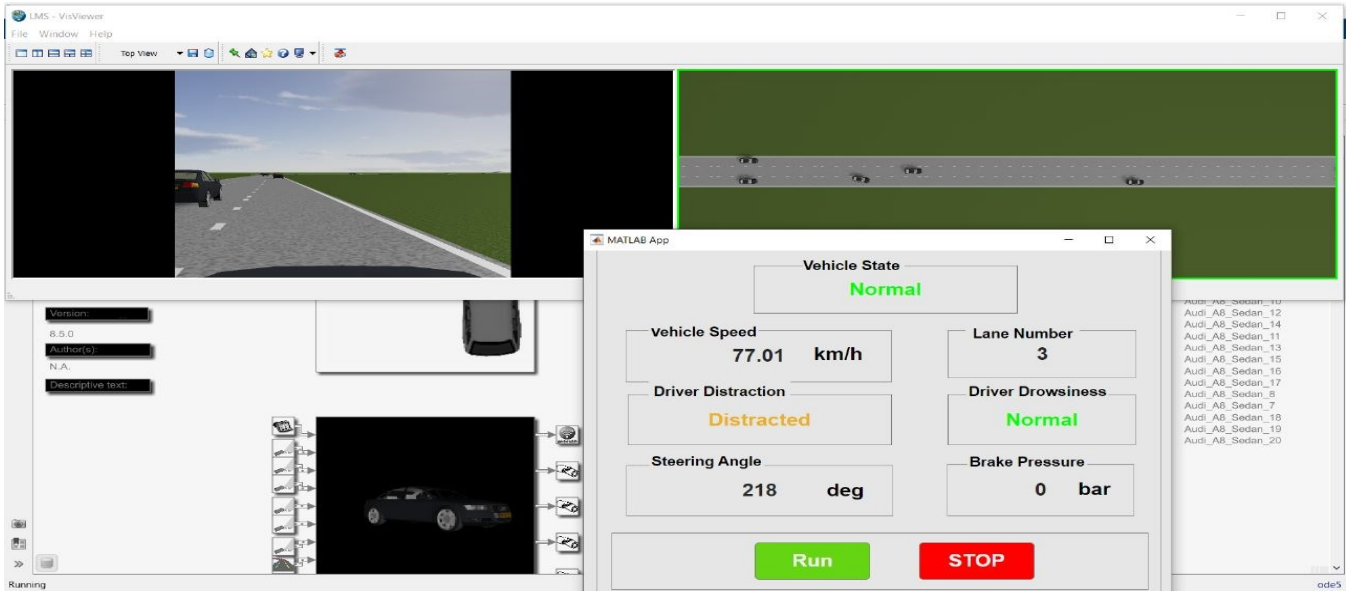


Figure 53: Distraction is detected and the vehicle status is changed.

When the driver is found to fall asleep will driving, the car if it's in the middle lane will change the lane to go to the right is available, if not, it will go to the left lane and stop the car.

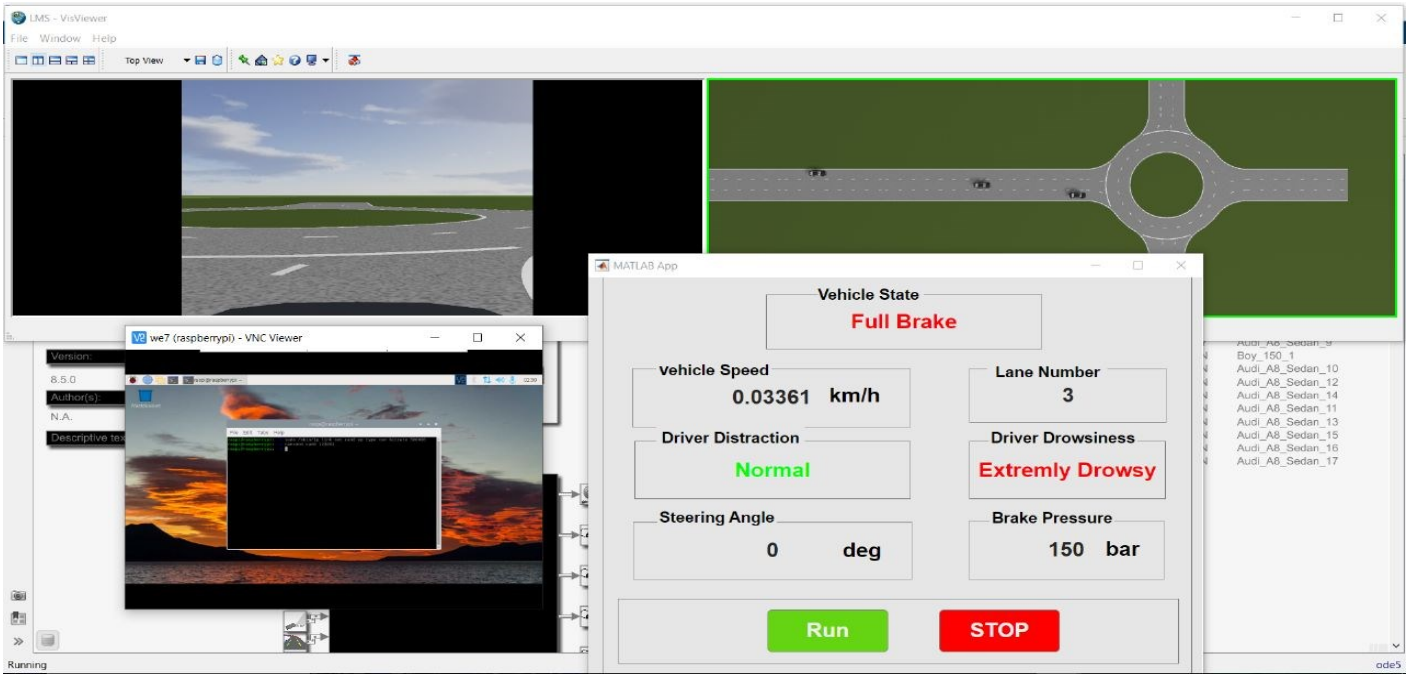


Figure 54: Full brake state when the driver has fallen asleep.

If the car has no way to continue in its path, will apply full brake until a change is detected such that it can return back to its normal state.

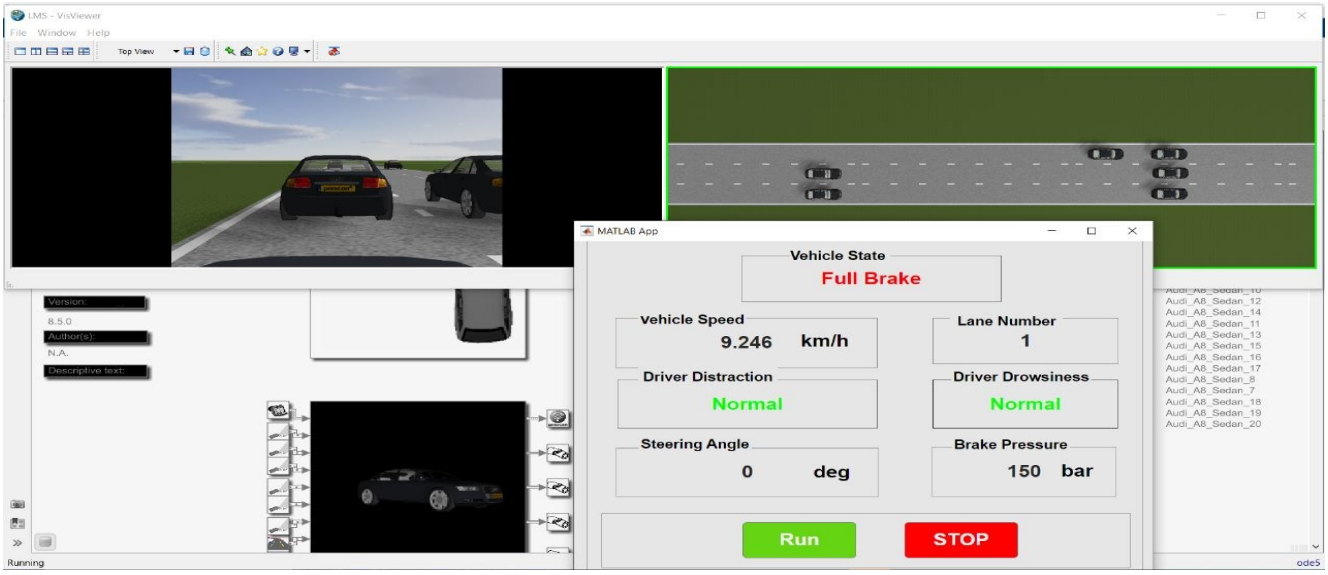


Figure 55: Full brake is applied when the lanes were full of cars.

Chapter 4

Auto Braking System Algorithm

4. Algorithm

4.1. Objective:

The objective of this study is to develop a method to auto brake the car if there was an emergency if an object appeared suddenly, the driver did not take action and the car was about to crash or the driver was distracted or drown.

The algorithm estimates the time to collision (TTC) and if it is lower than a critical time (TCritical) then the car brakes. In next section will explain how to calculate both of TTC and TCritical.

The algorithm is designed to work only within the speeds lower than or equal to 90 km/h. If the speed is greater than 90 km/h, the driver has the full control as auto-braking is such a speed may cause a disaster.

The logic should handle many cases in which we make sure of two things: -

1. The car should apply the full brake force if something appeared suddenly and TTC was already lower than the critical time needed at this speed.
2. The car brakes gradually if the object appeared and TTC was still higher than the critical time needed at this speed.

We make sure that TTC is greater than zero before we apply this concept. The brake force now is inversely proportional to TTC.

- Max TCC will cause min brake force.
- Min TCC will cause max brake force.
- The value of the brake force varies depending on TTC.

Or we can just apply 50% of the max force.

4.2. Calculating TTC:

Using the radar, we can get the relative speed of the nearest object and the distance between the car and that object. If the relative speed -from our car POV- is less than or equal to 0, there will be no collision. But if the relative speed is greater than 0, we calculate the actual TTC using the formula:

$$TTC = \frac{Distance}{Relative\ Speed}$$

By knowing our car velocity, we get the critical time needed to start the braking action - the way we get it in the last section - so that the car stops just before hitting the object.

By comparing these two values (TTC & Critical time). If the TTC is greater than the critical time, the car will do nothing, but if the TTC is equal to or less than the critical time, the car will autobrake immediately.

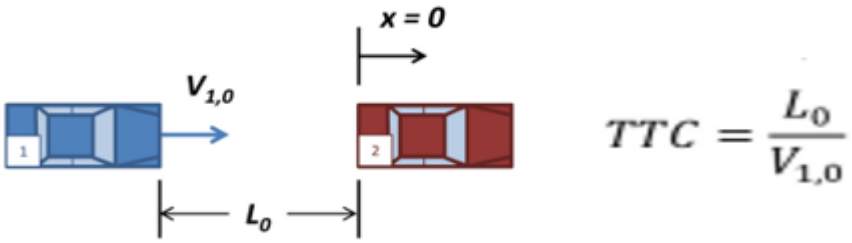


Figure 56 :TCC

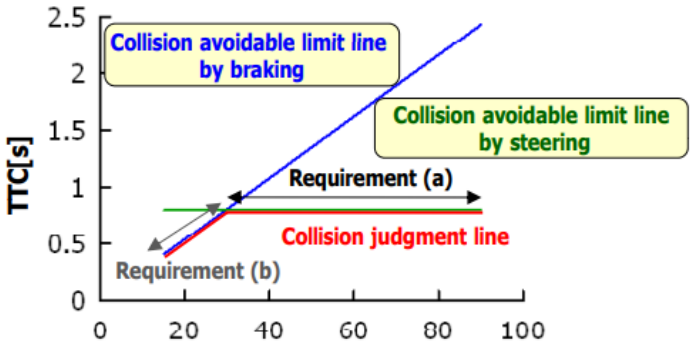


Figure 57 : TTC vs Distance

4.3. Algorithm procedure

Firstly, the Algorithm is divided into two main features **Auto Braking Emergency System, Steering angle** to avoid the Stopping Car suddenly problems which will be discoursed in the next sections.

For applying the first main features is ABES, we use 2 radar sensors:

- long range radar (LRR). Range = 150m narrow beam (9 degrees) to detect the far objects and make into consideration what events and actions must be applied.
- short range system (SRR). Range = 30m wide beam (80 degrees) to detect the side car that suddenly appeared.

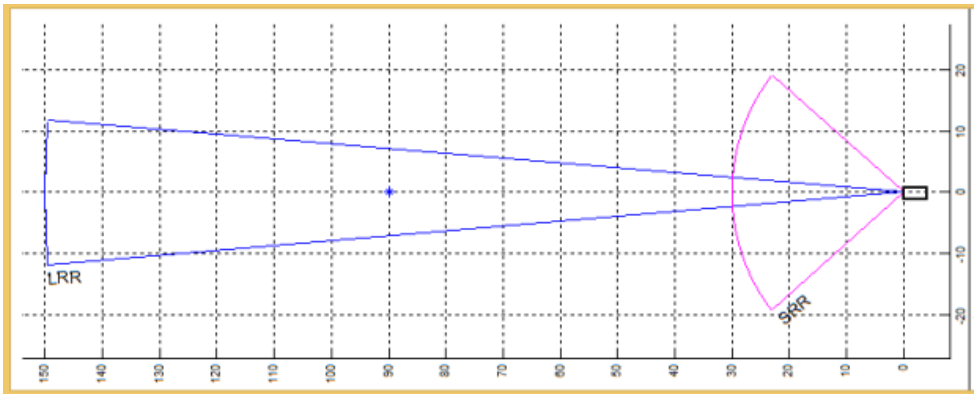


Figure 58 : LRR & SRR radars

The system starts to warn the driver 2.6s before the predicted impact (TTC = 2.6s).

At TTC= 1.6s, 50% brake pressure is applied. This gives some additional time for driver to react and increases the possibility to evade or brake.

If the driver starts to brake on his own, system will help him with additional force, reaching maximum possible value.

If no reaction is taken, from TTC = 0.6s full braking pressure is applied.

It should be noted that autonomous braking is activated only when a colliding object is registered by both sensors. Partial braking requires information only from the first sensor, the LRR.

The information from the sensors is used to determine relative speed between the host vehicle and an obstacle. Based on this information, absolute velocity vector of an obstacle is calculated. With the assumption that the actor is moving strictly forward at the moment, absolute velocity vector allows to determine the point, where the collision will take place. Then time to collision is calculated, compared with set values. Based on TTC, signal flags that are used as system triggers, are assigned. In addition, when system is triggered, an animation of blinking braking lights was added in Viewer to represent AEBS work and inform other drivers about emergency braking.

It should be noted, that radar 1 (long range beam) can raise flag 2.6s and 1.6s, while radar 2 (short range beam) can raise flag 0.6s only.

Warning Flag	1.6s Flag	0.6s Flag	Action	Status
0	**0**	**0**	**Do nothing**	No any object is detected
0	**0**	**1**	**Do nothing**	Side object is detected and isn't opposite to the car.
0	**1**	**0**	**Apply 40% braking force**	Object appeared suddenly in range greater than SSR range and its TCC <=1.6.
0	**1**	**1**	**Apply Max braking force**	Object appeared in range less than SSR range and its TCC <=0.6 after appearing suddenly in half braking range.
1	**0**	**0**	**Apply warning to driver**	Object appeared in range greater than SSR range and its TCC <=2.6 but TCC >1.6

1	**0**	**1**	**Apply warning to driver**	2 separate objects, side one and opposite object range is greater than SSR range and its TCC <=2.6 but TCC >1.6
1	1	0	Apply 40% braking force	Object appeared in range greater than SSR range and its TCC <=1.6 after appearing in warning range
1	1	1	Apply Max braking force	Object appeared in range less than SSR range after appearing in warning range then half braking range.

Table 1: Algorithm Flags

The table shows what action is deployed depending on flags sequence. Where there is no detection, or only 0.6 flag is raised, no action is taken. Rising 1.6s flag results in 40% of braking force being applied. If both flags 1.6 and 0.6s are raised, it means that impact will take place earlier than 0.6s and full braking is activated. If the driver starts braking when 1.6s flag is active, system will help him to reach full braking force; the forward collision warning mode. If driver brakes independently from system and there is no risk of collision detected, his braking force is passed along.

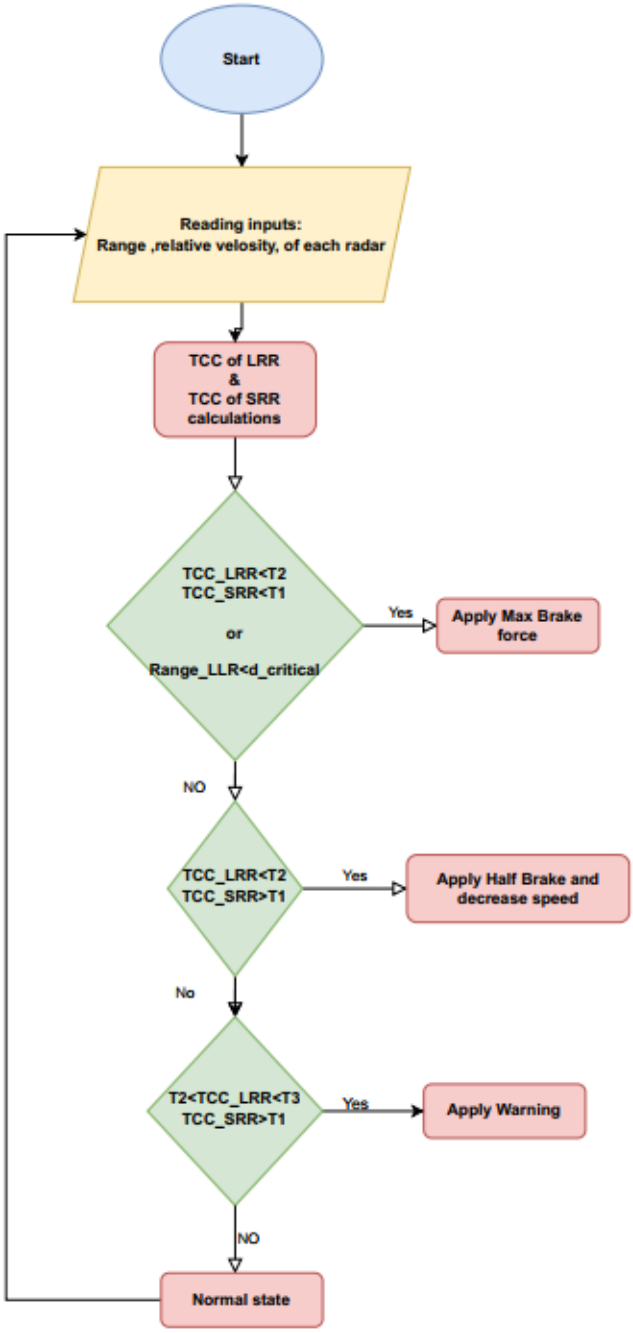
4.4. Algorithm State Diagram:

We get three inputs from each radar readings and car mechanics:

- Car speed
- Relative velocity of the nearest object
- Distance of the nearest object

After that we calculate TTC as actual time of collision and $T_{Critical}$ as the time needed for the car or fully stop as mentioned before.

Then, as shown in *figure 53,* we compare between the TTC and $T_{Critical}$ to decide the action as follows:



T1 :Threshold time of the Full Braking.
T2 :Threshold time of the half Braking and decreasing speed.
T3 :Threshold time of the Warning state.
T1<T2<T3
d_critical: distance after which the braking must be applied immediately to solve some problem.

Figure 59 :Algorithm Flow Chart

4.4.1. Normal State

In this state, our car is moving without any constrains. The car is fully controlled by the driver without any interaction of our system. If the car detects an opposite object that gets closer and may crash into,the system will transfer into the ***Warning State***.

- TCC > T_3

4.4.2. Warning State

When our long rang radar starts to detect an object that gets closer to our car and the driver did not take any action to slow down the car velocity, our system starts to raise a warning signal to warn the driver indicating that the danger is coming within less than T3 sec if he did not take action. If the driver took the action and the distance became in the safe region again the system will return back to the ***SafeState***.

- T_2 <TCC_LRR < T_3

4.4.3. Half Braking State

When our long-range radar starts to detect opposite object and the short-range radar not detect anything in his range, in this case we apply partial braking and decrease the speed gradually as a response.

- TCC_LRR < T_2
- TCC_SRR >T_1

4.4.4. Auto Braking State

In this state, the driver did not take any action to slow down the car to prevent it from crashing into the object. If the brake does not start immediately the car will crash the object, so the system takes the control and starts to apply the braking force till maximum the car fully stops then the control is back to the driver and the system returns into the ***Safe State*** again.

- TCC_LRR<T_2
- TCC_SRR<T_1

We get the problem between the relation of Normal and Full-Brake states , when the car enter the Full-brake region and apply the maximum brake force, decreases the speed immediately to zero ,the equation TCC=Range/relative speed describes the problem when relative speed tends to zero , the TCC tends to infinity which make the following iteration to the TCC make the region decision is Normal region not Full-brake as TCC becomes of Course greater than T_3, so the following action taken based on the Normal state and the speed increases again and the TCC decreases again to follow the Full-Brake state and this scenario will repeat itself and make the Braking signal oscillate and switches up to max and down to zero at the same time.

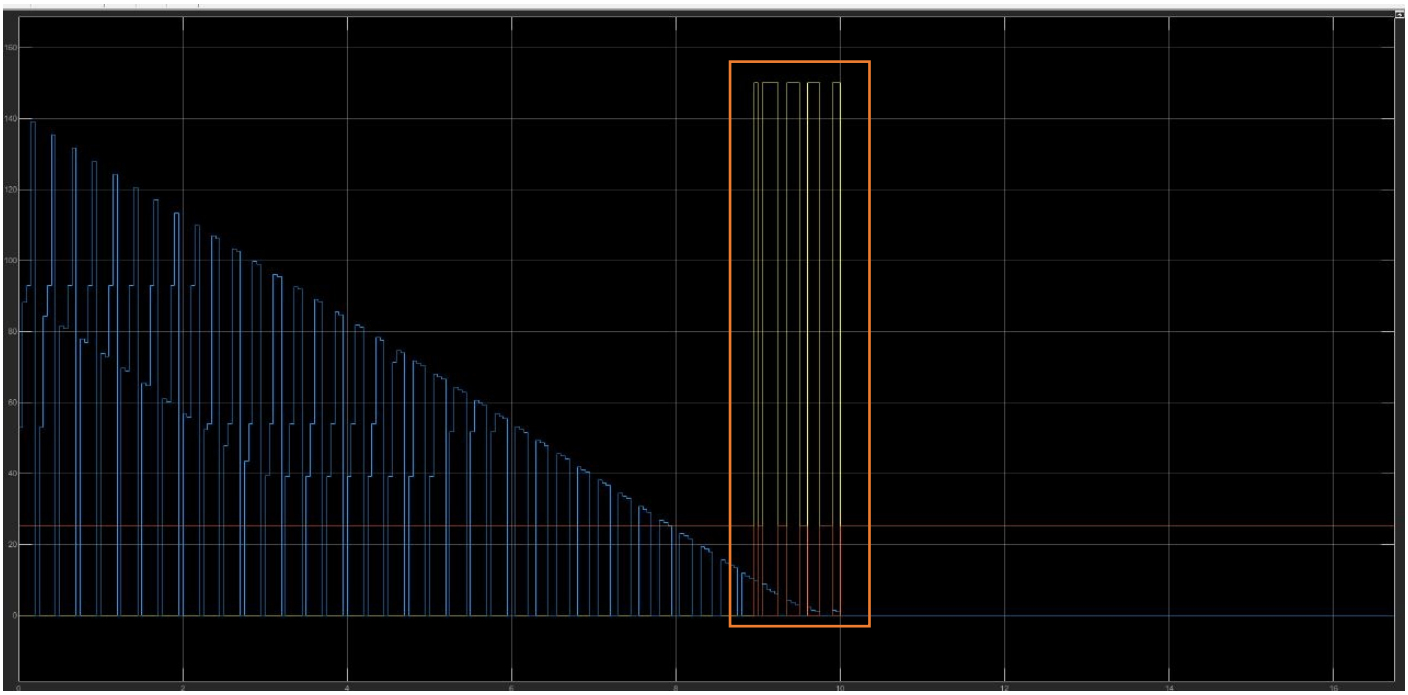


Figure 60 : TCC and relative speed problem between Normal and Full-Brake states

To solve this problem, we make condition to set critical distance after which the max braking applied only and nothing happen again until the distance between the car and the opposite object becomes greater than this critical distance.

4.5. Algorithm State Machine:

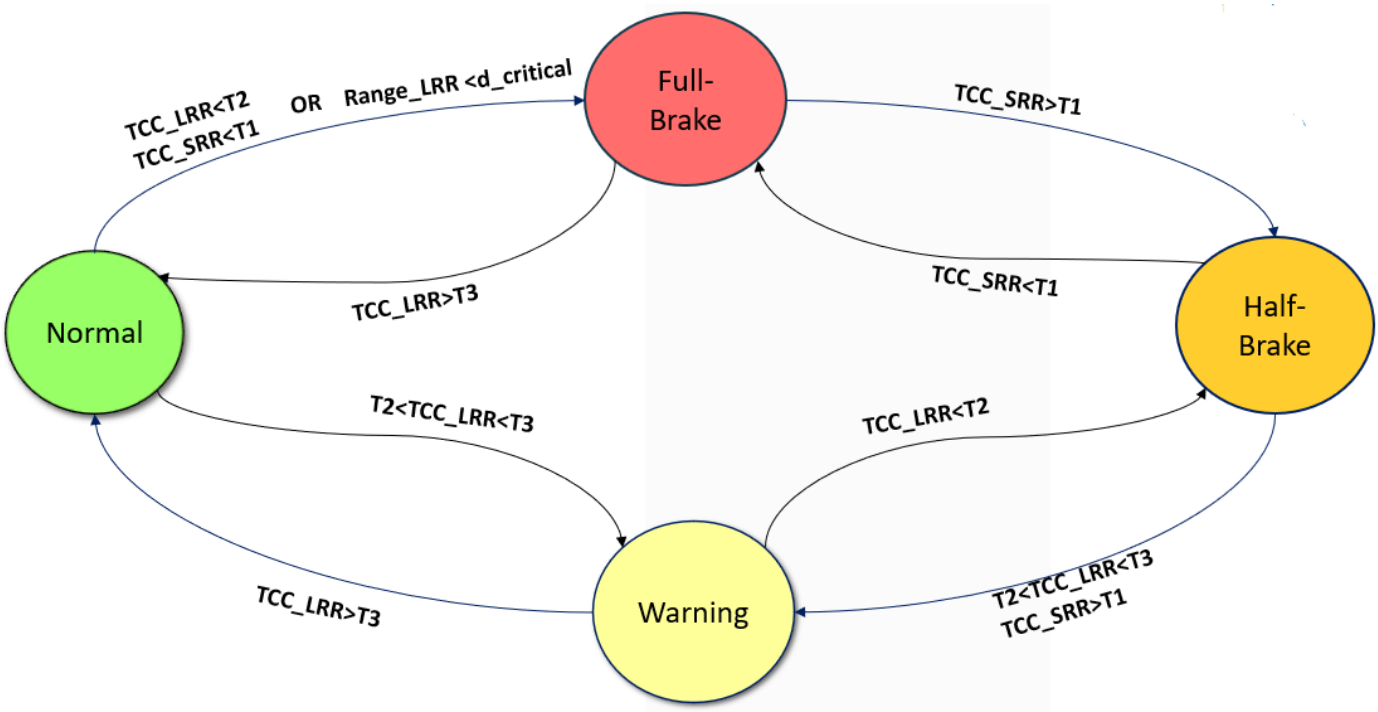


Figure 61 : Algorithm State diagram

Note:

T_1: Threshold time of the Full Braking.
T_2: Threshold time of the half Braking and decreasing speed.
T_3: Threshold time of the Warning state.
$T_1 < T_2 < T_3$
d-critical: distance after which the braking is a Must.

4.6. Pseudo code

- If car velocity > 90 km/h then

  do nothing.

  Driver controls the car.

- Else

  Calculate TCC for each Radar.

  - if TCC_LRR<T_2 && TCC_SRR<T_1 || Range_LRR <d_critical

    Apply max brake force.

    System sets speed to zero.

    System stops a warning.

  - End if TCC_LRR< T_2 && TCC_SRR>T_1

    Apply half brake force

    Decrease speed

    System stops a warning.

  - Else if TCC_LRR <T_3

    System raises a warning.

  - Else

    System will be in the normal state.

    System stops a warning.

4.7. Algorithm Implementation in FreeRTOS Operating System

FreeRTOS is utilized to implement an algorithm for real-time simulation control of the auto braking system. Simulink serves as a middle layer between the algorithm and the simulation environment, receiving parameters such as brake force, steering angle, lane ID, and lane marker angle. FreeRTOS tasks handle data reception, processing, control logic, and simulation control. Task synchronization mechanisms, like queues, facilitate inter-task communication. The algorithm utilizes the received data to make decisions and control the vehicle's behavior in Prescan.

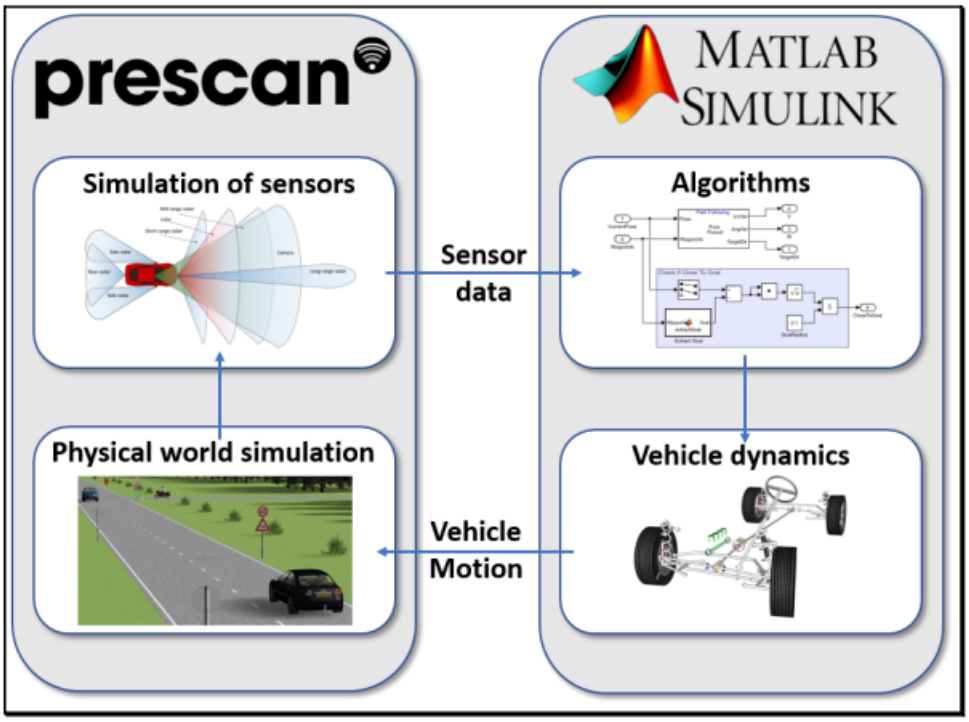


Figure 62 :Relation between PreScan and Simulink

4.7.1. Task 1

Task 1 receives real-time simulation data from Prescan via UART 1, including information from the 6 used radars (range, relative velocity, and azimuth angle) and other data about the vehicle (desired velocity, current velocity, heading angle, and lane marker sensor readings for scan 1 and scan 2). The received data needs to be sent to Task 2 and Task 3 using two queues. Since UART 1 is a shared resource and will be used later in Task 4, a Mutex is used to protect the resource from being accessed by multiple tasks simultaneously. Additionally, Task 1 is blocked for a specific amount of time to match the rate of sending data from Simulink to the discovery board.

- **UART 1 Configuration:** UART 1 is initialized and configured to receive data from PreScan. The UART parameters, such as baud rate, data format, and flow control, are set accordingly.

```
static void MX_USART1_UART_Init(void)
{
    huart1.Instance = USART1;
    huart1.Init.BaudRate = 115200;
    huart1.Init.WordLength = UART_WORDLENGTH_8B;
    huart1.Init.StopBits = UART_STOPBITS_1;
    huart1.Init.Parity = UART_PARITY_NONE;
    huart1.Init.Mode = UART_MODE_TX_RX;
    huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart1.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart1) != HAL_OK)
    {
        Error_Handler();
    }
}
```

Figure 63 :UART Configurations

- **Data Reception:** Task 1 continuously receives real-time simulation data from UART 1. It waits for incoming data and processes and stores it appropriately. The received data includes information from the 6 radars (range, relative velocity, and azimuth angle) and other data about the car (desired velocity, current velocity, heading angle, lane marker sensor readings for scan 1 and scan 2).
- **Queue Creation:** Two queues are created to facilitate communication with Task 2 and Task 3. These queues will be used to send the processed data to the respective tasks.

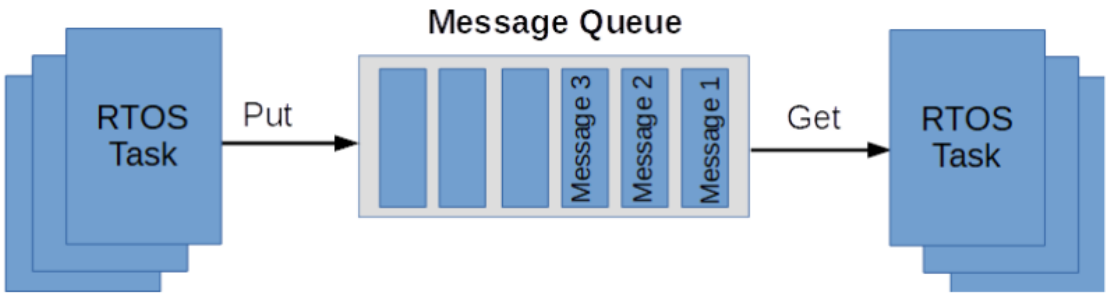


Figure 64 :Tasks Communications using UART

- **Mutex Creation:** A Mutex is created to protect UART 1, which is a shared resource. This Mutex ensures that only one task can access UART 1 at a time, preventing concurrent access and potential conflicts.

- **Queue Data Transfer:** Task 1 sends the processed data to Task 2 and Task 3 using the previously created queues. It packages the data appropriately and sends it via the respective queues for further processing in the respective tasks.
- **Semaphore Usage:** In Task 4, before sending data over UART 1 to Simulink after running the algorithm, it acquires the semaphore created by Task 1 to gain exclusive access to UART 1. This ensures that only Task 4 can use UART 1 while other tasks are blocked from accessing it. After completing the UART operation, Task 4 releases the semaphore, allowing other tasks, including Task 1, to access UART 1 again.

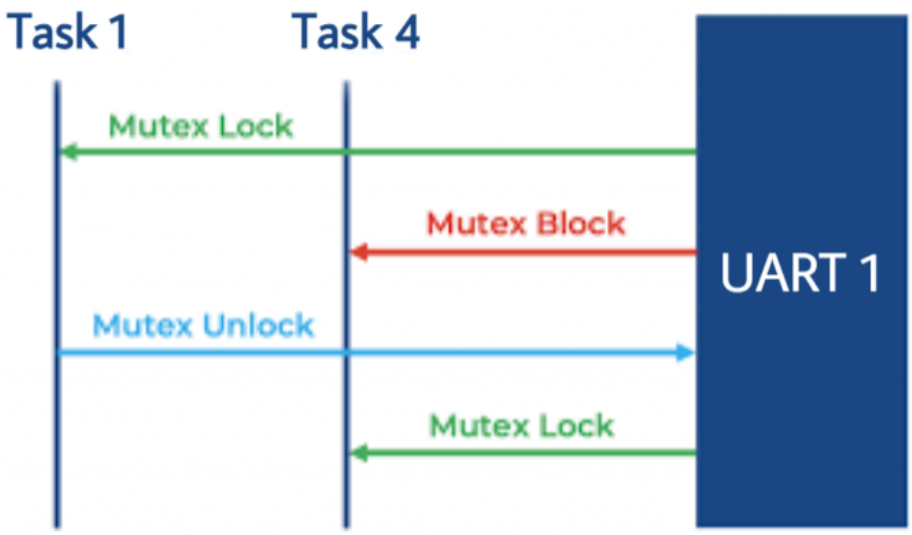


Figure 65 :Mutex for UART (Sharing Resource)

- **Blocking:** Task 1 gets blocked for a specific amount of time at the end of the task. This duration matches the rate of sending data from Simulink to the discovery board, ensuring synchronization with the data flow.

4.7.2. Task 2

In Task 2, the data received from Task 1 through the queue will be used to calculate the time to collision for the 6 used radars and set time to collision flags based on predefined thresholds for warning, partial-brake, and full-brake states. The calculations are performed as follows:

- **Data Reception:** Task 2 receives data from Task 1 through the queue. This data includes information about the range, relative velocity, and azimuth angle for the 6 used radars.
- **Time to Collision Calculation:** Task 2 calculates the time to collision for the long-range radar and short-range radar by dividing the relative range between the host vehicle and the detected vehicle by the relative velocity between them. This calculation gives an estimate of the time required for a collision to occur.

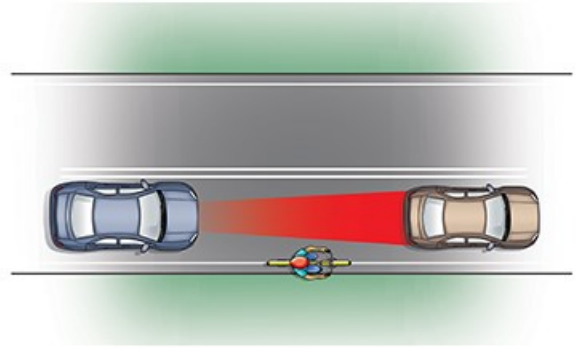


Figure 66 :Object detection using ABS

- **Time to Collision for Other Radars:** The time to collision for the other radars is calculated using the same approach as the long-range and short-range radars. However, an additional factor is considered by multiplying the calculated time to collision with the cosine of azimuth angle between the two vehicles. This accounts for the possibility of lane changes, as these radars are responsible for detecting lane change potential.

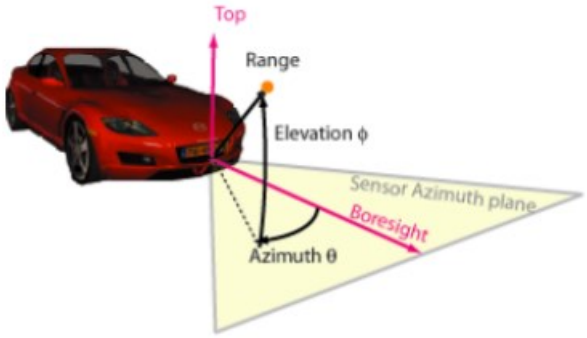


Figure 67 :Radar's Angles and Ranges

- **Comparison with Thresholds:** The calculated time to collision for each radar is compared with predefined thresholds. For example, if the time to collision of the long-range radar is lower than the warning threshold (3 seconds), the warning flag is set. Similarly, if the time to collision of the long-range radar is lower than the partial-brake threshold (2 seconds) or if the short-range radar is lower than the full-brake threshold (1 second), the corresponding flags are set.
- **Lane Change Detection:** The algorithm prioritizes moving to the right lane if possible when an object is detected in the current lane and the partial-brake or full-brake flags are set. To check the possibility of moving to the right lane, the time to collision of the front right radar is compared to the time to collision of the long-range radar. If the time to collision of the front right radar is lower, it indicates that the vehicle can safely move to the right lane. If not, the algorithm checks the possibility of moving to the left lane by comparing the time to collision of the front left radar to the time to collision of the long-range radar. If the time to collision of the front left radar is lower, it means that the vehicle can safely move to the left lane.

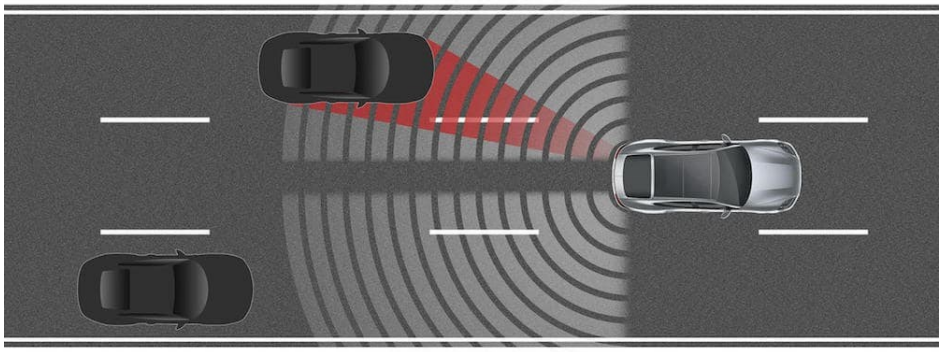


Figure 68 :Lane Change detection

- **Result Dispatch:** The resulting time to collision flags and lane change possibility flags are sent to Task 3 using a queue. This allows Task 3 to process the flags and take appropriate actions based on the received information.
- **Blocking:** Task 2 gets blocked for a specific amount of time, similar to the previous task, to maintain the sequential execution of tasks. This blocking period ensures synchronization and coordination with the overall system operation.

4.7.3. Task 3

In Task 3, data from Task 1 and flag values from Task 2 are received through two different queues. If the long-range radar and the front right and front left radars all read zero, the lane marker sensor takes responsibility for keeping the vehicle in its lane. The process is as follows:

Lane Marker Sensor Processing: The sensor's coordinates vector in the first scan are subtracted from the sensor's coordinates vector in the second scan, resulting in the sensor's vector. Similarly, the world's coordinates of the first intersection with lanes from the right in the first scan are subtracted from the first intersection with lanes from the right in the second scan, resulting in the world's vector. A cross-product is performed between the sensor's vector and the world's vector. The angular error parameter is calculated as the tan-inverse of the norm of the cross-product divided by the dot product between the sensor's vector and the world's vector **$\tan^{-1}$(norm (cross-product (sensor's vector, world's vector))/dot-product (sensor's vector, world's vector))**. The result is multiplied by the sign of the last element in the cross-product then multiplied by a gain value (e.g., 1250). This result is added to another lane monitoring parameter to obtain the steering angle needed to keep the vehicle in the same lane.

- Auto Braking System Algorithm: Task 3 begins executing the auto braking system algorithm. The algorithm checks for the following cases:
  - Case 1: If the full brake and partial brake flags are both set, the algorithm checks the turn right flag. If it is set, it means the vehicle can change lanes to the right adjacent lane after ensuring no other vehicle is approaching from behind at high speed. The vehicle changes lanes by applying a steering angle of **–tan-inverse (car length/lane width)**.

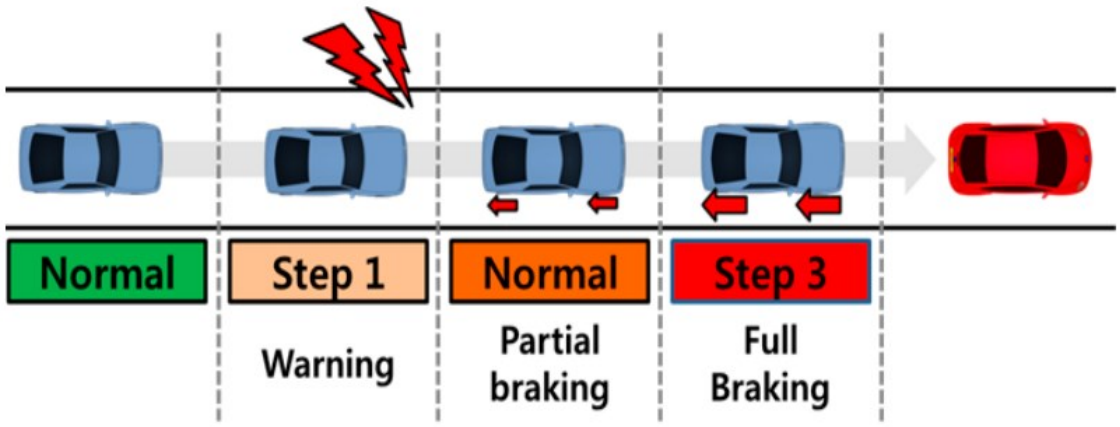


Figure 69 :Algorithm regions

If the turn right flag is not set, the algorithm checks the turn left flag. If it is set, it means the vehicle can change lanes to the left adjacent lane after ensuring no other vehicle is approaching from behind at high speed. The vehicle changes lanes by applying a steering angle of **tan-inverse (car length/lane width)**. If the turn left flag is not set, the vehicle applies the full brake force, which equals **150 bars.**

- b. Case 2: If the partial brake flag is set and the full brake flag is not set, the algorithm checks the turn right flag. If it is set, it means the vehicle can change lanes to the right adjacent lane after ensuring no other vehicle is approaching from behind at high speed. The vehicle changes lanes by applying a steering angle of **–tan-inverse (car length/lane width)**. If the turn right flag is not set, the algorithm checks the turn left flag. If it is set, it means the vehicle can change lanes to the left adjacent lane after ensuring no other vehicle is approaching from behind at high speed. The vehicle changes lanes by applying a steering angle of **tan-inverse (car length/lane width)**. If the turn left flag is not set, the vehicle applies the partial brake force, which equals **90 bars**.

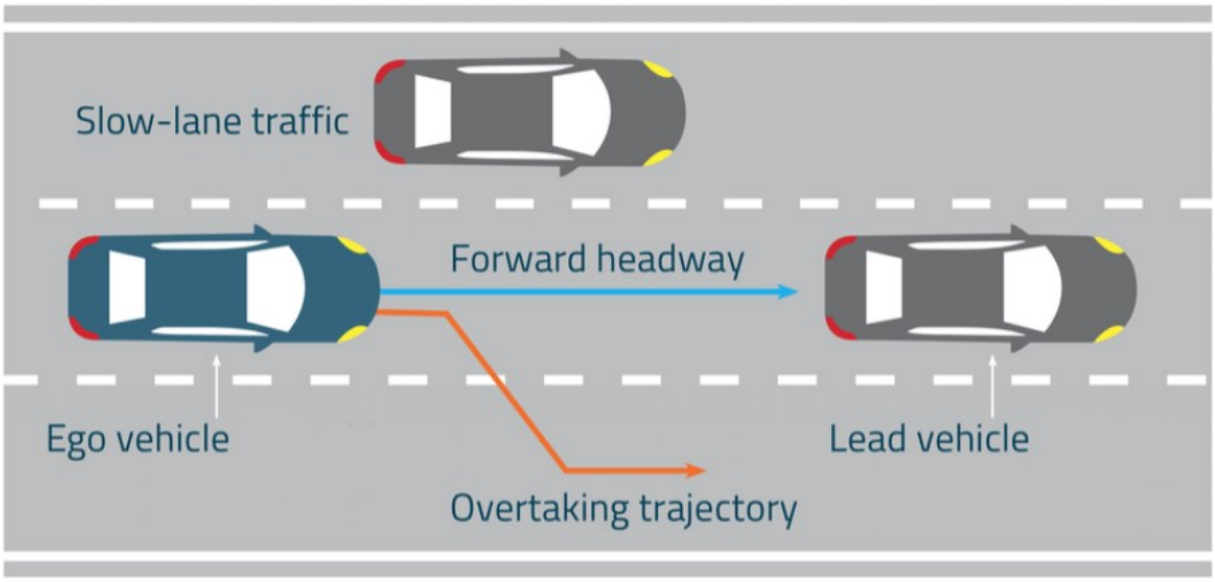


Figure 70 :Lane keeping to steering angles

If the lane changing option is chosen over the auto braking option, the vehicle will continue steering with the calculated angle until the heading increases by 10 degrees from the beginning of the lane change in the case of turning right. During this process, the lane number will increment, indicating the vehicle's transition to the right lane. Once the heading increase of 10 degrees is

reached, the responsibility of centering the vehicle in the new lane will be handed over to the lane marker sensor.

Similarly, in the case of turning left, the steering will continue until the heading decreases by 10 degrees from the beginning of the lane change. The lane number will decrement to indicate the vehicle's transition to the left lane. Once the heading decrease of 10 degrees is achieved, the lane marker sensor will take over the responsibility of centering the vehicle in its new lane.

These processes ensure that the vehicle smoothly transitions to the desired lane while maintaining proper alignment within that lane.

- c. Case 3: If the warning flag is set, the algorithm does not apply any brake force (brake force = 0) and lets the lane marker sensor choose the appropriate steering angle to keep the vehicle in its lane.
- d. Case 4: If none of the above cases are true, it means the vehicle is in normal mode. No braking force is applied, and the lane marker sensor is responsible for keeping the vehicle in its lane.

- Result Dispatch: The calculated data from the algorithm, including the brake force, steering angle, lane ID, lane keeping flag, lane marker angle, and control signal representing the vehicle's state (0 for normal state, 1 for warning state, 2 for partial brake state, and 3 for full brake state), are sent to the next task using a queue.
- Blocking: Task 3 gets blocked for a specific amount of time, similar to the previous tasks, to maintain sequential execution and synchronization with the overall system operation.

4.7.4. Task 4

In Task 4, the results from Task 3 are received through a queue. The result data includes the control signal, brake force, steering angle, lane ID, lane keeping flag, and lane marker angle. These results are used to form a UART frame, which is then sent to Simulink to trigger the appropriate action in Prescan.

- **Queue Reception:** Task 4 receives the results data from Task 3 through a queue. The data includes the control signal, brake force, steering angle, lane ID, lane keeping flag, and lane marker angle.

- **UART Frame Formation:** Task 4 uses the received results data to form a UART frame. The frame is structured based on the communication protocol between Task 4 and Simulink, ensuring compatibility and proper interpretation of the data by Simulink.
- **UART Transmission:** The formed UART frame is sent to Simulink. Task 4 initiates the transmission process over the UART interface, ensuring that the frame is delivered to the appropriate recipient in the Simulink environment. The UART communication protocol and parameters are configured appropriately for reliable and efficient data transmission.
- **Action in Simulink:** Simulink receives the UART frame from Task 4 and processes the data within the frame. Based on the received control signal, Simulink triggers the appropriate action in the Prescan simulation. This action may involve adjusting the vehicle dynamics, applying brakes, or changing the steering angle, among other possibilities, to simulate the desired behavior.

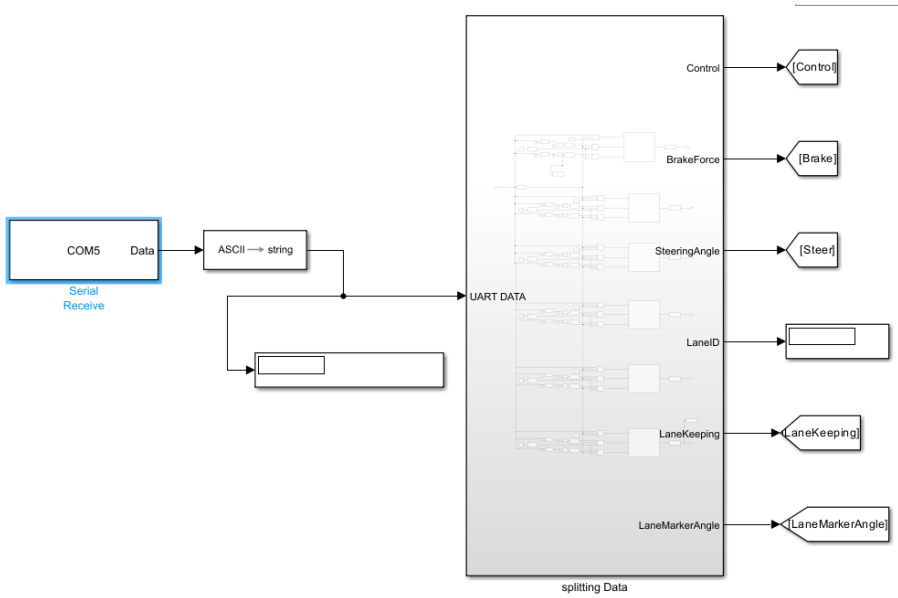


Figure 71 :UART frame splitting

- **Blocking:** Task 4 is blocked for a specific amount of time at the end, similar to the previous tasks, to maintain sequential execution and synchronization with the overall system operation. This blocking period ensures proper coordination with other tasks and the expected timing requirements of the system.

4.7.5. Task 5

In Task 5, which is initially a suspended task, its priority is set higher than all other tasks since it handles the case when the driver is drowsy. Task 5 runs only when a specific bit is transmitted from the drowsiness detection ECU (Raspberry Pi) to the discovery board using CAN protocol. The task is resumed when the bit is received, triggering an interrupt. The interrupt's ISR executes a callback function that clears the interrupt by reading the received bit from the CAN message, and then resumes Task 5.

- **Task Initialization:** Task 5 is initially set as a suspended task, ensuring it does not run until explicitly resumed.
- **Drowsiness Detection Bit:** The Raspberry Pi transmits a specific bit to the discovery board using CAN protocol, indicating the driver's drowsiness status. The first bit sent is always set to 1, indicating that the driver is initially starting to feel drowsy.

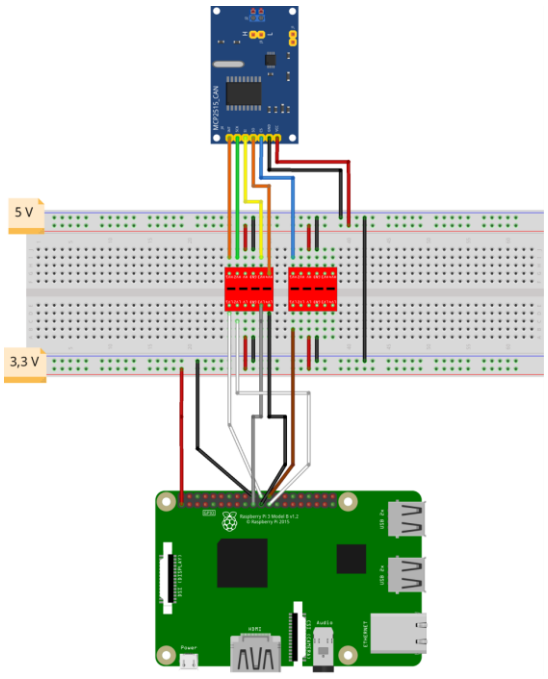


Figure 72 :Raspberry to CAN level shifting

- **Interrupt Handling:** When the discovery board receives the bit, an interrupt occurs, and the interrupt service routine (ISR) is executed. The ISR's callback function reads the received bit, clears the interrupt, and resumes Task 5.

- **Drowsy Driver Wake-Up:** Upon resuming, Task 5 checks the received bit. If it is equal to 1, it indicates that the driver is in the beginning stages of drowsiness. Task 5 turns on a low-frequency buzzer to gently wake up the driver without causing alarm or fear. After this, Task 5 is suspended again, and the four sequential tasks resume their execution with the buzzer activated.

- **Driver Response:** If the driver wakes up and responds to the buzzer, a bit equal to 0 is sent from the Raspberry Pi to the discovery board, indicating that the driver is in a condition that allows him to continue driving. Task 5 runs, turns off the buzzer, and then suspends again. The four sequential tasks resume their execution.

- **Non-Responsive Driver:** If the driver does not respond to the buzzer within a certain time period after receiving a bit equal to 1, the Raspberry Pi sends a bit equal to 2 to the discovery board, indicating that the driver is in a condition that does not allow him to continue driving, and the buzzer is set to its maximum frequency in an attempt to wake up the driver if he is asleep.

- **Task 5 Execution:** When Task 5 is resumed with the bit equal to 2, it executes the necessary actions. If the vehicle is in the far-right or far-left lane, Task 5 immediately applies the full brake force to stop the vehicle. If the vehicle is in one of the middle lanes, Task 5 is blocked for a specific time period, allowing the algorithm to change lanes immediately, if possible, until reaching the far-right or far-left lane. Once in the far-right or far-left lane, Task 5 applies the full brake force to stop the vehicle.

- **Safety Measures:** The application of full brake force or lane change operations should be performed while considering safety precautions, such as ensuring no other vehicles are approaching at high speeds from behind.

Chapter 5

Visualization

5. MATLAB GUI

5.1. What Is a GUI?

A graphical user interface (GUI) is a graphical display in one or more windows containing controls, called ***components***, that enable a user to perform interactive tasks. The user does not have to create a script or type commands at the command line to accomplish the tasks. Unlike coding programs to accomplish tasks, the user does not need to understand the details of how the tasks are performed.

GUI components can include menus, toolbars, push buttons, radio buttons, list boxes, and sliders—just to name a few. UIs created using MATLAB tools can also perform any type of computation, read and write data files, communicate with other UIs, and display data as tables or as plots.

The following figure illustrates a simple UI.

The UI contains these components:

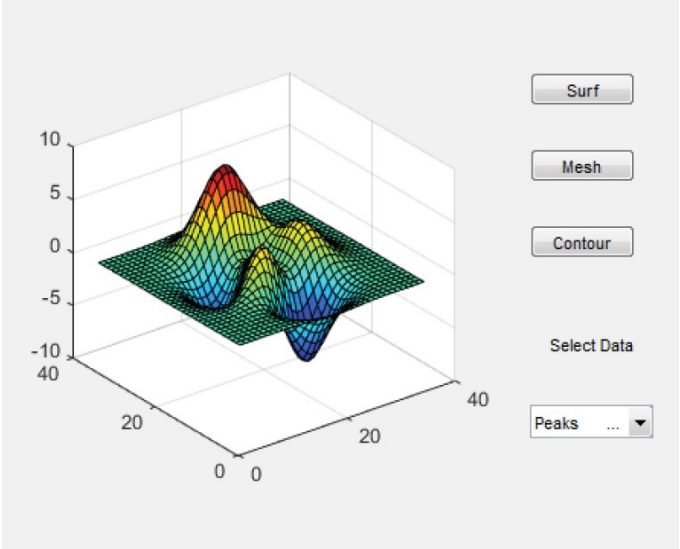


Figure 73: Simple GUI design.

• An axes component

• A pop-up menu listing three data sets that correspond to MATLAB functions: peaks, membrane, and sinc

• A static text component to label the pop-up menu

• Three buttons that provide different kinds of plots: surface, mesh, and contour

When you click a push button, the axes component displays the selected data set using

the specified type of 3-D plot.

5.2. How Does a GUI Work?

Typically, GUIs wait for a user to manipulate a control, and then respond to each user action in turn. Each control, and the GUI itself, has one or more ***callbacks***, named for the fact that they "call back" to MATLAB to ask it to do things. A particular user action, such as pressing a screen button, or passing the cursor over a component, triggers the execution of each callback. The GUI then responds to these. You, as the GUI creator, write callbacks that define what the components do to handle events.

This kind of programming is often referred to as ***event-driven*** programming. In eventdriven programming, callback execution is ***asynchronous***, that is, events external to the software trigger callback execution. In the case of MATLAB GUIs, most events are user interactions with the GUI, but the GUI can respond to other kinds of events as well, for example, the creation of a file or connecting a device to the computer.

You can code callbacks in two distinct ways:

• As MATLAB language functions stored in files

• As strings containing MATLAB expressions or commands (such as 'c = sqrt(a*a + b*b);'or 'print')

Using functions stored in code files as callbacks is preferable to using strings, because functions have access to arguments and are more powerful and flexible. You cannot use MATLAB scripts (sequences of statements stored in code files that do not define functions) as callbacks.

Although you can provide a callback with certain data and make it do anything you want, you cannot control when callbacks execute. That is, when your GUI is being used, you have no control over the sequence of events that trigger particular callbacks or what other callbacks might still be running at those times. This distinguishes event-driven programming from other types of control flow, for example, processing sequential data files.

5.3. Ways to Build MATLAB UIs

A MATLAB GUI is a figure window to which you add user-operated components. You can select, size, and position these components as you like. Using callbacks you can make the components do what you want when the user clicks or manipulates the components with keystrokes.

You can build MATLAB UIs in two ways:

• Create the GUI using Appdesigner.

This approach starts with a figure that you populate with components from within a graphic layout editor. Appdesigner creates an associated code file containing callbacks for the GUI and its components. Appdesigner saves both the figure (as a FIG-file) and the code file. You can launch your application from either file.

• Create the GUI programmatically

Using this approach, you create a code file that defines all component properties and behaviors. When a user executes the file, it creates a figure, populates it with components, and handles user interactions. Typically, the figure is not saved between sessions because the code in the file creates a new one each time it runs.

The code files of the two approaches look different. Programmatic GUI files are generally longer, because they explicitly define every property of the figure and its controls, as well as the callbacks. Appdesigner UIs define most of the properties within the figure itself. They store the definitions in its FIG-file rather than in its code file. The code file contains callbacks and other functions that initialize the UI when it opens.

You can create a GUI with APPDESIGNER and then modify it programmatically. However, you cannot create a UI programmatically and then modify it with Appdesigner.

The approach you choose depends on your experience, your preferences, and your goals.

Here are some ways to achieve specific goals.

Goal	Description of Approach
Create a dialog box	Call a function that creates predefined dialog box.
Create a UI containing a few components	It is often simpler to create UIs that contain only a few components programmatically. You can fully define each component with a single function call.
Create a moderately complex UI	Appdesigner simplifies the creation of moderately complex UIs.
Create a complex UI with many components, or one that interacts with another UI.	Creating complex UIs programmatically lets you control exact placement of the components and provides reproducibility.

Table 2: choices in App designer

5.4. Designing a GUI to display information about the vehicle.

5.4.1. Open a New UI in the Appdesigner Layout Editor

Type in the command window appdesigner to open the layout editor.

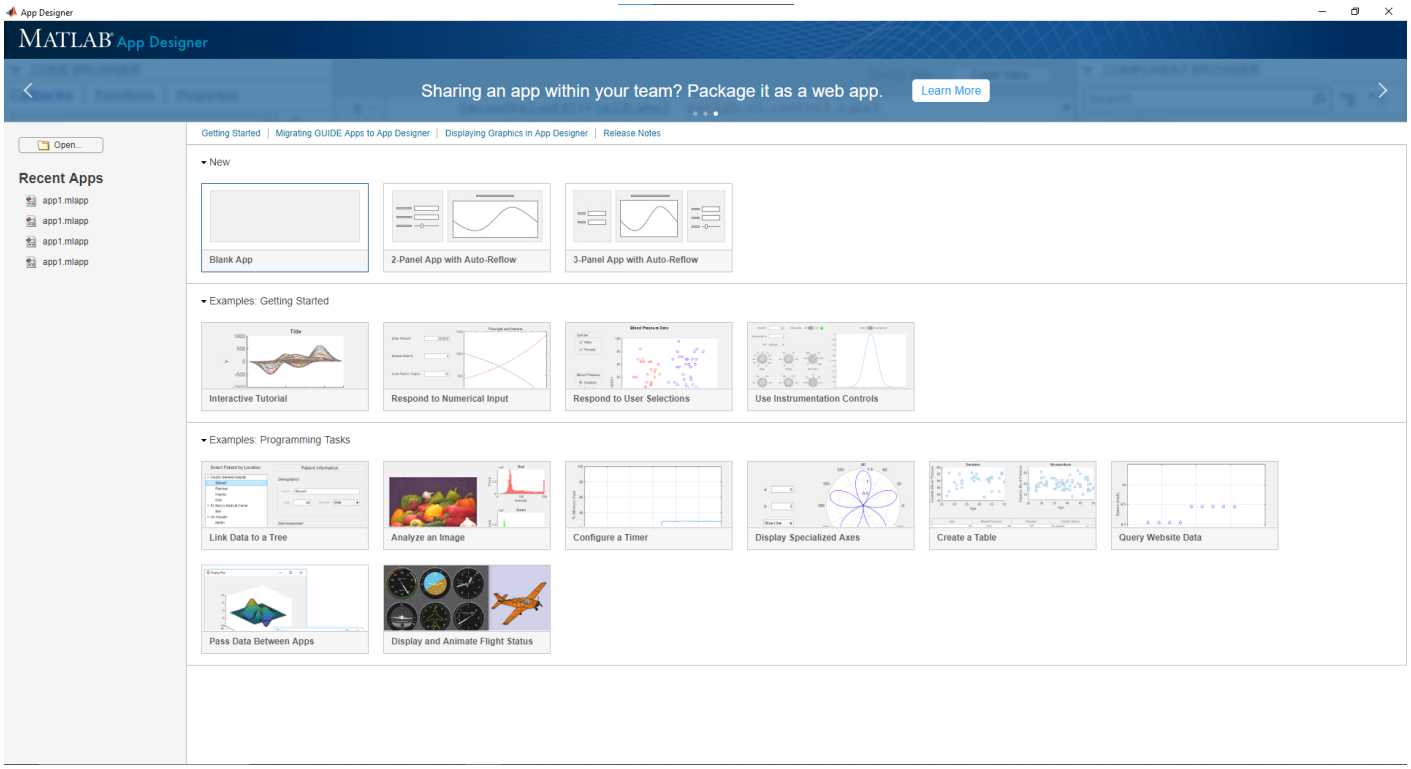


Figure 74: MATLAB Appdesigner.

5.4.2. Lay Out Apps in App Designer Design View

Design View in App Designer provides a rich set of layout tools for designing modern, professionallooking

applications. It also provides an extensive library of UI components, so you can create various

interactive features. Any changes you make in **Design View** are automatically reflected in **Code**

View. Thus, you can configure many aspects of your app without writing any code.

To add a component to your app, use one of these methods:

· Drag a component from the **Component Library** and drop it on the canvas.

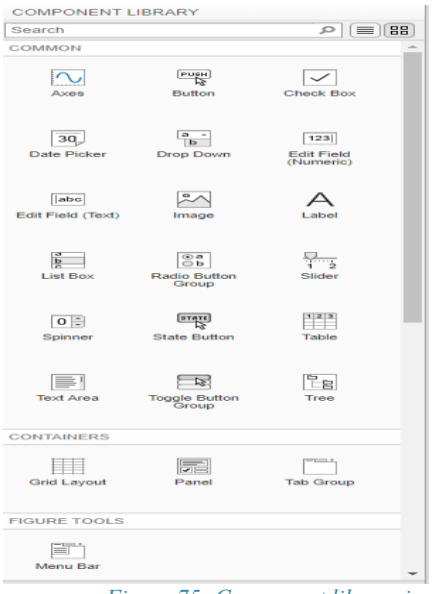


Figure 75: Component library in Appdesigner.

・Click a component in the **Component Library** and then move your cursor over the canvas. The cursor changes to a crosshair. Click your mouse to add the component to the canvas in its default size, or click and drag to size the component as you add it. Some components can only be added in their default size.

The name of the component appears in the **Component Browser** after you add it to the canvas. You can select components in either the canvas or the **Component Browser**. The selection occurs in both places simultaneously.

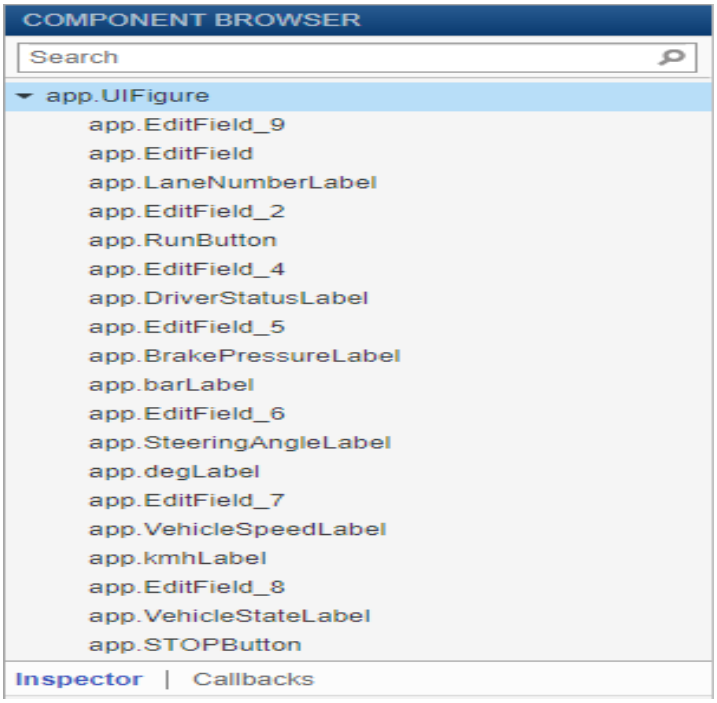


Figure 77: Appdesigner component browser.

Some components, such as edit fields and sliders, are grouped with a label when you drag them onto the canvas.

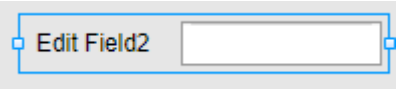


Figure 78: Edit field with a field to display numbers.

These labels do not appear in the **Component Browser** by default, but you can add them to the list by right-clicking anywhere in the **Component Browser** and selecting **Include component labels in Component Browser**. If you do not want the component to have a label, you can exclude it by

pressing and holding the **Ctrl** key as you drag the component onto the canvas. If you want to add a

label to a component without one, right-click the component and select **Add Label**.

If a component has a label, and you change the label text, the name of the component in the

Component Browser changes to match that text. You can customize the name of the component by double-clicking it and typing a new name.

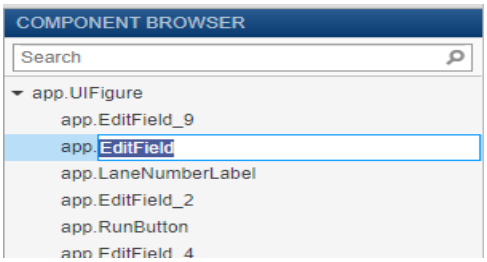


Figure 79: Edit the field of the function name when called back.

5.4.3. Align and Space Components

In Design View, you can arrange and resize components by dragging them on the canvas, or you can use the tools available in the Canvas tab of the toolstrip.

App Designer provides alignment hints to help you align components as you drag them in the canvas. Orange dotted lines passing through the centers of multiple components indicate that their centers are aligned. Orange solid lines at the edges indicate that the edges are aligned. Perpendicular lines indicate that a component is centered in its parent container.

As an alternative to dragging components on the canvas, you can align components using the tools in the **Align** section of the toolstrip.

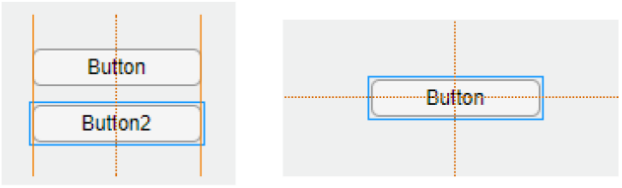


Figure 80: Align the buttons manually.

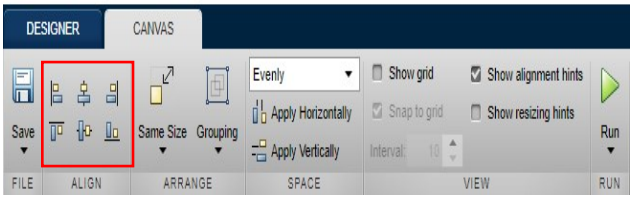


Figure 81: Align the buttons automatically by align in the canvas.

You can control the spacing among neighboring components using the tools in the **Space** section of the toolstrip. Select a group of three or more components, and then select an option from the dropdown list in the **Space** section of the toolstrip. The **Evenly** option distributes the space evenly within the space occupied by the components. The **20** option spaces the components 20 pixels apart. If you want to customize the number of pixels between the components, type a number into the drop-down list.

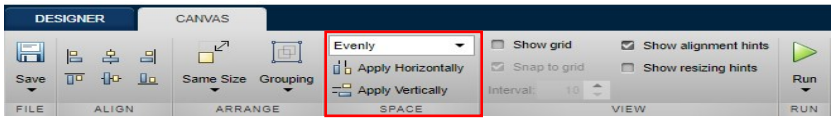


Figure 82: Controlling spacing.

5.5. Design the layout of the Appdesigner.

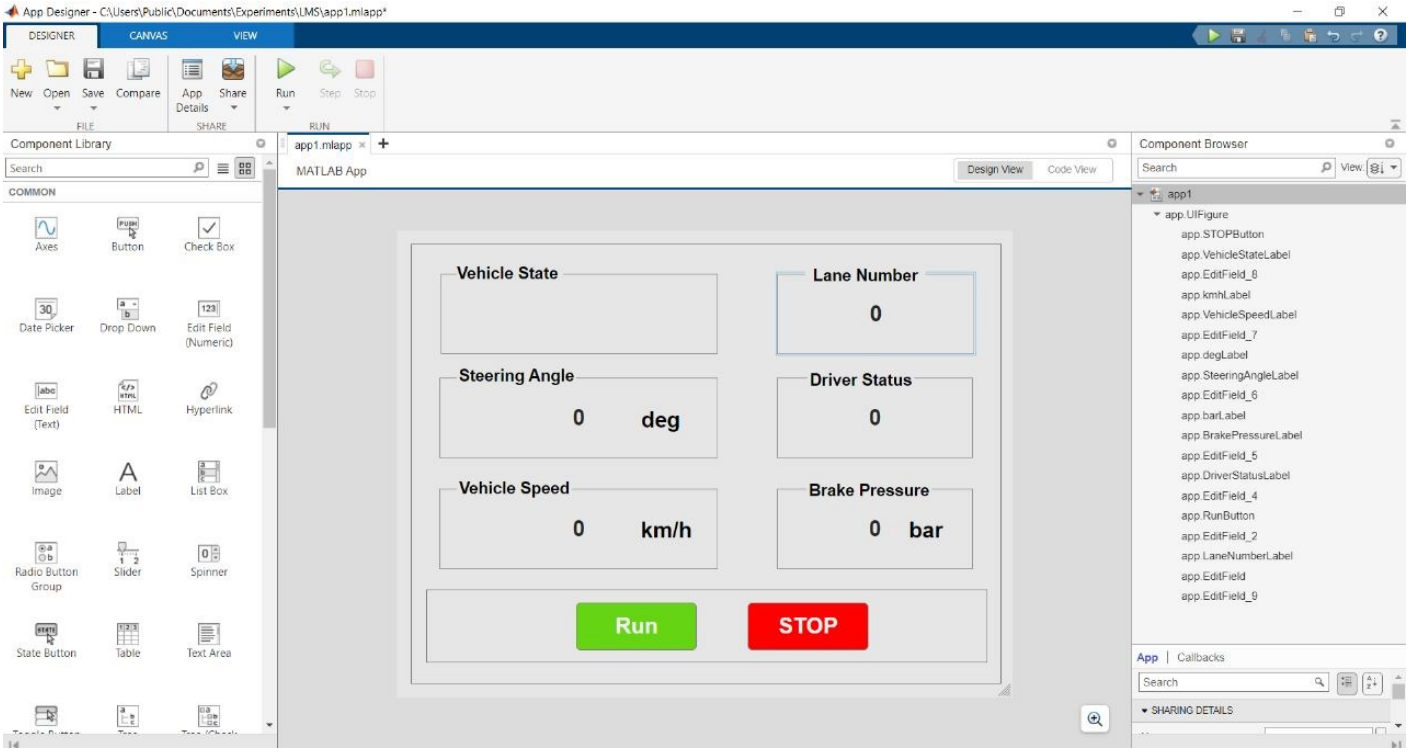


Figure 83: Final layout for the GUI.

5.5.1. Code view for the displaying the region of operation.

```
function updateGUI_Control(app, varargin)
      rto = get_param([bdroot,'/Audi_A8_Sedan_1/Display2'],'RuntimeObject');
      if(rto.InputPort(1).Data == 0)
      app.EditField_8.Value = 'Normal';
      app.EditField_8.FontColor = [0.00,1.00,0.00];
      elseif(rto.InputPort(1).Data == 1 || rto.InputPort(1).Data == 4)
      app.EditField_8.Value = 'Warning';
      app.EditField_8.FontColor = [0.878,0.906,0.133];
      elseif(rto.InputPort(1).Data == 2)
      app.EditField_8.Value = 'Partial Brake';
      app.EditField_8.FontColor = [0.93,0.69,0.13];
      elseif(rto.InputPort(1).Data == 3)
      app.EditField_8.Value = 'Full Brake';
      app.EditField_8.FontColor = [1.00,0.00,0.00];
      end
end
```

Figure 84: Code for displaying the region of operation.

Other functions are made to update the brake, steering angle, lane number, driver status and velocity.

```
% Button pushed function: RunButton
function RunButtonPushed(app, event)
    mdl = 'LMS_cs';
    open_system(mdl);
    out = sim(mdl,[0 60]);
end
```

Figure 85: Run button function to run the simulation once pressed.

```
% Button pushed function: STOPButton
function STOPButtonPushed(app, event)
    set_param('LMS_cs', 'SimulationCommand','stop');
end
end
```

Figure 86: Stop button function to stop the simulation once pressed

5.5.2. GUI when the simulation was started.

When running the simulation the vehicle state, lane number, steering angle, driver status, vehicle speed and brake pressure are displayed.

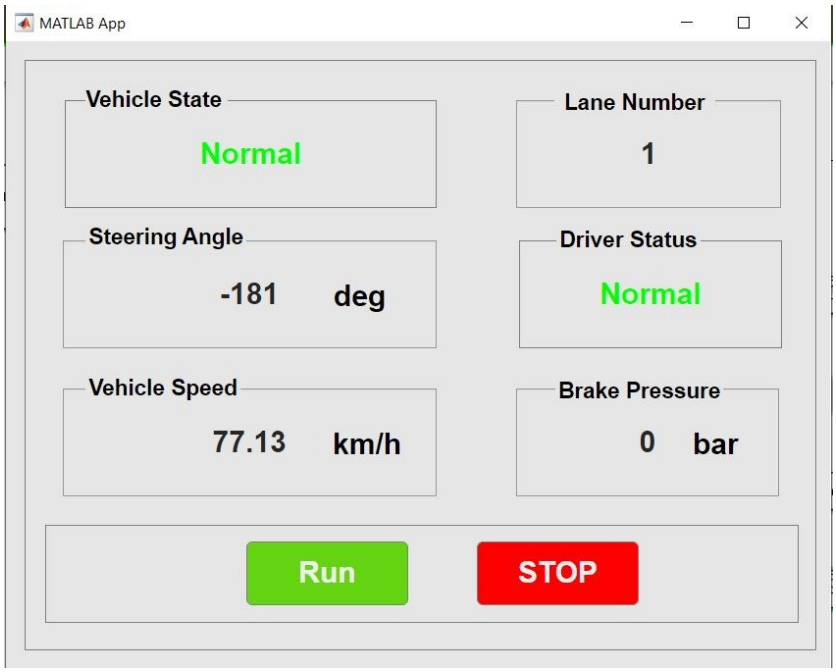


Figure 87: GUI at normal state.

When the driver is detected to be drowsy the alarm is set to alert the driver and awaken him before falling asleep.

Figure 88: GUI when the driver is detecting drowsy.

The car when it detects an object in warning state, it raises the warning flag and it's displayed on the GUI.

When the car in the range of full brake that's calculated according to time to collision, the brakes are pressed to achieve the maximum value of brake pressure to stop the car.

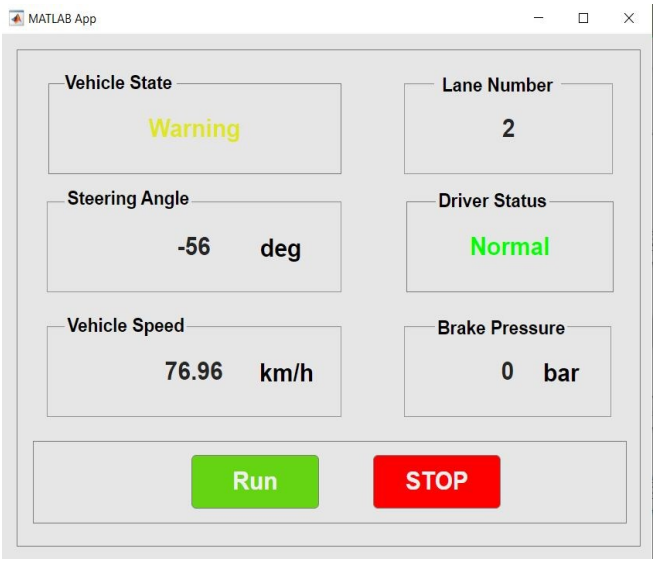


Figure 89: GUI at warning state.

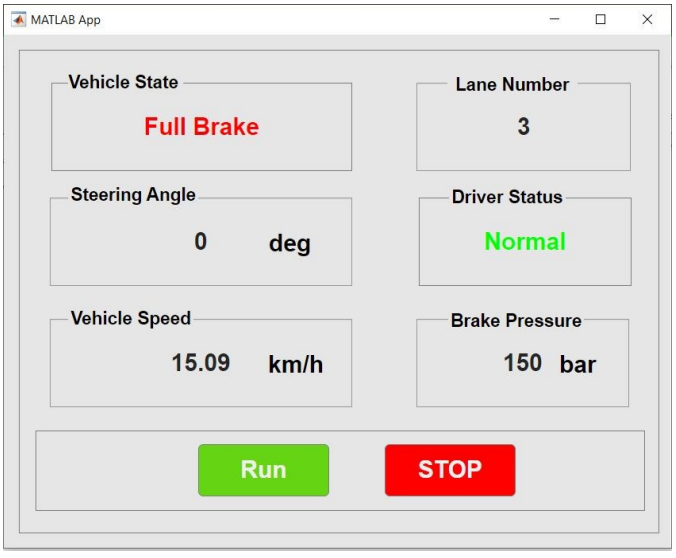


Figure 90: GUI at full brake state.

Once the driver is detected to be drowsy the algorithm will be applied to check first if the car in the middle lane if so, the car will change lane to stop beside the pavement and apply full brake.

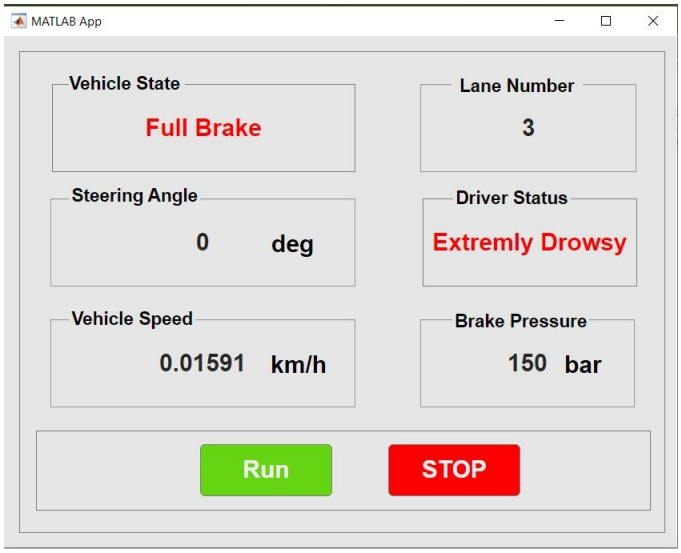


Figure 91: GUI displaying the case when applying full brake when the driver is detected to be extremely drowsy.

Chapter 6

Hardware

6. Hardware

6.1. Discovery Board

The Discovery Board is a development board based on the STM32F429ZI microcontroller from STMicroelectronics. It is designed to provide an easy and convenient platform for prototyping, testing, and evaluating applications using the STM32F429ZI microcontroller.

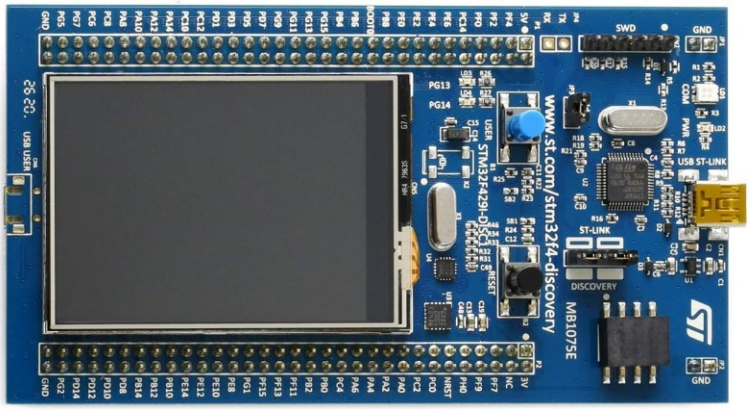


Figure 93::STM32f429 Discovery Kit

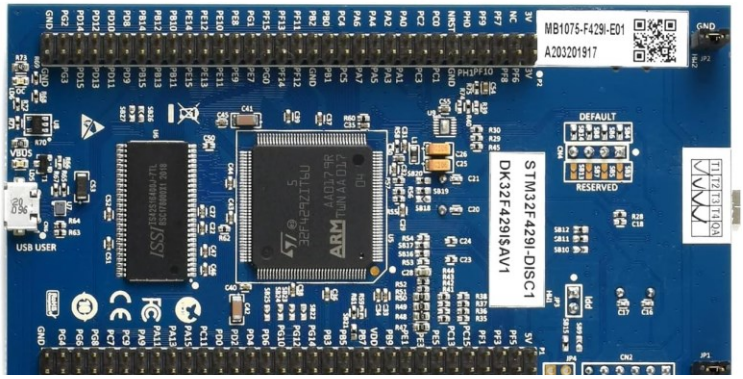


Figure 92 :STM32f429ZI MCU

Here are some key features and specifications of the Discovery Board:

- Microcontroller: The Discovery Board is built around the STM32F429ZI microcontroller, which belongs to the STM32F4 series. The microcontroller features a 32-bit ARM Cortex-M4 core running at a maximum frequency of 180 MHz and offers a wide range of peripherals and features.
- Memory: The STM32F429ZI microcontroller on the board typically comes with 2 MB of Flash memory for program storage and 256 KB of SRAM for data storage. Additionally, the board may have external memory interfaces for further expansion.
- Connectivity: The board offers various communication interfaces, including USB (Host, Device, or

OTG), UART, SPI, I2C, CAN, Ethernet, and more. These interfaces enable connectivity with other devices, sensors, and communication networks.

- Display: The Discovery Board provides a TFT-LCD display with a touchscreen. To be used in graphical user interface (GUI) development and interactive applications.
- Debugging and Programming: The Discovery Board usually includes an on-board ST-LINK/V2 debugger and programmer. This allows for easy programming, debugging, and flashing of the microcontroller using the ST-Link utility or integrated development environments (IDEs) such as STM32CubeIDE or Keil MDK.
- Power Supply: The board can be powered via USB or an external power supply. It often includes voltage regulators to provide stable and regulated power to the microcontroller and other components.
- Software Development: The Discovery Board is supported by various software development tools, including STM32Cube software development platform, STM32CubeIDE, Keil MDK, and other third-party IDEs. These tools provide libraries, examples, and drivers to facilitate software development for the microcontroller.

6.2. Raspberry Pi

The Raspberry Pi 3B is a single-board computer developed by the Raspberry Pi Foundation. It is part of the Raspberry Pi series, which aims to provide an affordable and accessible platform for learning programming and building various projects.

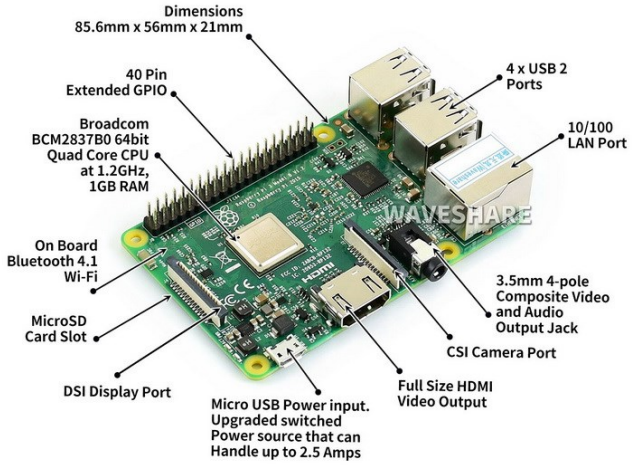


Figure 94 :Raspberry pi 3 Model B Board

Key specifications of the Raspberry Pi 3B include:

- Processor: The Raspberry Pi 3B is powered by a Broadcom BCM2837 SoC (System-on-Chip), which features a quad-core ARM Cortex-A53 CPU running at 1.2 GHz. This processor offers improved performance compared to previous Raspberry Pi models.

- Memory: It is equipped with 1GB LPDDR2 RAM, allowing for smooth multitasking and running various applications.
- Connectivity: The Raspberry Pi 3B provides several connectivity options, including:
    - Ethernet: It has a 10/100 Ethernet port for wired network connectivity.
    - Wi-Fi: Built-in 2.4 GHz 802.11n wireless LAN allows for wireless internet connectivity.
    - Bluetooth: It supports Bluetooth 4.2, enabling wireless communication with compatible devices.
- Storage: The Raspberry Pi 3B does not include built-in storage, but it features a microSD card slot for primary storage. The operating system and user data are typically stored on the microSD card.
- GPIO Pins: It offers a 40-pin General Purpose Input/Output (GPIO) header, allowing for connection to various external devices and sensors.
- USB: The Raspberry Pi 3B has four USB 2.0 ports, enabling connections to peripherals such as keyboards, mice, external storage devices, and more.
- Video Output: It supports HDMI output, allowing connection to a monitor or TV for display purposes. Additionally, it has a composite video (RCA) output for older displays.
- Operating System: The Raspberry Pi 3B is compatible with a variety of operating systems, including Raspbian (official OS for Raspberry Pi), Ubuntu, Windows 10 IoT Core, and more. These operating systems offer a range of software and development tools.
- Power: The board can be powered via a micro USB port using a 5V power adapter or through the GPIO pins.

6.3. USB Webcam

The USB Webcam is manufactured by Manhattan, a well-known brand in computer peripherals. The webcam is designed to provide high-definition video quality and is commonly used for video conferencing, online streaming, content creation, and other applications that require clear and crisp video footage.

Figure 95 :Manhattan 1080p USB Webcam

Here are some key features and specifications of the Manhattan 1080p USB Webcam:

- Resolution: The webcam supports a maximum resolution of 1080p (1920 x 1080 pixels), delivering sharp and detailed video.
- Plug-and-Play: It is a USB webcam, which means it can be easily connected to a computer or laptop via a USB port without requiring additional drivers or software installations. It follows the plug-and-play principle, allowing for quick and hassle-free setup.
- Built-in Microphone: The webcam features a built-in microphone, enabling clear audio capture during video calls or recordings. This eliminates the need for an external microphone in most scenarios.
- Adjustable Mounting: The webcam is typically equipped with an adjustable mounting mechanism, such as a clip or a stand, allowing for easy placement on different surfaces like monitors, laptops, or tripods.
- Compatibility: The Manhattan 1080p USB Webcam is designed to be compatible with various operating systems, including Windows, macOS, and Linux. It can work with popular video conferencing applications and streaming platforms.

6.4. MCP2515

The MCP2515 is a widely used stand-alone CAN controller IC developed by Microchip Technology. It provides a simple and cost-effective solution for implementing CAN communication in various embedded systems and automotive applications.

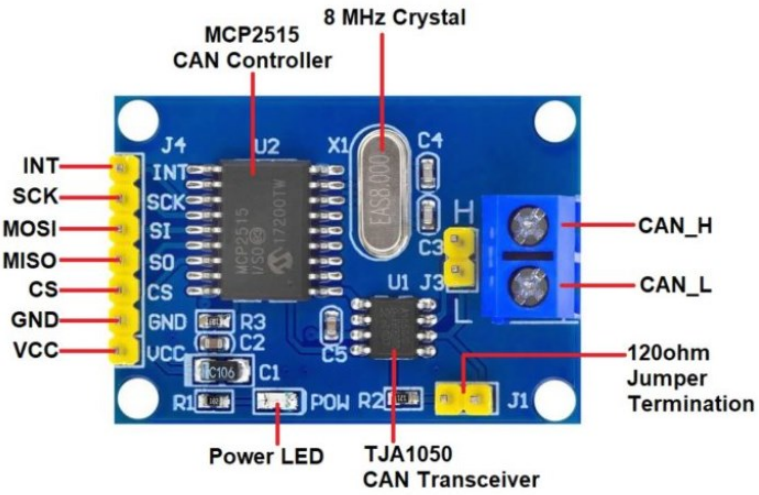


Figure 96 :MCP2515 External CAN Controller

Here are some key features and information about the MCP2515:

- Controller Area Network (CAN): The MCP2515 is specifically designed to support the CAN protocol, which is a widely used communication protocol for robust and reliable data exchange between electronic devices in automotive and industrial applications. CAN is known for its ability to handle real-time, high-speed, and fault-tolerant communication.
- SPI Interface: The MCP2515 utilizes a Serial Peripheral Interface (SPI) for communication with a

microcontroller or other devices. It supports both SPI mode 0 (CPOL=0, CPHA=0) and SPI mode 3 (CPOL=1, CPHA=1).

- Message Buffer: The MCP2515 provides two receive buffers and two transmit buffers to store and process CAN message. It supports both standard (11-bit) and extended (29-bit) identifier formats for CAN messages. The MCP2515 can handle up to 115 CAN message objects.
- Bit Timing Configuration: The MCP2515 allows for flexible configuration of the bit timing parameters for CAN communication. It supports various CAN bit rates, including standard rates such as 125 kbps, 250 kbps, and 500 kbps, as well as user-defined custom rates.
- Interrupt Capability: The MCP2515 features interrupt pins that can be connected to the microcontroller to indicate the status of the CAN bus, such as the arrival of a new message or an error condition.
- Error Handling and Diagnostics: The MCP2515 provides error detection and reporting mechanisms, including error flags and error counters, to handle and diagnose CAN bus errors.
- Power Supply and Package Options: The MCP2515 typically operates at a supply voltage of 2.7V to 5.5V.
- Software Support: Microchip provides a library and software drivers, called the MCP2515 CAN Controller Driver, to facilitate the integration and programming of the MCP2515 in embedded systems.

6.5. MCP2551

The MCP2551 is a widely used high-speed CAN transceiver IC developed by Microchip Technology. It is designed to provide a robust and reliable interface between a CAN controller and the physical CAN bus.

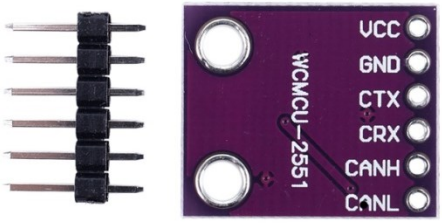


Figure 97 :MCP2551 CAN Transceiver

Here are some key features and information about the MCP2551:

- CAN Transceiver: The MCP2551 serves as a bidirectional interface between a CAN controller (such as the MCP2515) and the CAN bus. It supports both the ISO 11898 standard (high-speed CAN) and the Bosch CAN 2.0B protocol. The transceiver handles the electrical signaling, impedance matching, and noise filtering required for proper communication over the CAN bus.
- High-Speed Operation: The MCP2551 operates at high speeds, allowing for reliable communication in applications that require fast data transfer rates. It supports data rates up to 1 Mbps, making it

suitable for applications demanding high-speed CAN communication.

- Fault Protection and Transient Protection: The MCP2551 includes built-in protection features to safeguard against faults and transients that may occur on the CAN bus. It provides thermal shutdown protection, which prevents the transceiver from overheating during abnormal operating conditions. The transceiver also incorporates features like input protection against voltage transients and bus lines short-circuit protection.
- Low Power Consumption: The MCP2551 is designed to consume low power, making it suitable for applications where power efficiency is a concern. It features a low standby current mode to minimize power consumption when the transceiver is not actively transmitting or receiving data.
- Power Supply and Package Options: The MCP2551 typically operates at a supply voltage of 4.5V to 5.5V.
- Compatibility: The MCP2551 is compatible with various CAN controllers and microcontrollers available in the market. It can be used in automotive applications, industrial automation, medical devices, and other systems that utilize the CAN bus for communication.

6.6. Level Shifter

An 8-bit level shifter is a device that allows the conversion of digital signals between different voltage levels. It is commonly used to interface between microcontrollers or digital circuits operating at different voltage levels, ensuring proper communication and compatibility.

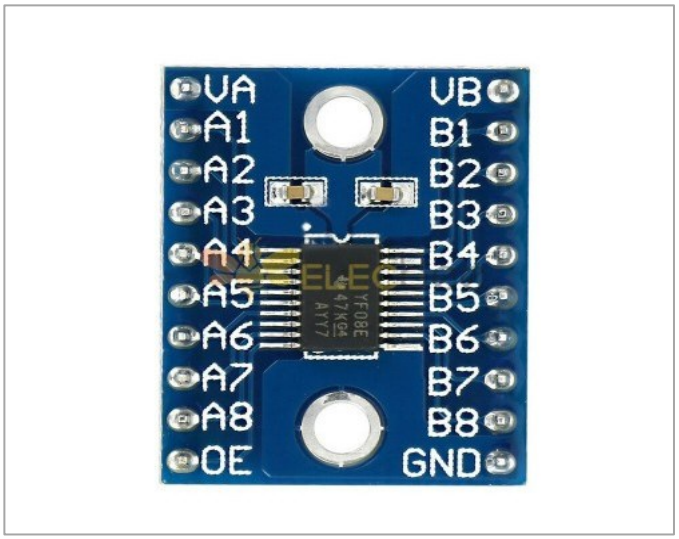


Figure 98 :8-bit level shifter

Here are some key points about an 8-bit level shifter:

- Voltage Level Conversion: An 8-bit level shifter facilitates the conversion of signals from one voltage level to another. For example, it can convert signals from 3.3V to 5V or vice versa. It ensures that the voltage levels of the input signals are compatible with the voltage levels required by the receiving device.

- Bidirectional Communication: An 8-bit level shifter can handle bidirectional communication, allowing data to be transmitted in both directions between devices operating at different voltage levels. It supports the conversion of both input and output signals.
- Parallel Interface: The "8-bit" in the term "8-bit level shifter" refers to the number of data lines that can be shifted simultaneously. It can handle 8 parallel data lines or signals simultaneously, making it suitable for applications requiring the conversion of multiple signals.
- Logic Level Translation: The level shifter translates the logic levels of the signals while maintaining the integrity of the data. It ensures that the logical high and low states of the input signals are correctly translated to the corresponding logical levels of the output signals.
- Voltage Compatibility: An 8-bit level shifter can typically support a wide range of voltage levels, such as 1.8V, 3.3V, 5V, and others commonly used in digital circuits. The specific voltage levels and ranges supported may vary depending on the particular level shifter model.

6.7. USB to TTL

A USB TTL (Transistor-Transistor Logic) adapter is a common interface device used to convert signals between USB and UART serial communication protocols. It allows a computer with a USB port to communicate with and control devices that use UART for serial communication.

Figure 99 :USB to serial TTL Converter

Here's an overview of USB TTL adapters for UART communication:

- Purpose: USB TTL adapters serve as a bridge between a computer's USB port and devices that use UART for serial communication. They enable bidirectional data transfer between the computer and the target device, allowing control, monitoring, and data exchange.
- USB Interface: USB TTL adapters have a USB connector on one end that connects to the computer's USB port. They are typically USB 2.0 or USB 3.0 compatible and support various baud rates for UART communication.
- UART Interface: The other end of the USB TTL adapter has pins or connectors for UART communication. These pins usually include Tx (Transmit), Rx (Receive), GND (Ground), and

sometimes additional control lines like RTS (Request to Send) and CTS (Clear to Send). The Tx and Rx pins are used for data transmission and reception between the computer and the target device.

- Voltage Levels: USB TTL adapters support different voltage levels for UART communication, such as 3.3V or 5V, depending on the target device's requirements. It's essential to select an adapter that matches the voltage levels supported by the target device to ensure proper communication and avoid potential damage.
- Driver and Software: USB TTL adapters usually require driver software to be installed on the computer to enable communication. The driver software allows the computer's operating system to recognize the adapter as a virtual COM port. Additionally, terminal emulator software, such as PuTTY or Tera Term, can be used to interact with the target device through the virtual COM port.

Chapter 7

Software

7. Software

7.1. PreScan

PreScan is a software tool used in the automotive industry for virtual simulation and testing of advanced driver assistance systems (ADAS) and autonomous vehicle technologies. It is developed by TASS International, a subsidiary of Siemens Digital Industries Software.

Here are some key points about PreScan:

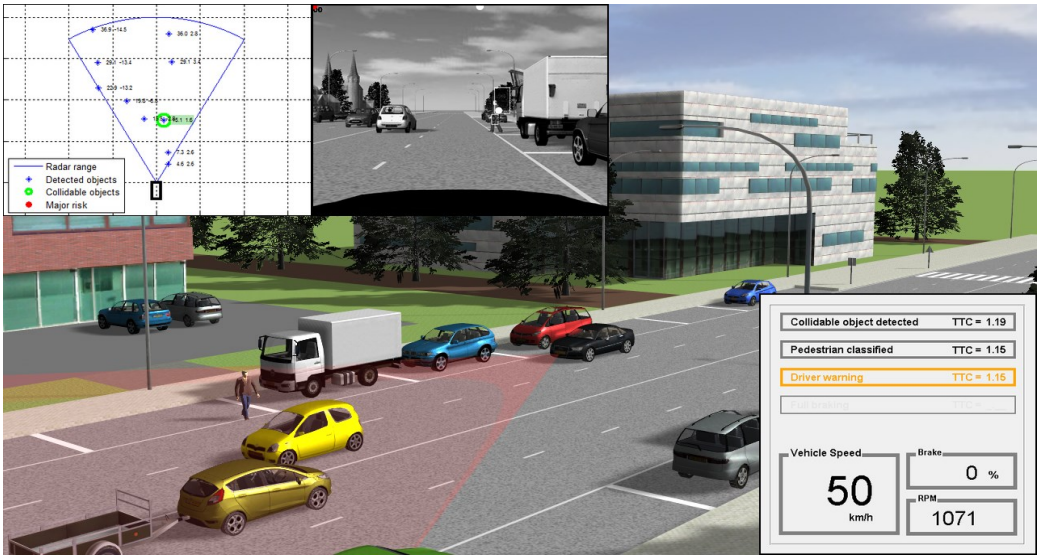


Figure 100 :Prescan Software Simulation Platform

- Virtual Simulation: PreScan enables the creation of virtual environments and scenarios to simulate real-world driving conditions. It allows engineers and researchers to test and evaluate ADAS and autonomous vehicle functionalities in a virtual environment before physical testing.
- Sensor Simulation: PreScan provides accurate simulation of various sensors used in ADAS and autonomous vehicles, such as radar, lidar, camera, and ultrasonic sensors. The simulated sensors generate realistic data, including distance measurements, point clouds, and images, allowing for sensor fusion and perception system testing.
- Vehicle Dynamics Simulation: PreScan incorporates a physics-based vehicle dynamics model to simulate the behavior of vehicles accurately. It considers factors like vehicle weight, aerodynamics, tire characteristics, and suspension properties to provide realistic vehicle motion and handling.
- Traffic and Environment Simulation: PreScan enables the creation of complex traffic scenarios, including multiple vehicles, pedestrians, and other road users. It simulates realistic traffic behavior, including lane-changing, merging, and intersection interactions. The software also models various environmental conditions, such as weather effects, lighting conditions, and road surface properties.
- Scenario Editor and Scripting: PreScan provides a scenario editor that allows users to define and

customize test scenarios. Users can create complex driving scenarios, define vehicle trajectories, set sensor configurations, and control the behavior of traffic participants. Scripting capabilities enable automation and customization of simulation scenarios.

- Sensor Data Analysis: PreScan offers tools for analyzing the sensor data generated during simulations. It allows users to visualize and evaluate sensor data, perform object detection and tracking, and validate the performance of ADAS algorithms.
- Integration with Development Workflows: PreScan integrates with other simulation and development tools used in the automotive industry, such as MATLAB/Simulink and CarSim. It supports the exchange of data and models between different software platforms, facilitating a comprehensive development workflow. PreScan is widely used by automotive manufacturers, suppliers, and research institutions for the development and validation of ADAS and autonomous vehicle systems. It helps to reduce development costs, accelerate time-to-market, and improve the safety and reliability of advanced automotive technologies.

7.2. Simulink

Simulink is a visual programming environment and simulation tool developed by MathWorks. It is widely used in the field of engineering and science, particularly in the areas of control systems, signal processing, robotics, and automotive systems.

Figure 101 :MATLAB & Simulink

Simulink can be integrated with PreScan to combine the power of virtual simulation in Prescan with the modeling and simulation capabilities of Simulink. This integration allows for a comprehensive development and testing environment for advanced driver assistance systems (ADAS) and autonomous vehicles.

Here are some key points about the integration of Simulink with PreScan:

- Co-Simulation: Simulink and PreScan can be connected through co-simulation interfaces, enabling the exchange of data and signals between the two environments. Simulink models can be connected to PreScan simulations, allowing for real-time interaction between the Simulink-controlled system and the virtual environment in PreScan.
- Model-in-the-Loop (MIL) Simulation: Simulink models can be imported into PreScan for Model-in-the-Loop (MIL) simulation. This allows users to test and validate the behavior of their Simulink models within the virtual environment provided by PreScan. The Simulink model can interact with the simulated sensors, actuators, and environment in PreScan, providing a realistic testing platform.
- Hardware-in-the-Loop (HIL) Simulation: Simulink models can also be used in conjunction with

PreScan for Hardware-in-the-Loop (HIL) simulation. In an HIL setup, the Simulink model runs on real-time hardware (such as a dSPACE or National Instruments platform) while interacting with the virtual environment simulated in PreScan. This enables the testing and validation of control algorithms and hardware interfaces with the virtual world.

- Sensor and Actuator Modeling: PreScan provides accurate simulation models for various sensors (such as radar, lidar, and cameras) and actuators (such as steering, braking, and acceleration). Simulink can utilize these models to incorporate sensor and actuator behaviors in the overall system simulation, ensuring realistic interactions between the Simulink model and the virtual environment.

7.3. STM Cube

STM Cube refers to the STM32Cube software development platform provided by STMicroelectronics. It is a comprehensive suite of tools, software libraries, and middleware that simplifies the development process for STM32 microcontrollers.

Figure 102 :STM Cube IDE for STM32 Microcontrollers

Here are some key points about STM32Cube:

- STM32 Microcontrollers: STM32Cube is primarily targeted towards developers working with the STM32 family of microcontrollers, which are based on the ARM Cortex-M processor architecture. STM32 microcontrollers are widely used in a variety of applications, including industrial automation, consumer electronics, IoT devices, and automotive systems.
- Integrated Development Environment (IDE): STM32Cube integrates with various popular development environments, including STM32CubeIDE and third-party IDEs such as Keil MDK and IAR Embedded Workbench. The IDE provides a user-friendly interface for coding, debugging, and flashing STM32 microcontrollers.
- Software Libraries and Middleware: STM32Cube includes a rich collection of software libraries and middleware components that provide ready-to-use functionality for common tasks and peripherals. These libraries and middleware cover a wide range of features, including communication protocols (UART, SPI, I2C, USB), file systems, graphic libraries, motor control, and more. The software libraries and middleware are designed to facilitate rapid development and reduce the time and effort required to implement complex functionality.
- Configuration Tools: STM32CubeMX is a graphical configuration tool provided as part of the

STM32Cube ecosystem. It allows developers to easily configure the STM32 microcontroller and its peripherals by selecting and configuring various parameters using a visual interface. STM32CubeMX generates initialization code based on the selected configurations, which can be seamlessly integrated into the development environment.

- Device-specific and Platform-independent: STM32Cube supports a wide range of STM32 microcontroller series, offering device-specific software packages for each series. The software development platform is also designed to be platform-independent, allowing developers to work with different IDEs, compilers, and operating systems.
- Hardware Abstraction Layer (HAL): STM32Cube includes the STM32 Hardware Abstraction Layer (HAL), which provides a consistent API for accessing and controlling the microcontroller peripherals. The HAL abstracts the low-level hardware details, enabling developers to write portable and reusable code across different STM32 microcontroller variants.
- Examples and Documentation: STM32Cube provides a wealth of examples and documentation, including application notes, user manuals, and code examples. These resources help developers get started quickly, understand the features and capabilities of the STM32 microcontrollers, and leverage the STM32Cube ecosystem effectively.

7.4. Tera Term

Tera Term is free and open-source terminal emulation software that allows users to connect to and communicate with remote devices using various communication protocols. It is commonly used in the field of embedded systems, network administration, and serial communication. Tera Term can be used in conjunction with UART (Universal Asynchronous Receiver-Transmitter) communication to establish a serial connection between a computer and a device that supports UART communication.

Here's a general overview of how you can use Tera Term with UART:

Figure 103 :Tera Term

- Connect the UART Hardware: Ensure that your computer has a serial port or USB-to-serial adapter to connect to the UART device. Connect the appropriate UART cables between your computer and the device, ensuring the correct pin connections (e.g., RX to TX and TX to RX).
- Configure Tera Term: Launch Tera Term on your computer. Select the correct serial port by navigating to the "Setup" menu and choosing "Serial Port". Choose the appropriate serial port number and configure the baud rate, data bits, stop bits, and parity settings according to the specifications of your UART device.
- Establish the Serial Connection: Once the serial port settings are configured, click on the "Open" button or select "Control" and then "Open" from the menu. Tera Term will attempt to establish a serial connection with the UART device using the specified settings.
- Interact with the UART Device: Once the serial connection is established, you can send commands or data to the UART device by typing directly into the Tera Term terminal window. Press the "Enter" key to send the command or data to the device. The device's responses or data will be displayed in the Tera Term terminal window.

7.5. PuTTY

PuTTY is a free and open-source terminal emulator, serial console, and network file transfer application. It is widely used for remote access and communication with various devices and systems.

PuTTY can be used to establish a remote connection with a Raspberry Pi, allowing you to access and control the Raspberry Pi's command-line interface (CLI) from a remote computer. Here's a step-by-step guide on how to use PuTTY with a Raspberry Pi:

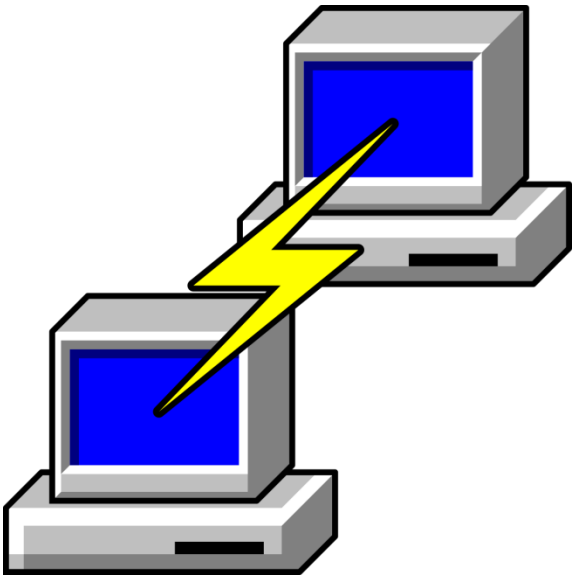


Figure 104 : PuTTY

- Connect the Raspberry Pi to the Network: Ensure that your Raspberry Pi is connected to the same network as your remote computer. This can be done via Ethernet or Wi-Fi.
- Obtain the IP Address of the Raspberry Pi: On the Raspberry Pi, open a terminal and type ifconfig to display the network configuration. Look for the IP address assigned to the Raspberry Pi. It will typically be under the "inet" field of the network interface connected to the network.
- Download and Install PuTTY: download the PuTTY installation package suitable for your operating system (Windows, Linux, or macOS). Run the PuTTY installer and follow the on-screen instructions to install PuTTY on your remote computer.
- Launch PuTTY: Once PuTTY is installed, launch the application.
- Configure PuTTY for SSH Connection: In the PuTTY Configuration window, enter the IP address of the Raspberry Pi in the "Host Name (or IP address)" field. Select the SSH protocol. Ensure that the port is set to 22 (the default SSH port). Choose a name for the PuTTY session, e.g., "Raspberry Pi SSH." Click the "Save" button to save the configuration for future use.
- Establish the SSH Connection: Double-click on the saved session (e.g., "Raspberry Pi SSH") or select it and click the "Load" button. Click the "Open" button to start the SSH connection.
- Provide Login Credentials: When prompted, enter the username and password for your Raspberry Pi.

The default username for a Raspberry Pi is usually "pi," and the default password is "raspberry" unless you have changed them.

- Access and Control Raspberry Pi: After successful authentication, you will have a command-line interface to interact with the Raspberry Pi. You can execute commands, run programs, edit files, and perform various tasks on the Raspberry Pi using the PuTTY terminal window.

7.6. VNC

VNC (Virtual Network Computing) is a graphical desktop sharing system that allows you to remotely control a computer's graphical desktop interface over a network connection. It enables you to access and interact with a remote computer's graphical user interface (GUI) as if you were sitting in front of it.

Figure 105 :VNC Viewer

VNC can be used with a Raspberry Pi to remotely access and control its graphical desktop interface. By setting up VNC on your Raspberry Pi, you can access its GUI from a remote computer, allowing you to interact with the Raspberry Pi as if you were physically sitting in front of it.

Here's how you can use VNC with a Raspberry Pi:

- Install and Enable VNC Server on the Raspberry Pi: On your Raspberry Pi, open a terminal or connect via SSH. Install the VNC server software. One popular option is RealVNC, which offers a free version called "VNC Connect". Follow the installation instructions provided by the VNC server software to complete the installation process. Enable the VNC server on the Raspberry Pi by configuring the VNC server software. This typically involves setting a password and adjusting other settings as required.

Chapter 8

Universal synchronous asynchronous receiver transmitter(USART)

8. USART

8.1. Introduction

The universal synchronous asynchronous receiver transmitter (USART) offers a flexible means of full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format. The USART offers a very wide range of baud rates using a fractional baud rate generator.

It supports synchronous one-way communication and half-duplex single wirecommunication. It also supports the LIN (local interconnection network), Smartcard Protocol and IrDA (infrared data association) SIR ENDEC specifications, and modem operations (CTS/RTS). It allows multiprocessor communication. High speed data communication is possible by using the DMA for multibuffering configuration.

8.2. USART main features in Discovery board (stm32f429ZI)

Full duplex, asynchronous communications

- NRZ standard format (Mark/Space)
- Configurable oversampling method by 16 or by 8 to give flexibility between speed andclock tolerance
- Fractional baud rate generator systems
    - Common programmable transmit and receive baud rate (refer to the datasheets for the value of the baud rate at the maximum APB frequency.
- Programmable data word length (8 or 9 bits)
- Configurable stop bits - support for 1 or 2 stop bits
- LIN Master Synchronous Break send capability and LIN slave break detection capability.
    - 13-bit break generation and 10/11bit break detection when USART is hardware configured for LIN.
- Transmitter clock output for synchronous transmission
- IrDA SIR encoder decoder

o Support for 3/16-bit duration for normal mode

- Smartcard emulation capability
    - The Smartcard interface supports the asynchronous protocol Smartcards as defined in the ISO 7816-3 standards.

0.5, 1.5 stop bits for Smartcard operation

- Single-wire half-duplex communication.
- Configurable multi buffer communication using DMA (direct memory access).
    - Buffering of received/transmitted bytes in reserved SRAM using centralized DMA.
- Separate enable bits for transmitter and receiver.

Universal synchronous asynchronous receiver transmitter (USART) RM0090

966/1751 RM0090 Rev 19

- Transfer detection flags:
    - Receive buffer full
    - Transmit buffer empty
    - End of transmission flags
- Parity control:
    - Transmits parity bit
    - Checks parity of received data byte
- Four error detection flags:
    - Overrun error
    - Noise detection
    - Frame error
    - Parity error
- Ten interrupt sources with flags:
    - CTS changes
    - LIN break detection
    - Transmit data register empty
    - Transmission complete
    - Receive data register full
    - Idle line received
    - Overrun error
        - Framing error
        - Noise error
        - Parity error
- Multiprocessor communication - enter into mute mode if address match does not occur

- Wake up from mute mode (by idle line detection or address mark detection)
- Two receiver wakeup modes: Address bit (MSB, 9th bit), Idle line

8.3. USART character description

Word length may be selected as either 8 or 9 bits by programming the M bit in the USART_CR1 register.

The TX pin is in low state during the start bit. It is in high state during the stop bit.

An ***Idle character*** is interpreted as an entire frame of "1s" followed by the start bit of thenext frame which contains data (The number of "1" 's will include the number of stop bits).

A ***Break character*** is interpreted on receiving "0s" for a frame period. At the end of thebreak frame the transmitter inserts either 1 or 2 stop bits (logic "1" bit) to acknowledge the start bit.

Transmission and reception are driven by a common baud rate generator, the clock for each is generated when the enable bit is set respectively for the transmitter and receiver.

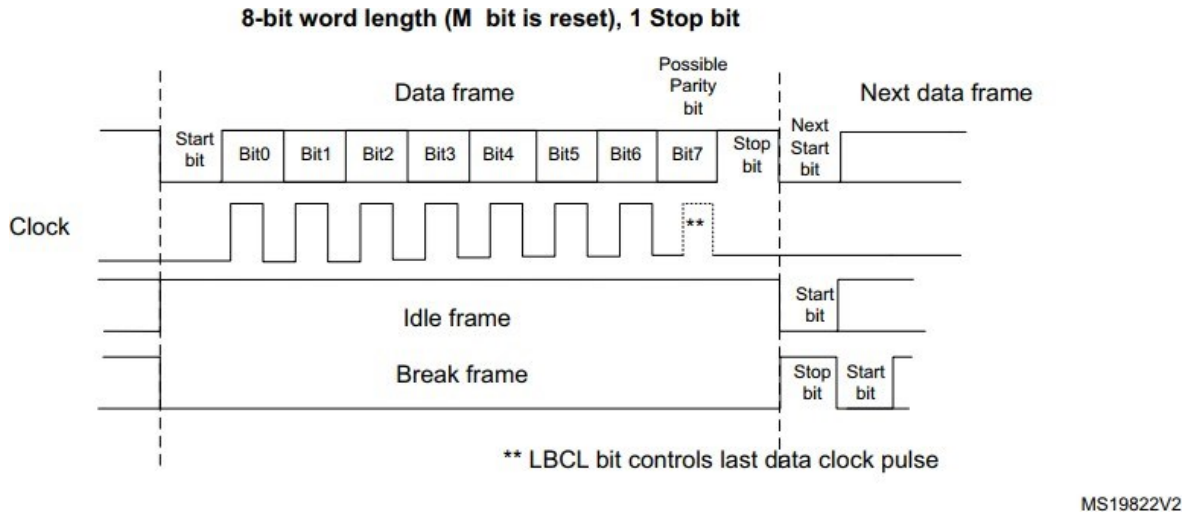


Figure 106 :Word length programming

8.3.1. Transmitter

The transmitter can send data words of either 8 or 9 bits depending on the M bit status.When the transmit enable bit (TE) is set, the data in the transmit shift register is output on the TX pin and the corresponding clock pulses are output on the CK pin. **Character transmission**

During an USART transmission, data shifts out least significant bit first on the TX pin. In thismode, the USART_DR register consists of a buffer (TDR) between the internal bus and the transmit shift register.

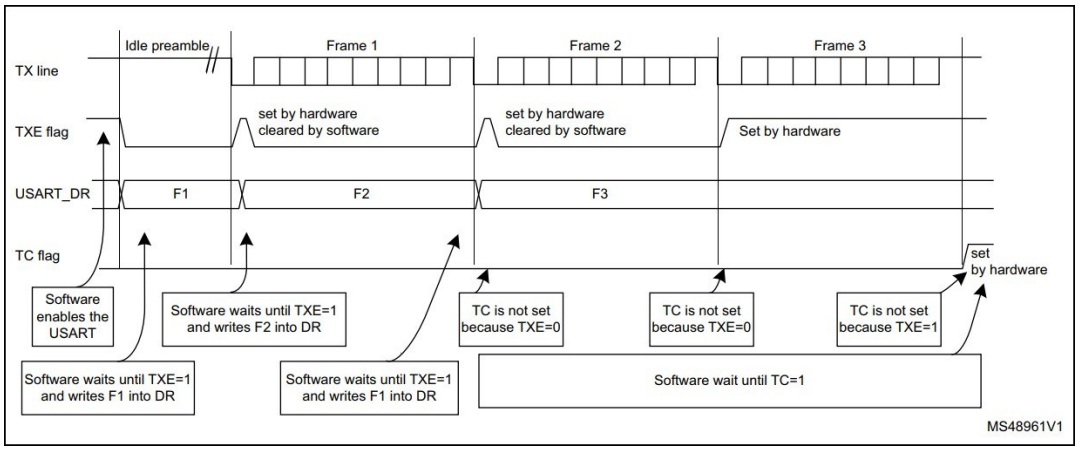


Figure 107 :TC/TXE behavior when transmitting

8.3.2. Receiver

The USART can receive data words of either 8 or 9 bits depending on the M bit in the USART_CR1 register.

Start bit detection

The start bit detection sequence is the same when oversampling by 16 or by 8.In the USART, the start bit is detected when a specific sequence of samples is recognized. This sequence is: 1 1 1 0 X 0 X 0 X 0 0 0 0.

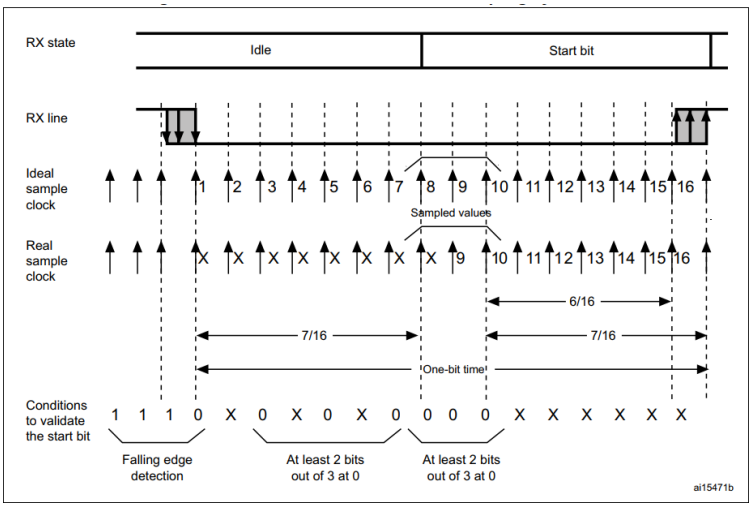


Figure 108 :Start bit detection when oversampling by 16 or 8

Character reception

During an USART reception, data shifts in least significant bit first through the RX pin. In thismode, the USART_DR register consists of a buffer (RDR) between the internal bus and the received shift register.

Procedure:

1. Enable the USART by writing the UE bit in USART_CR1 register to 1.
2. Program the M bit in USART_CR1 to define the word length.
3. Program the number of stop bits in USART_CR2.
4. Select DMA enable (DMAR) in USART_CR3 if multibuffer communication is to takeplace. Configure the DMA register as explained in multibuffer communication. STEP 3
5. Select the desired baud rate using the baud rate register USART_BRR
6. Set the RE bit USART_CR1. This enables the receiver which begins searching for a start bit.

When a character is received

- The RXNE bit is set. It indicates that the content of the shift register is transferred to the RDR. In other words, data has been received and can be read (as well as itsassociated error flags).
- An interrupt is generated if the RXNEIE bit is set.
- The error flags can be set if a frame error, noise or an overrun error has been detected during reception.
- In multibuffer, RXNE is set after every byte received and is cleared by the DMA read to the Data Register.
- In single buffer mode, clearing the RXNE bit is performed by a software read to the USART_DR register. The RXNE flag can also be cleared by writing a zero to it. The RXNE bit must be cleared before the end of the reception of the next character to avoid an overrun error.

Note: The RE bit should not be reset while receiving data. If the RE bit is disabled during reception, the reception of the current byte will be aborted.

Break character

When a break character is received, the USART handles it as a framing error.

Idle character

When an idle frame is detected, there is the same procedure as a data received character plus an interrupt if the IDLEIE bit is set.

Selecting the proper oversampling method

The receiver implements different user-configurable oversampling techniques (except in synchronous mode) for data recovery by discriminating between valid incoming data and noise.

The oversampling method can be selected by programming the OVER8 bit in the USART_CR1 register and can be either 16 or 8 times the baud rate clock Depending on the application:

- select oversampling by 8 (OVER8=1) to achieve higher speed (up to fPCLK/8). In this case the maximum receiver tolerance to clock deviation is reduced
- select oversampling by 16 (OVER8=0) to increase the tolerance of the receiver to clock deviations. In this case, the maximum speed is limited to maximum fPCLK/16.

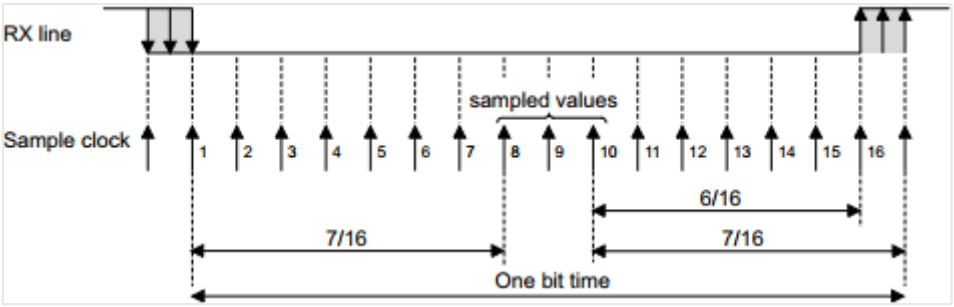


Figure 109 :Data sampling when oversampling by 16

Framing error

A framing error is detected when:

The stop bit is not recognized on reception at the expected time, following either a desynchronization or excessive noise.

When the framing error is detected:

- The FE bit is set by hardware
- The invalid data is transferred from the Shift register to the USART_DR register.
- No interrupt is generated in case of single byte communication. However, this bit rises at the same time as the RXNE bit which itself generates an interrupt. In case of multi buffer communication an interrupt will be issued if the EIE bit is set in the USART_CR3 register.

The FE bit is reset by a USART_SR register read operation followed by a USART_DRregister read operation.

8.3.3. USART mode configuration

USART modes	USART 1	USART 2	USART 3	UART4	UART5	USART 6
Asynchronous mode	X	X	X	X	X	X
Hardware flow control	X	X	X	NA	NA	X
Multibuffer communication (DMA)	X	X	X	X	X	X
Multiprocessor communication	X	X	X	X	X	X
Synchronous	X	X	X	NA	NA	X
Smartcard	X	X	X	NA	NA	X
Half-duplex (single-wire mode)	X	X	X	X	X	X
IrDA	X	X	X	X	X	X
LIN	X	X	X	X	X	X

1. X = supported; NA = not applicable.

Figure 110 :USART mode configuration:

8.4. USART registers

8.4.1. Status Register (USART_SR)

Address offset: 0x00

Reset value: 0x0000 00C0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

Figure 111 : Status Register

Bit 7 TXE: Transmit data register

This bit is set by hardware when the content of the TDR register has been transferred into

the shift register. An interrupt is generated if the TXEIE bit =1 in the USART_CR1 register. It is cleared by writing in the USART_DR register.

0: Data is not transferred to the shift register
1: Data is transferred to the shift register

Note: *This bit is used during single buffer transmission.*

Bit 6 TC: Transmission complete

This bit is set by hardware if the transmission of a frame containing data is complete and if TXE is set. An interrupt is generated if TCIE=1 is in the USART_CR1 register. It is cleared by a software sequence (a read from the USART_SR register followed by a write to the USART_DR register). The TC bit can

also be cleared by writing a '0' to it. This clearing sequence is recommended only for multibuffer communication.

0: Transmission is not complete
1: Transmission is complete

8.4.2. Data register (USART_DR)

DR [8:0]: Data value Contains the Received or Transmitted data character, depending on whether it is read from or written to.

The Data register performs a double function (read and write) since it is composed of two registers, one for transmission (TDR) and one for reception (RDR)

The TDR register provides the parallel interface between the internal bus and the output shift register.

The RDR register provides the parallel interface between the input shift register and the internal bus.

When transmitting while the parity enabled (PCE bit set to 1 in the USART_CR1 register), the value written in the MSB (bit 7 or bit 8 depending on the data length) has no effect because it is replaced by the parity.

When receiving while the parity enabled, the value read in the MSB bit is the received parity bit.

8.4.3. Baud rate register (USART_BRR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 112 :Baud rate register

Bits 31:16 Reserved, must be kept at reset value

Bits 15:4 DIV_Mantissa [11:0]: mantissa of USARTDIV.

These 12 bits define the mantissa of the **USART Divider (USARTDIV) Bits 3:0 DIV_Fraction [3:0]:** fraction of **USARTDIV**.

These 4 bits define the fraction of the **USART Divider (USARTDIV)**. When **OVER8=1**, the **DIV_Fraction3** bit is not considered and must be kept cleared.

8.4.4. Control registers 1 (USART_CR1)

Address offset: 0x0C

Reset value: 0x0000 0000

Bit 13 UE: USART enable

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	Reserved	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
rw	Res.	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 113 : Control registers 1

When this bit is cleared, the USART presales and outputs are stopped and the end of the current byte transfer in order to reduce power consumption (This bit is set and cleared by software).

0: USART prescaler and outputs disabled.
1: USART enabled.

Bit 3 TE: Transmitter enable This bit enables the transmitter, it is set and cleared by software.

0: Transmitter is disabled

1: Transmitter is enabled

Note: During transmission, a "0" pulse on the TE bit ("0" followed by "1") sends a preamble (idle line) after the current word, except in smartcard mode.
When TE is set, there is a 1 bit-time delay before the transmission starts.

Bit 2 RE: Receiver enable This bit enables the receiver. It is set and cleared by software.

8.4.5. Control register2 (USART_CR2)

Address offset: 0x10

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LINEN	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.	ADD[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

Figure 114 : Control register2

Reset value: 0x0000 0000

Bits 13:12 STOP: STOP bits These bits are used for programming the stop bits.

00: 1 Stop bit

01: 0.5 Stop bit

10: 2 Stop bits

11: 1.5 Stop bit

8.4.6. Control register3 (USART_CR3)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				ONEBIT	CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 115 : Control register3

8.4.7. Guard time and prescaler register (USART_GTPR)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GT[7:0]								PSC[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 116 :Guard time and prescaler register

8.4.8. USART register map

Table 149. USART register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	USART_SR	Reserved																						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
	Reset value																							0	0	1	1	0	0	0	0	0	0
0x04	USART_DR	Reserved																						DR[8:0]									
	Reset value																							0	0	0	0	0	0	0	0	0	0
0x08	USART_BRR	Reserved																DIV_Mantissa[15:4]												DIV_Fraction [3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	USART_CR1	Reserved																OVER8	Reserved	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
	Reset value																	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	USART_CR2	Reserved																	LINEN	STOP [1:0]		CLKEN	CPOL	CPHA	LBCL	Reserved	LBDIE	LBDL	Reserved	ADD[3:0]			
	Reset value																		0	0	0	0	0	0	0		0	0		0	0	0	0
0x14	USART_CR3	Reserved																				ONEBIT	CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x18	USART_GTPR	Reserved																GT[7:0]								PSC[7:0]							
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 117 :USART register map and reset values

8.5. USART in Code Register Configurations

```
/* USER CODE END USART1_Init 1 */
huart1.Instance = USART1;
huart1.Init.BaudRate = 115200;
huart1.Init.WordLength = UART_WORDLENGTH_8B;
huart1.Init.StopBits = UART_STOPBITS_1;
huart1.Init.Parity = UART_PARITY_NONE;
huart1.Init.Mode = UART_MODE_TX_RX;
huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart1.Init.OverSampling = UART_OVERSAMPLING_16;
```

Figure 118 :USART Code Register Configurations

8.5.1. USART in Code Pins Configurations

```
/* GPIO Ports Clock Enable */
SET_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOAEN_Pos);

GPIO_InitStruct2.Pin = GPIO_PIN_9|GPIO_PIN_10;
GPIO_InitStruct2.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct2.Pull = GPIO_NOPULL;
GPIO_InitStruct2.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
GPIO_InitStruct2.Alternate = GPIO_AF7_USART1;
GPIO_Init(GPIOA, &GPIO_InitStruct2);

SET_BIT(RCC->APB2ENR, RCC_APB2ENR_USART1EN_Pos);
```

Figure 119 :USART Code Pins Configurations

8.6. UART Testing between Simulink and Tera term (terminal) using virtual port

8.6.1. UART Testing (virtual port configuration)

We want to make testing for sending and receiving data without any software to ensure the functionality of the UART in Simulink, so we perform testing in "virtual serial port driver" program between "Tera term" Program and Simulink with simple following configurations:

- Baud rate :9600
- Number of data bits: 8
- Stop bit :1

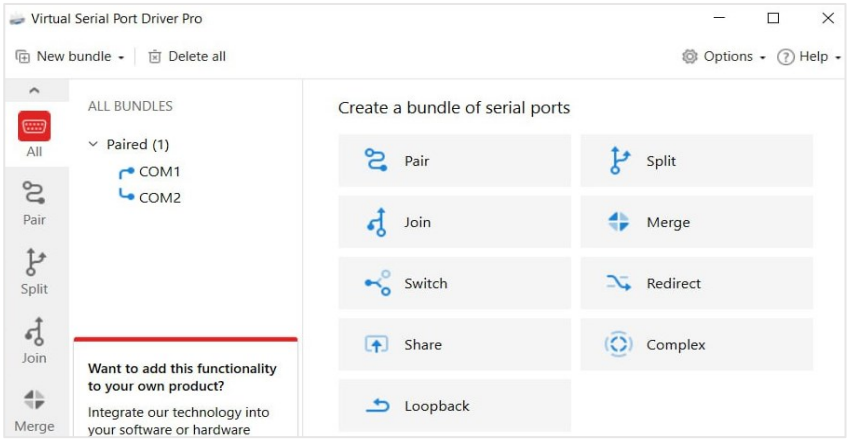


Figure 120 :USART virtual port configuration

8.6.2. UART Testing (Simulink Configuration)

First of all, we need to configure the Serial Port we are using for this communication. To do that, simply put the Serial Configuration block anywhere you want in your Simulink project.

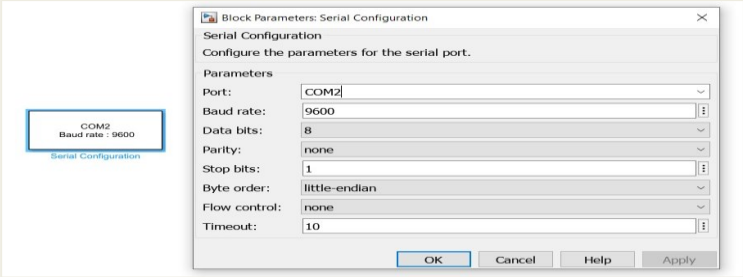


Figure 121 :USART Simulink Serial Configuration

Select COM2 for Simulink, and then select the baud rate and the standard configurations for UART.

8.6.2.1. UART Testing (Simulink setup - receive)

The Serial Receive block is used to receive serial data in our Simulink project. We need to configure this block to make the communication correctly.

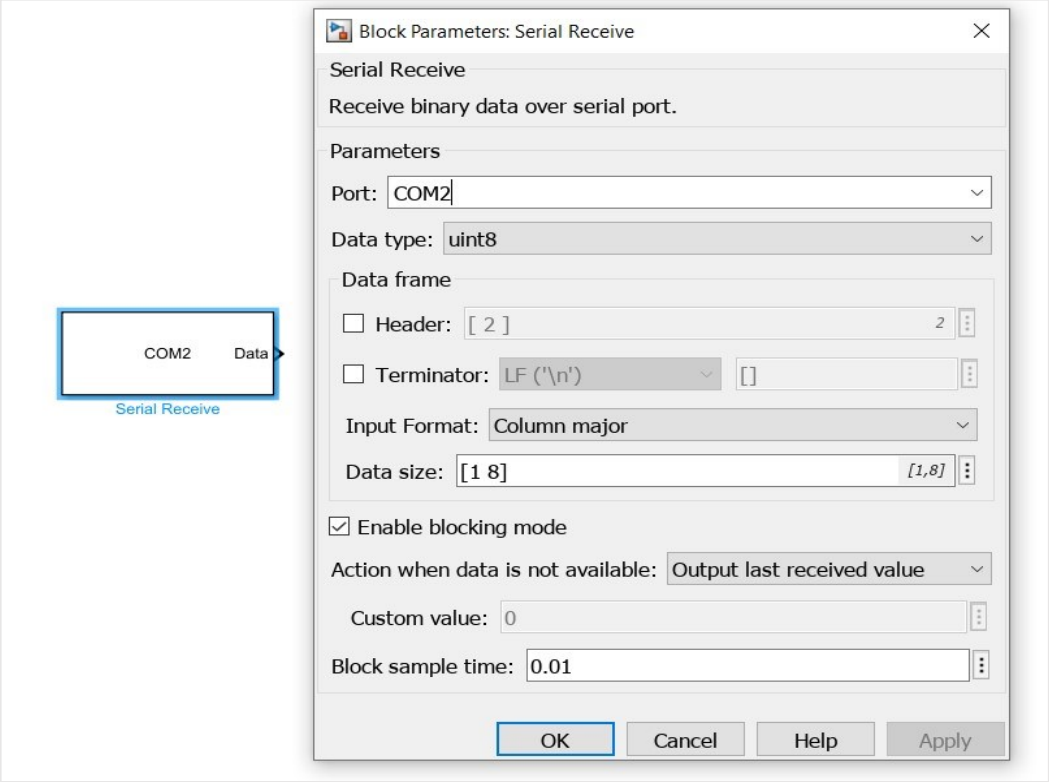


Figure 122 :USART Simulink Serial Receive Setup

- **Communication port:** Use the same one we configured in the step above.
- **Data size:** to send only one char, use [1 1] but we can change it to [1 N] where N is the number of characters we want to receive from Serial.

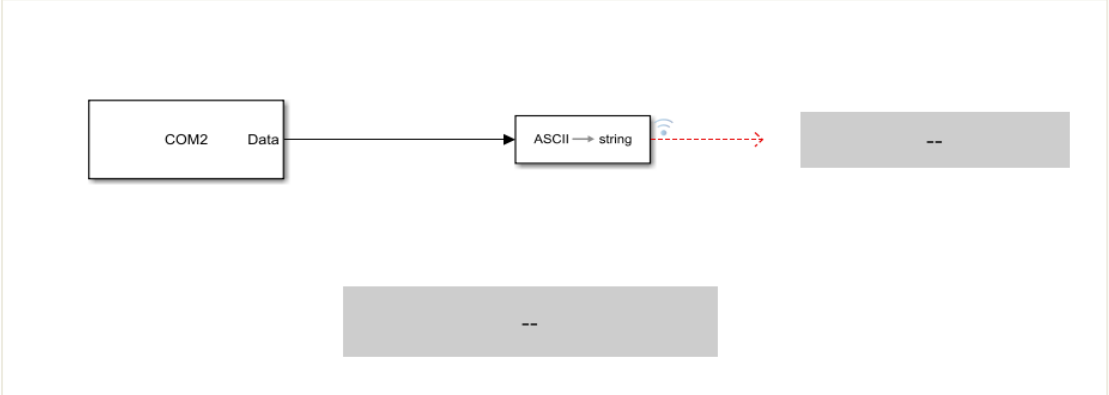


Figure 123 :USART Simulink Receiver block diagram

Now we

can receive the data. The only step left is to convert it to any variable you want, in our case we will display the **ASCII** and its equivalent characters.

8.6.2.2. UART Testing (Simulink setup - Send)

To send data from Simulink create a project similar to the image below

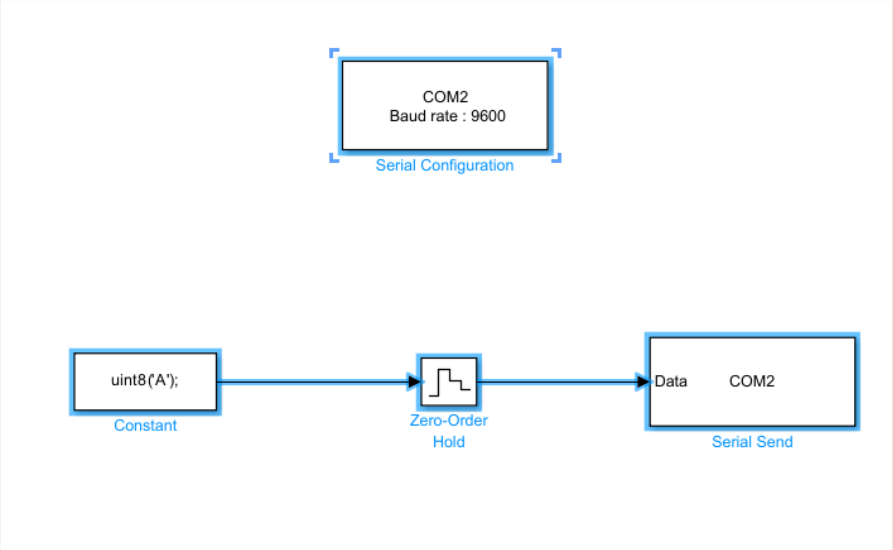


Figure 124 :USART Simulink Sender block diagram

- **Serial Send**: Send the bytes, you can add a Header and a Terminator if you want but I had no problem sending this data without them.
- **Zero-Order Hold**: Set the simulation send rate.

8.6.3. UART Testing (Tera term Configuration)

When we open Tera Term, New connection window will pop-up, we will select serial port “COM1”.

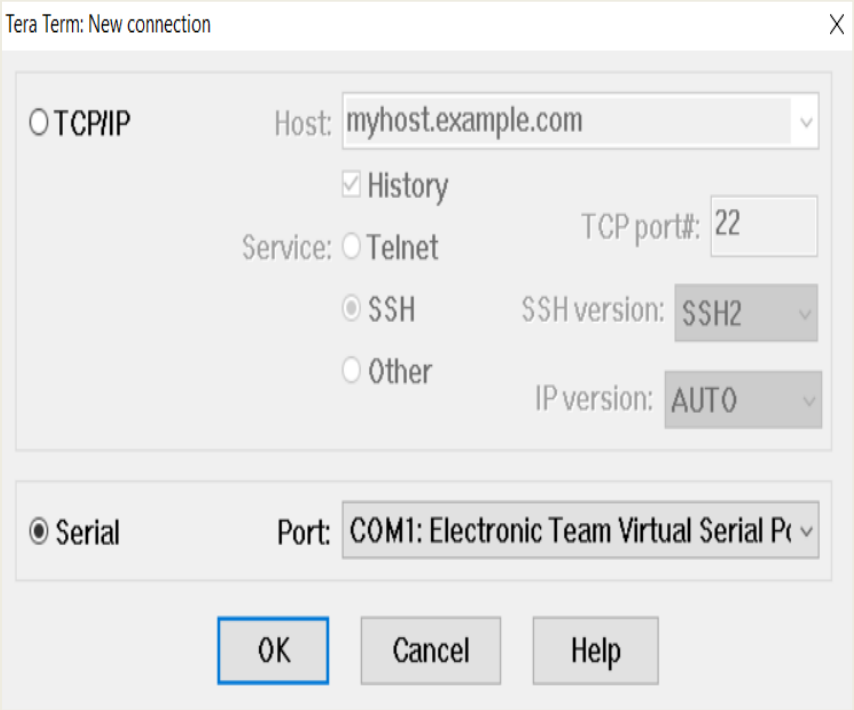


Figure 125 :USART Tera term Selecting virtual serial COM1

Select UART configurations that match with the serial block configurations in Simulink.

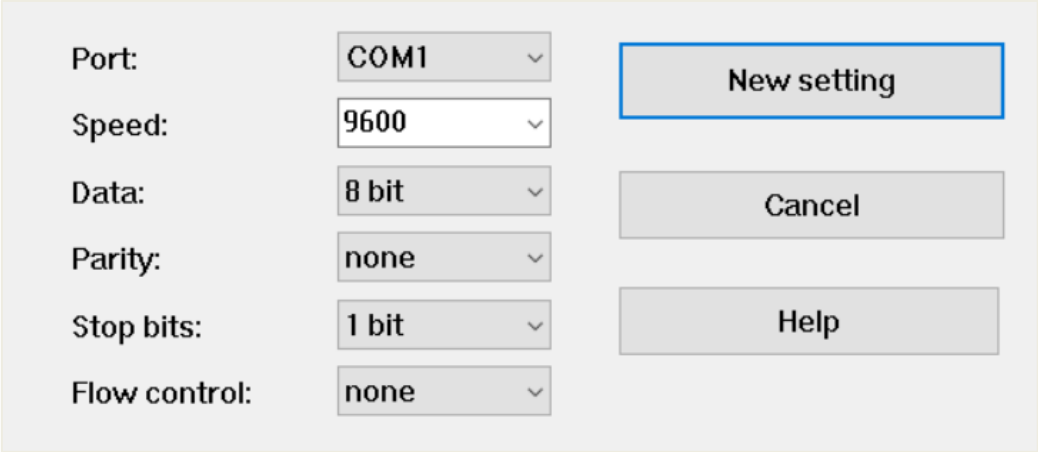


Figure 126 :USART Tera term setting configurations

8.6.4. UART Testing (Checking Successful connection)

Now we will run Tera Term on COM1 and Simulink on COM2, and then observe how data are transmitted from Simulink

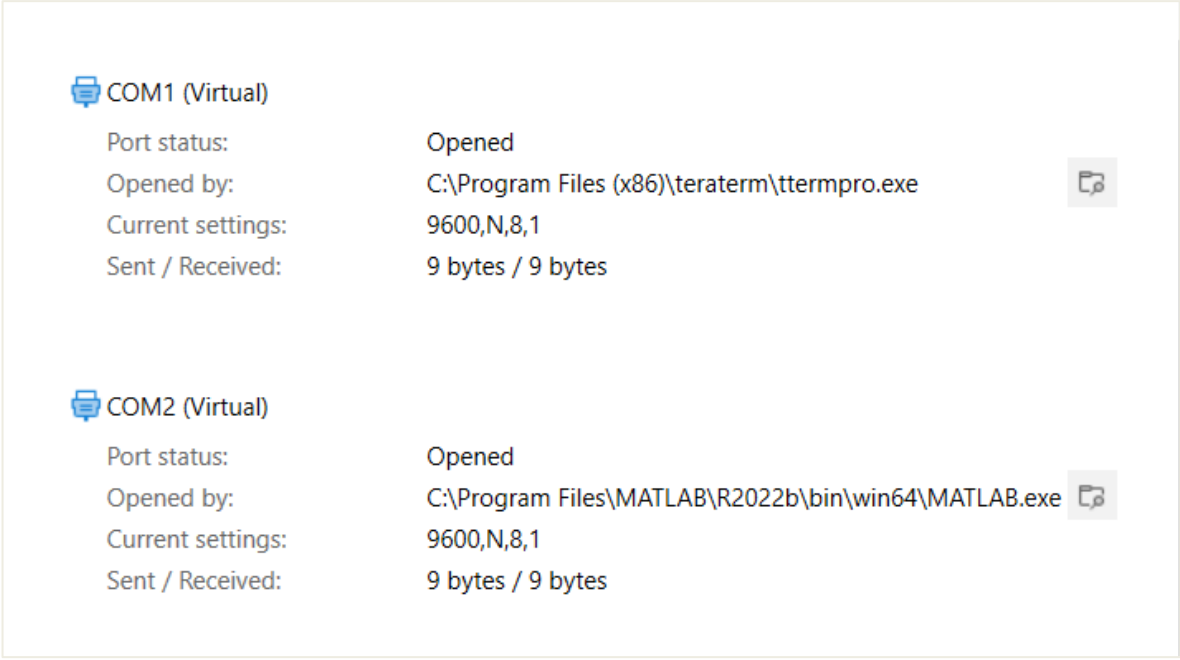


Figure 127 :USART Checking Successful connection in Virtual Serial port

As shown below, character 'A' is transmitted from serial send block in Simulink (COM2) to Tera Term (COM1)

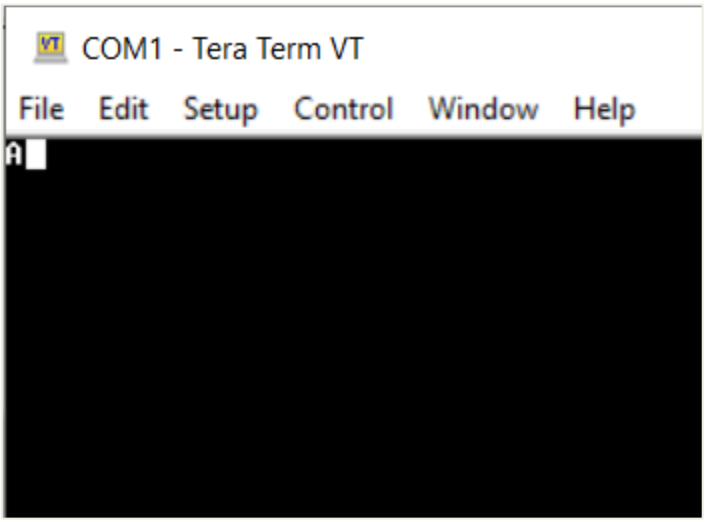


Figure 128 :USART Tera term terminal receiving char 'A'

8.6.5. UART Testing (Send and receive (Hardware in the loop)

Now we have both the send and receive blocks working together, we can join everything into a single project. The idea here is to create a simulation where the microcontroller (Tera Term for now) receive some data, process it, and sends back to Simulink. To do that, the Simulink block diagram is shown below.

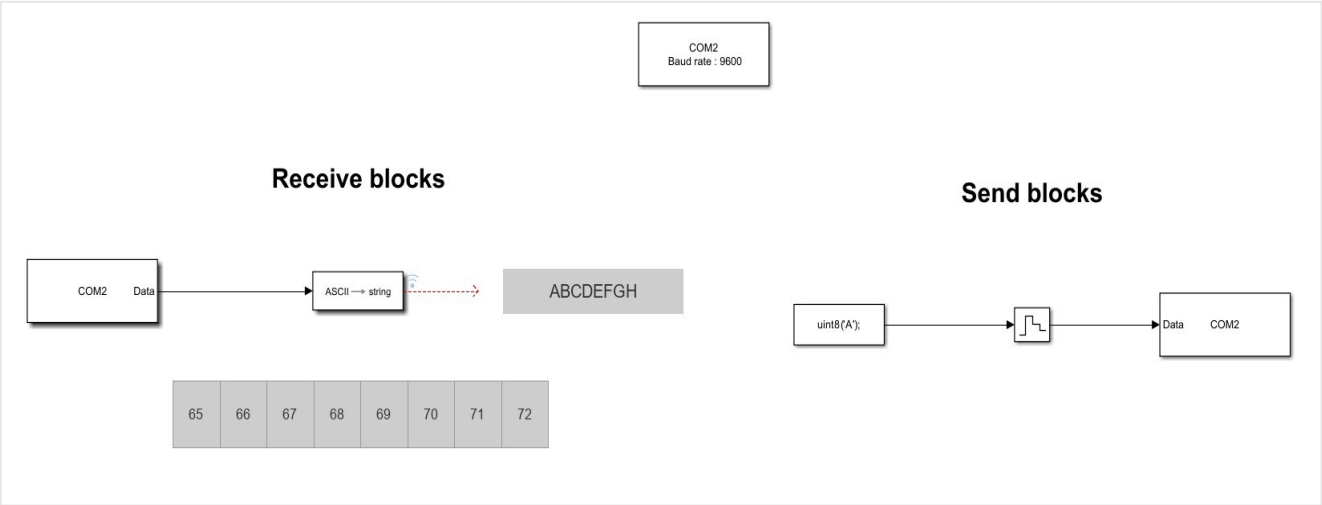


Figure 129 :USART block diagram of Sender & Receiver in Simulink

Chapter 9
Controller Area Network

(CAN)

9. CAN

CANopen is a trademark of CAN in Automation.

DeviceNet is a trademark of Open **DeviceNet** Vendor Association, Inc.

9.1. Introduction

The CAN bus was developed by **BOSCH** as a multi-master, message broadcast system that specifies a maximum signaling rate of 1 megabit per second (bps). Unlike a traditional network such as USB or Ethernet, CAN does not send large blocks of data point-to-point from node A to node B under the supervision of a central bus master. In a CAN network, many short messages like temperature or RPM are broadcast to the entire network, which provides for data consistency in every node of the system. Once CAN basics such as message format, message identifiers, and bit-wise arbitration a major benefit of the CAN signaling scheme are explained, a CAN bus implementation is examined, typical waveforms presented, and transceiver features examined.

9.2. The CAN Standard

CAN is an International Standardization Organization (ISO) defined serial communications bus originally developed for the automotive industry to replace the complex wiring harness with a two-wire bus. The specification calls for high immunity to electrical interference and the ability to self-diagnose and repair data errors. These features have led to CAN's popularity in a variety of industries including building automation, medical, and manufacturing.

The CAN communications protocol, ISO-11898: 2003, describes how information is passed between devices on a network and conforms to the Open Systems Interconnection (OSI) model that is defined in terms of layers. Actual communication between devices connected by the physical medium is defined by the physical layer of the model. The ISO 11898 architecture defines the lowest two layers of the seven layers OSI/ISO model as the data-link layer and physical layer in Figure.123.

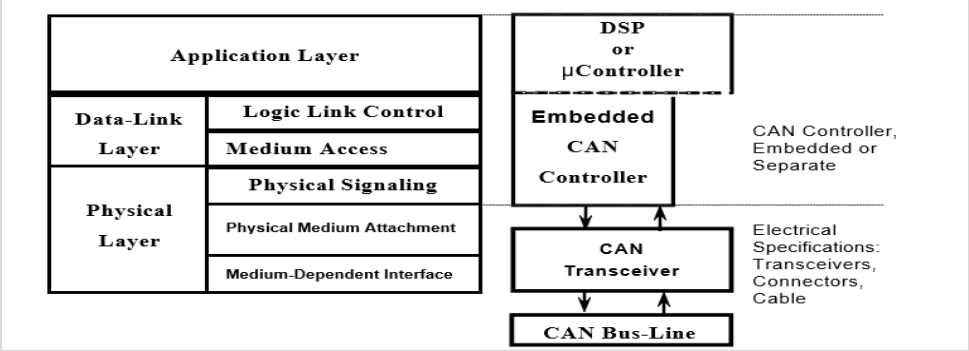


Figure 130 :The Layered ISO 11898 Standard Architecture

9.3. Standard CAN or Extended CAN

The CAN communication protocol is a carrier-sense, multiple-access protocol with collision detection and arbitration on message priority (CSMA/CD+AMP) (Carrier Sense Multiple Access with Collision Avoidance+ Arbitration on Message Priority). CSMA means that each node on a bus must wait for a

prescribed period of inactivity before attempting to send a message. CD+AMP means that collisions are resolved through a bit-wise arbitration, based on a preprogrammed priority of each message in the identifier field of a message. The higher priority identifier always wins bus access. That is, the last logic- high in the identifier keeps on transmitting because it is the highest priority. Since every node on a bus takes part in writing every bit "as it is being written," an arbitrating node knows if it placed the logic-high bit on the bus.

The ISO-11898:2003 Standard, with the standard 11-bit identifier, provides for signaling rates from 125 kbps to 1 Mbps. The standard was later amended with the “extended” 29-bit identifier. The standard 11-bit identifier field in Figure 2 provides for 2^{11}, or 2048 different message identifiers, whereas the extended 29- bit identifier in Figure 3 provides for 2^{29}, or 537 million identifiers.

9.4. The Bit Fields of Standard CAN and Extended CAN

9.4.1. Standard CAN

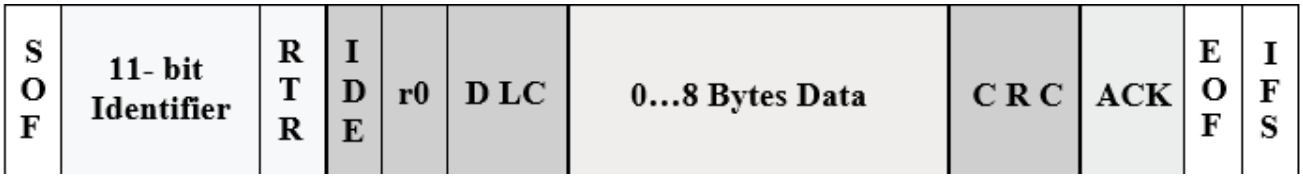


Figure 131 :Standard CAN: 11-Bit Identifier

The description of the bit fields of Figure.124 are:

- SOF–The single dominant start of frame (SOF) bit marks the start of a message, and is used to synchronize the nodes on a bus after being idle.
- Identifier-The Standard CAN 11-bit identifier establishes the priority of the message. The lower the binary value, the higher its priority.
- RTR–The single remote transmission request (RTR) bit is dominant when information is required from another node. All nodes receive the request, but the identifier determines the specified node. The responding data is also received by all nodes and used by any node interested. In this way, all data being used in a system is uniform.
- IDE–A dominant single identifier extension (IDE) bit means that a standard CAN identifier with no extension is being transmitted.
- r0–Reserved bit (for possible use by future standard amendment).
- DLC–The 4-bit data length code (DLC) contains the number of bytes of data being transmitted.
- Data–Up to 64 bits of application data may be transmitted.
- CRC–The 16-bit (15 bits plus delimiter) cyclic redundancy check (CRC) contains the checksum (number of bits transmitted) of the preceding application data for error detection.

- ACK–Every node receiving an accurate message overwrites this recessive bit in the original message with a dominate bit, indicating an error-free message has been sent. Should a receiving node detect an error and leave this bit recessive, it discards the message and the sending node repeats the message after rearbitration. In this way, each node acknowledges (ACK) the integrity of its data. ACK is 2 bits; one is the acknowledgment bit and the second is a delimiter.
- EOF–This end-of-frame (EOF), 7-bit field marks the end of a CAN frame (message) and disables bit- stuffing, indicating a stuffing error when dominant. When 5 bits of the same logic level occur in succession during normal operation, a bit of the opposite logic level is *stuffed* into the data.
- IFS–This 7-bit interframe space (IFS) contains the time required by the controller to move a correctly received frame to its proper position in a message buffer area.

9.4.2. Extended CAN

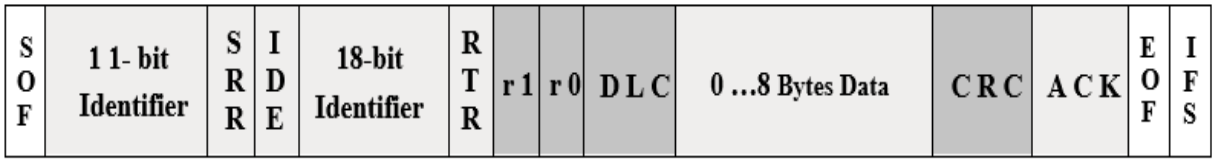


Figure 132 :Extended CAN: 29-Bit Identifier

As shown in Figure 125, the Extended CAN message is the same as the Standard message with the addition of:

- SRR–The substitute remote request (SRR) bit replaces the RTR bit in the standard message location as a placeholder in the extended format.
- IDE–A recessive bit in the identifier extension (IDE) indicates that more identifier bits follow. The 18-bit extension follows IDE.
- r1–Following the RTR and r0 bits, an additional reserve bit has been included ahead of the DLC bit.

9.5. A CAN Message

9.5.1. Arbitration

A fundamental CAN characteristic shown in Figure.126 is the opposite logic state between the bus, and the driver input and receiver output. Normally, a logic-high is associated with a one, and a logic-low is associated with a zero - but not so on a CAN bus. This is why TI CAN transceivers have the driver input and receiver output pins passively pulled high internally, so that in the absence of any

input, the device automatically defaults to a recessive bus state on all input and output pins.

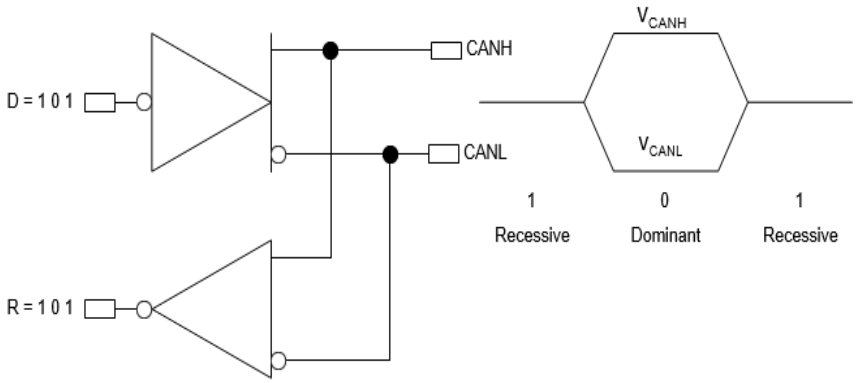


Figure 133 :The Inverted Logic of a CAN Bus

The allocation of message priority is up to a system designer, but industry groups mutually agree on the significance of certain messages. For example, a manufacturer of motor drives may specify that message 0010 is a winding current feedback signal from a motor on a CAN network and that 0011 is the tachometer speed. Because 0010 has the lowest binary identifier, messages relating to current values always have a higher priority on the bus than those concerned with tachometer readings.

In the case of **DeviceNet™,** devices from many manufacturers such as proximity switches and temperature sensors can be incorporated into the same system. Because the messages generated by **DeviceNet** sensors have been predefined by their professional association, the Open **DeviceNet** Vendors Association (ODVA), a certain message always relates to the specific type of sensor such as temperature, regardless of the actual manufacturer.

9.5.2. Message Types

The four different message types or frames, (see Figure.124 and.125) that can be transmitted on a CAN bus are the data frame, the remote frame, the error frame, and the overload frame.

9.5.3. The Data Frame

The data frame is the most common message type, and comprises the Arbitration Field, the Data Field, the CRC Field, and the Acknowledgment Field. The Arbitration Field contains an 11-bit identifier in Figure.124 and the RTR bit, which is dominant for data frames. In Figure.125, it contains the 29-bit identifier and the RTR bit. Next is the Data Field which contains zero to eight bytes of data, and the CRC Field which contains the 16-bit checksum used for error detection. Last is the Acknowledgment Field.

9.5.4. The Remote Frame

The intended purpose of the remote frame is to solicit the transmission of data from another node. The

remote frame is similar to the data frame, with two important differences. First, this type of message is explicitly marked as a remote frame by a recessive RTR bit in the arbitration field, and secondly, there is no data.

9.5.5. The Error Frame

The error frame is a special message that violates the formatting rules of a CAN message. It is transmitted when a node detects an error in a message, and causes all other nodes in the network to send an error frame as well. The original transmitter then automatically retransmits the message. An elaborate system of error counters in the CAN controller ensures that a node cannot tie up a bus by repeatedly transmitting error frames.

9.5.6. The Overload Frame

The overload frame is mentioned for completeness. It is similar to the error frame with regard to the format, and it is transmitted by a node that becomes too busy. It is primarily used to provide for an extra delay between messages.

9.5.7. A Valid Frame

A message is considered to be error free when the last bit of the ending EOF field of a message is received in the error-free recessive state. A dominant bit in the EOF field causes the transmitter to repeat a transmission.

9.5.8. Error Checking and Fault Confinement

The robustness of CAN may be attributed in part to its abundant error-checking procedures. The CAN protocol incorporates five methods of error checking: three at the message level and two at the bit level. If a message fails any one of these error detection methods, it is not accepted and an error frame is generated from the receiving node. This forces the transmitting node to resend the message until it is received correctly. However, if a faulty node hangs up a bus by continuously repeating an error, it's transmission capability is removed by its controller after an error limit is reached.

Error checking at the message level is enforced by the CRC and the ACK slots displayed in Figure 2 and Figure.125. The 16-bit CRC contains the checksum of the preceding application data for error detection with a 15-bit checksum and 1-bit delimiter. The ACK field is two bits long and consists of the acknowledge bit and an acknowledge delimiter bit.

Also at the message level is a form check. This check looks for fields in the message which must always be recessive bits. If a dominant bit is detected, an error is generated. The bits checked are the SOF, EOF, ACK delimiter, and the CRC delimiter bits

At the bit level, each bit transmitted is monitored by the transmitter of the message. If a data bit (not arbitration bit) is written onto the bus and its opposite is read, an error is generated. The only exceptions to this are with the message identifier field which is used for arbitration, and the acknowledge slot which requires a recessive bit to be overwritten by a dominant bit.

The final method of error detection is with the bit-stuffing rule where after five consecutive bits of the same logic level, if the next bit is not a complement, an error is generated. Stuffing ensures that rising edges are available for on-going synchronization of the network. Stuffing also ensures that a stream of bits is not been mistaken for an error frame, or the seven-bit interframe space that signifies the end of a message. Stuffed bits are removed by a receiving node's controller before the data is forwarded to the application.

With this logic, an active error frame consists of six dominant bits violating the bit stuffing rule. This is interpreted as an error by all of the CAN nodes which then generate their own error frame. This means that an error frame can be from the original six bits to twelve bits long with all the replies. This error frame is then followed by a delimiter field of eight recessive bits and a bus idle period before the corrupted message is retransmitted. It is important to note that the retransmitted message still has to contend for arbitration on the bus.

9.6. The CAN Bus

The data link and physical signaling layers of Figure.123, which are normally transparent to a system operator, are included in any controller that implements the CAN protocol, such as TI's **TMS320LF2812** 3.3-V DSP with integrated CAN controller. Connection to the physical medium is then implemented through a line transceiver such as Texas Instruments (TI's) **SN65HVD230** 3.3-V CAN transceiver to form a system node as shown in Figure.127.

Signaling is differential which is where CAN derives its robust noise immunity and fault tolerance. Balanced differential signaling reduces noise coupling and allows for high signaling rates over twisted-pair cable. Balanced means that the current flowing in each signal line is equal but opposite in direction, resulting in a field-canceling effect that is a key to low noise emissions. The use of balanced differential receivers and twisted-pair cabling enhance the common-mode rejection and high noise immunity of a CAN bus.

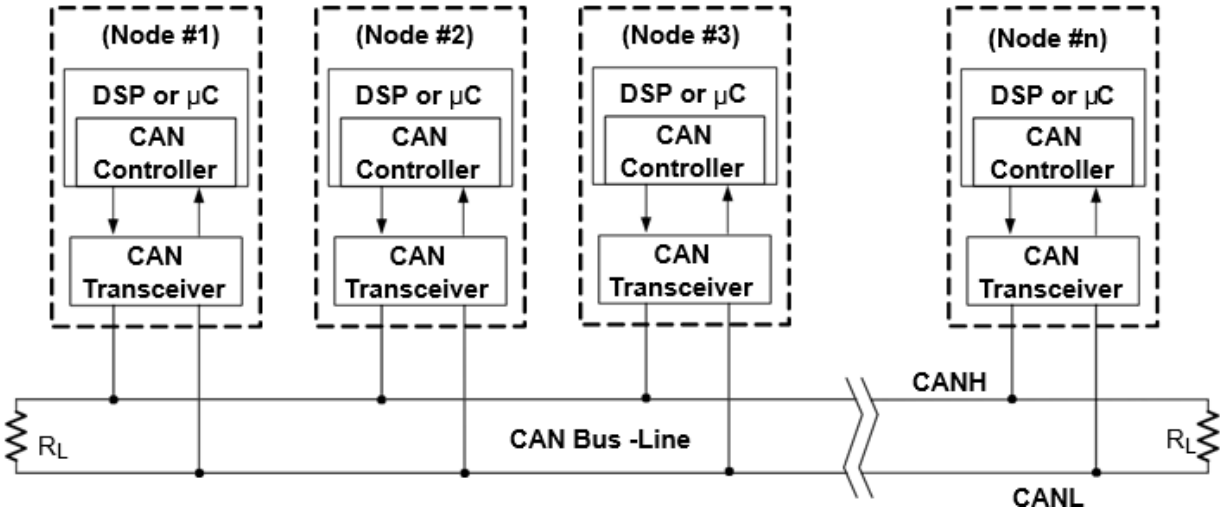


Figure 134 :Details of a CAN Bus

The High-Speed ISO 11898 Standard specifications are given for a maximum signaling rate of 1 Mbps with a bus length of 40 m with a maximum of 30 nodes. It also recommends a maximum unterminated

stub length of 0.3 m. The cable is specified to be a shielded or unshielded twisted-pair with a 120-Ω characteristic impedance (Zo). The ISO 11898 Standard defines a single line of twisted-pair cable as the network topology as shown in Figure.128, terminated at both ends with 120-Ω resistors, which match the characteristic impedance of the line to prevent signal reflections. According to ISO 11898, placing R_L on a node must be avoided because the bus lines lose termination if the node is disconnected from the bus.

The two signal lines of the bus, CANH and CANL, in the quiescent recessive state, are passively biased to ≈ 2.5 V. The dominant state on the bus takes CANH ≈ 1 V higher to ≈ 3.5 V, and takes CANL ≈ 1 V lower to ≈ 1.5 V, creating a typical 2-V differential signal as displayed in Figure.128.

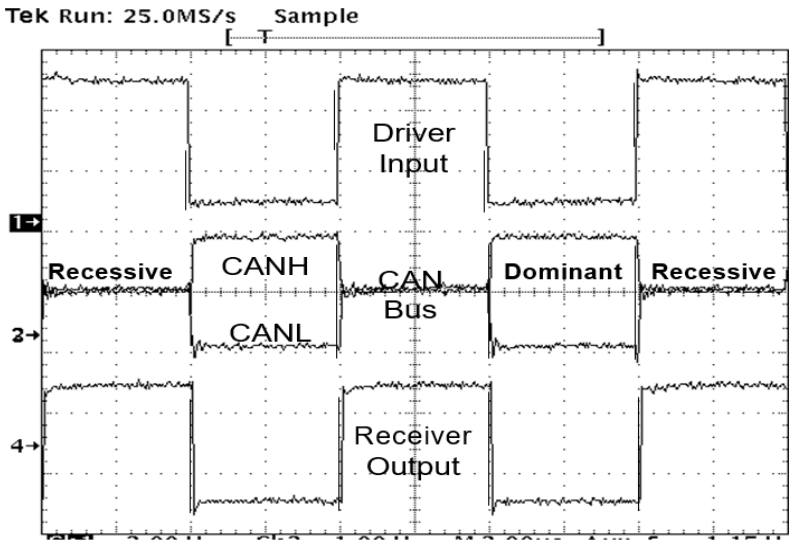


Figure 135 :CAN Dominant and Recessive Bus States

The CAN standard defines a communication network that links all the nodes connected to a bus and enables them to talk with one another. There may or may not be a central control node, and nodes may be added at any time, even while the network is operating (hot-plugging).The nodes in Figure 8 and Figure 9 could theoretically be sending messages from smart sensing technology and a motor controller. An actual application may include a temperature sensor sending out a temperature update that is used to adjust the motor speed of a fan. If a pressure sensor node wants to send a message at the same time, arbitration ensures that the message is sent.

For example, Node A in Figure.129 finishes sending its message (on the left side of Figure 129) as nodes B and C acknowledge a correct message being received. Nodes B and C then begin arbitration, node C wins the arbitration and sends its message. Nodes A and B acknowledge C's message, node B then continues on with its message.

Again note the opposite polarity of the driver input and output on the bus.

9.7. CAN Transceiver Features

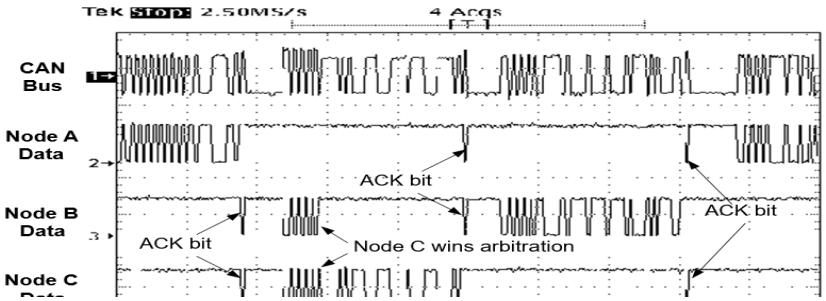


Figure 136 :CAN Bus Traffic

9.7.1. 3.3-V Supply Voltage

Most CAN transceivers require a 5-V power supply to reach the signal levels required by the ISO 11898 standard. However, by superior attention to high-efficiency circuit design, the TI 3.3-V CAN transceiver family operates with a 3.3-V power supply and is fully interoperable with 5-V CAN transceivers on the same bus. This allows designers to reduce total node power by 50% or more (Figure .130).

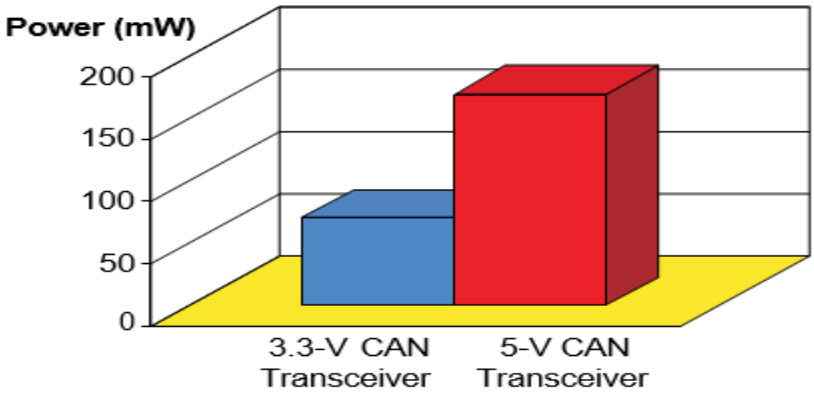


Figure 137 :3.3-V CAN Transceiver Power Savings

In addition to the inherent power savings of using a 3.3-V transceiver, for applications using 3.3-V technology, such as the TI TMS320C28xx family of DSPs with integrated CAN controllers, the need for a 5-V power supply can be eliminated. This lowers the overall part count for the node, reducing system cost and increasing system reliability.

For designers with an existing system design which requires a 5-V-powered transceiver, the TI 5-V transceivers are available with a wide variety of features such as high ESD protection and wide common- mode range.

9.7.2. Electrostatic discharge (ESD) Protection

Static charge is an unbalanced electrical charge at rest, typically created by the physical contact of different materials. One surface gains electrons, where the other surface loses electrons. This results in an unbalanced electrical condition known as a static charge. When a static charge moves from one surface to another, it is referred to as an electrostatic discharge (ESD). It can occur only when the voltage differential between the two surfaces is sufficiently high to break down the dielectric strength of the medium separating the two surfaces. ESD can occur in any one of four ways: a charged body can touch an integrated circuit (IC), a charged IC can touch a grounded surface, a charged machine can touch an IC, or an electrostatic field can induce a voltage across a dielectric sufficient to break it down. The main threat of ESD damage occurs during the assembly and manufacturing of circuits. After assembly and installation, the main protection required for the bus pins is surge protection.

9.7.3. Common-Mode Voltage Operating Range

Common-mode voltage is the difference in potential between grounds of sending and receiving nodes on a bus. This is often the case in the networked equipment typically found in a CAN application. Possible effects of this problem are intermittent reboots, lock-ups, bad data transfer, or physical damage to a transceiver. Network interface cards, parallel ports, serial ports, and especially transceivers are prime targets for some form of failure if not designed to accommodate high levels of ground shift and power supply imbalance between typical CAN nodes. With this in mind, most TI CAN transceivers are designed to operate with complete safety well beyond the bus voltage range of –2 V to 7 V required by the ISO11898 Standard.

9.7.4. Common-Mode Noise Rejection

Common-mode noise of varied magnitudes exist within the networks associated with CAN applications. Noise from pulsing motor controllers, switch-mode power supplies, or from fluorescent lighting load are the typical sources of noises that couple onto bus lines. transceivers are specifically designed and tested for their ability to reject this common-mode noise.

9.7.5. Controlled Driver Output Transition Times

Controlling the driver output slew rate dampens the rise time of a dominant bit to improve signal quality and provides for longer stub lengths and a better bit-error rate. For a discussion on how slew-rate control provides for longer stub lengths, see application report SLLA270.

9.7.6. Low-Current Bus Monitor, Standby and Sleep Modes

Many applications are looking to lower-power opportunities as more electronics are added to designs. The standby mode in many TI transceivers is generally referred to as the “listen only” mode, because in standby, the driver circuitry is switched off while the receiver continues to monitor bus activity. In the occurrence of a dominant bit on the bus, the receiver passes this information along to its DSP/CAN controller which in turn activates the circuits that are in standby. This is achieved by placing a logic-low level on the Rs pin (pin 8) of the device. The difference between the standby mode and the sleep mode is that both driver and receiver circuits can be switched off to create an extremely low-power sleep mode with no bus monitor. The local controller actively places the device into and out of sleep

mode. The HVD1040 contains the best of both standby and sleep features with a low-power (5 µA typical) bus monitor. The device driver and receiver circuitry is switched off while a small comparator monitors the bus and toggles the receiver output on bus activity.

9.7.7. Bus Pin Short-Circuit Protection

The ISO11898 Standard recommends that a transceiver survive bus wire short-circuits to each other, to the power supply, and to ground. This ensures that transceivers are not damaged by bus cable polarity reversals, cable crush, and accidental shorts to high power supplies. The short-circuit protection in TI devices protects for an unlimited time. Once a problem is removed, the devices perform as designed whereas the CAN transceivers offered from competing vendors are permanently damaged and require replacement.

9.7.8. Thermal Shutdown Protection

Another desirable safety feature for a CAN transceiver is the thermal shutdown circuitry of TI CAN transceivers. This feature protects a device against the destructive currents and resulting heat that can occur in a short-circuit condition. Once thermal shutdown is activated, the device remains shut down until the circuitry is allowed to cool. Once cooled down to normal operating temperature, the device automatically returns to active operation without damage.

9.7.9. Bus Input Impedance

A high bus input impedance increases the number of nodes that can be added to a bus above the ISO 11898 Standard's 30 nodes. The high impedance restricts the amount of current that a receiver sinks or sources onto a bus over common-mode voltage conditions. This ensures that a driver transmitting a message into such a condition is not required to sink or source a damaging amount of current from the sum of the receiver leakage currents on a bus.

9.7.10. Glitch-Free Power Up and Power Down

This feature provides for hot-plugging onto a powered bus without disturbing the network. The TI driver and receiver pins are passively pulled high internally while the bus pins are biased internally to a high- impedance recessive state. This provides for a power up into a known recessive condition without disturbing ongoing bus communication.

9.7.11. Unpowered Node Protection

Many CAN transceivers on the market today have a low output impedance when unpowered. This low impedance causes the device to sink any signal present on the bus and shuts down all data transmission. TI CAN transceivers have a high output impedance in powered and unpowered conditions and maintain the integrity of the bus any time power or ground is removed from the circuit.

9.7.12. Reference Voltage

Reference voltage on a CAN transceiver is the Vref pin (pin 5) of what is considered to be the standard CAN transceiver footprint. This is the footprint of the first CAN transceiver to market, the

NXP PCA82C250. When first introduced, the Vcc/2 Vref pin served a particular NXP CAN controller as a voltage reference used to compare the bus voltage of a remaining single bus line in the event of an accident. If the voltages were the same, it was a recessive bit; if different, it was a dominant bit. Although some users consider it handy for use as an actual voltage reference at the node, it is typically unused.

9.7.13. Loopback

This function places the bus input and output in a high-impedance state. The remaining transceiver circuitry remains active and available for driver-to-receiver loopback and self-diagnostic node functions without disturbing the bus.

9.7.14. Autobaud Loopback

In autobaud loopback, the “bus-transmit” function of the transceiver is disabled, while the “bus-receive” function and all of the normal operating functions of the device remain intact. With the autobaud function engaged, normal bus activity can be monitored by the device. Autobaud detection is best suited to applications that have a known selection of baud rates. For example, a popular industrial application has optional settings of 125 kbps, 250 kbps, or 500 kbps. Once a logic-high has been applied to pin 5 (AB) of the HVD235, assume a baud rate such as 125 kbps; then wait for a message to be transmitted by another node on the bus. If the wrong baud rate has been selected, an error message is generated by the host CAN controller. However, because the “bus-transmit” function of the device has been disabled, no other nodes receive the error message of the controller. This procedure makes use of the CAN controller’s status register indications of message received and error warning status to signal if the current baud rate is correct or not. The warning status indicates that the CAN chip error counters have been incremented. A message-received status indicates that a good message has been received. If an error is generated, reset the CAN controller with another baud rate and wait to receive another message. When an error-free message has been received, the correct baud rate has been detected.

9.8. CAN in our Project

firstly, we use CAN in our project to receive data from Raspberry Pi in which we run the algorithm of face detection for driver to set the status of this driver, we make Data come from raspberry using ISR which resume the task suspended firstly before the program begin after taking action it suspends the task again.

9.8.1. Connecting Raspberry Pi to a CAN Bus

Firstly, know that Raspberry Pi boards aren't ready out-of-the-box for CAN, so we must know some points:

- CAN Bus not supported by Raspberry Pi hardware (GPIO).
- CAN Bus wasn't supported by Raspberry Pi software (Raspbian).
- We will use a bridge between Raspberry Pi and CAN Bus (SPI Bus).

- We will use a CAN controller supported by Linux/Raspbian (MCP2515).

9.9. Hardware

9.9.1. CAN Bus Basics

CAN Bus uses two-wires differential signals. One is called CAN-H (High) and the other CAN-L (Low).

9.9.2. CAN Node

CAN Bus needs at least two nodes to make the network "up", Therefore testing with one node will not work.
obstacle n.1: Never use only one CAN Bus node.

9.9.3. CAN Termination

CAN Bus needs termination resistors between CAN-H and CAN-L at each end. Mainly to avoid signal reflections. ISO11898 recommends the value R=120Ω.

And that's obstacle n.2: Without termination your CAN Bus will not work correctly.

9.9.4. CAN Controller

CAN Bus is a multi-master protocol; each node needs a controller to manage its data. CAN controller is connected to the CAN bus using CAN-H and CAN-L.
Example of controllers: MCP2515, SJA100, ...

9.9.5. CAN Transceiver

CAN controller needs a send/receive chip to adapt signals to CAN Bus levels.
Controller and Transceiver are connected by two wires TxD and RxD.
Examples: MCP2551, TJA1040...

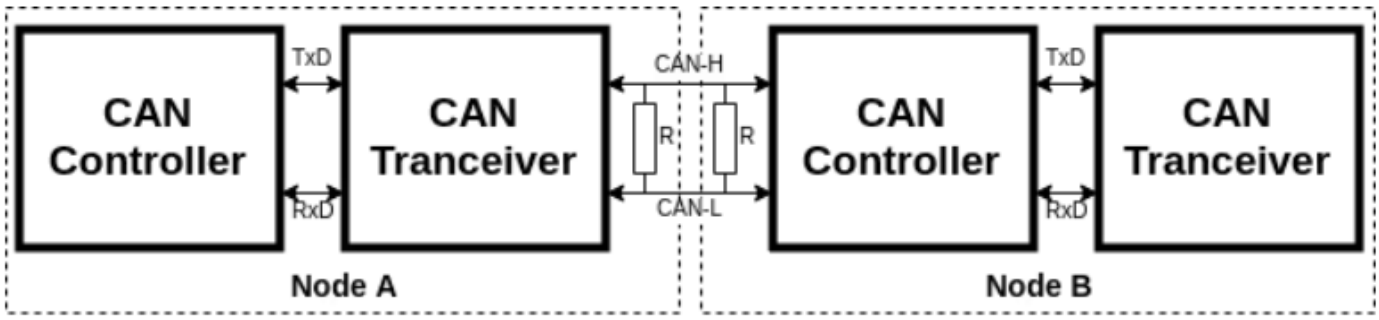


Figure 138 :2 Nodes CAN network

9.9.6. SPI Bus Basics

Raspberry Pi doesn't have a built-in CAN Bus (that's why we are doing all that...)

But its GPIO includes SPI Bus, that is supported by large number of CAN controllers.

9.9.7. SPI Wiring

SPI Bus uses 4 connections as follow:

MOSI (Master Out Slave In)

MISO (Master In Slave Out)

SCLK (Serial Clock)

Additional to that two other output pins:

- CS or SS (Chip Select, Slave Select) to enable and disable the chip.
- INT (Interruption) to interrupt the chip when needed.
- Including Raspberry Pi to the previous schematic gives:

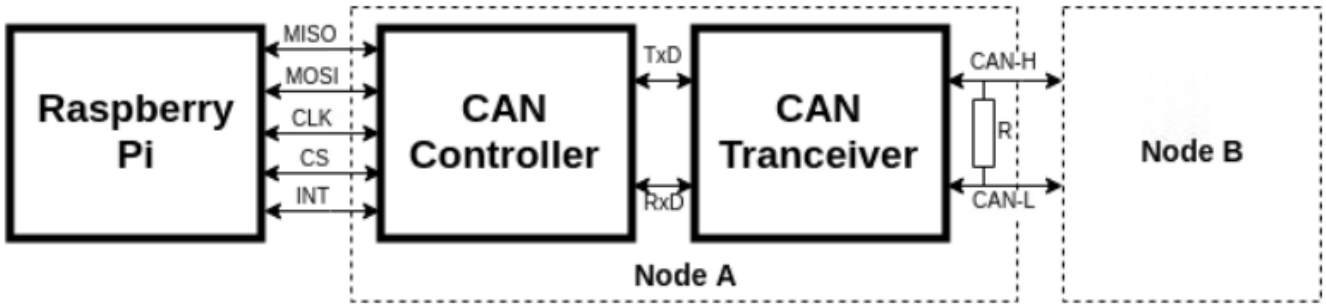


Figure 139 :connecting Raspberry pi with the CAN bus

9.9.8. MCP2515 (CAN Controller)

Supply: The MCP2515 can work at 5V or 3V3, so it can be connected to Raspberry Pi GPIO.

Clock: The MCP2515 needs an external quartz. For a value of 16Mhz, the datasheet recommends two capacitors of 22pF each.

Pull-up: Three pins need a pull-up resistor to stay at a recognized level, we took 4K7 (4,7KΩ) resistors, pins are:

Pin 17 for Reset.

Pin 16 for CS.

Pin 12 for INT.

The SPI bus itself (MOSI, MISO, SCLK) doesn't need pull-up resistors.

9.9.9. MCP2551 (CAN Transceiver)

Supply: Contrary to the MCP2515, the MCP2551 can only work at 5V.

obstacle n.3: Incorrect voltage levels between IC and Raspberry Pi.

If you connect MCP2515 to MCP2551 directly, the signals will not have the same level.

Reset: We will not need to reset the MCP2551 so the pin 8 can be connected to GND.

9.9.10. 5V to 3V3

The easiest way for any one is by a resistor divider (voltage divider) between MCP2515 and MCP2551.

We need this divider only for the RxD since here where the 5V will come in. like theses we may use:

R1: 10K (10KΩ)

R2 = 22K (22KΩ)

But we use 8 Bit Level Shifter TXB0108 Full Duplex IC to convert the 5v to 3.3v.

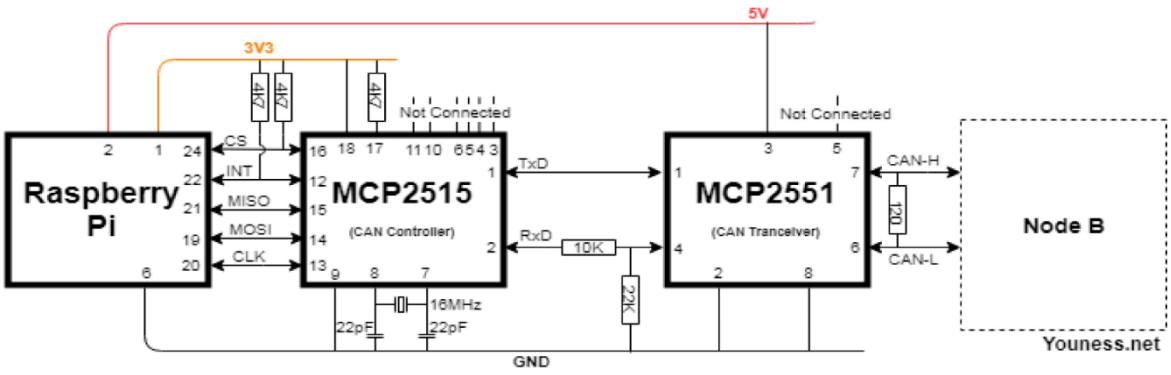


Figure 140: the overall schematic between Raspberry pi with the CAN bus

9.10. Software CAN configuration in Raspberry Pi:

In raspberry pi terminal follow these commands:

- sudo apt-get update
- sudo apt-get upgrade
- sudo reboot

SPI/CAN Configuration

1. Enable SPI and overlay it as follows:
   - sudo nano /boot/config.txt
2. Uncomment the line:

- dtparam=spi=on

3. After the above line add this:
   - dtoverlay=mcp2515-can0, oscillator=8000000, interrupt=25, dtoverlay=spi0-hw-cs.

The above lines to overlay SPI and set can0 interface to 8MHz, and interruption to GPIO25 pin.

This is the correct way for +4.4.x kernels, obstacle n.is to use a deprecated overlay method. (For earlier versions the overlay may be different)

4. Save the file: CTRL+X, then Y, then Enter.
5. Reboot, you can check that the SPI module was started:
   - dmesg | grep -i spi
6. By the same command you can check the CAN module is started by default:
   - dmesg | grep -i can
7. If for any reason this is not the case, you can add CAN module at system start
   - sudo nano /etc/modules
8. Add "can" in a new line, save the file and reboot.

9.10.1. CAN Utils

This is the tool to send, receive, and install data:

- sudo apt-get install can-utils

For each one, set up the can0 interface with the same speed:

- sudo ip link set can0 up type can bitrate 125000

The 125000, or 125 Kb/s is the minimum baud rate for CAN.

Set one of them to listen to any CAN message:

- candump any

Send something using the other one:

- cansend can0 111#FF

9.11. Software CAN configuration in Discovery stm32f429:

Firstly, The CubeMX Configuration is as shown below

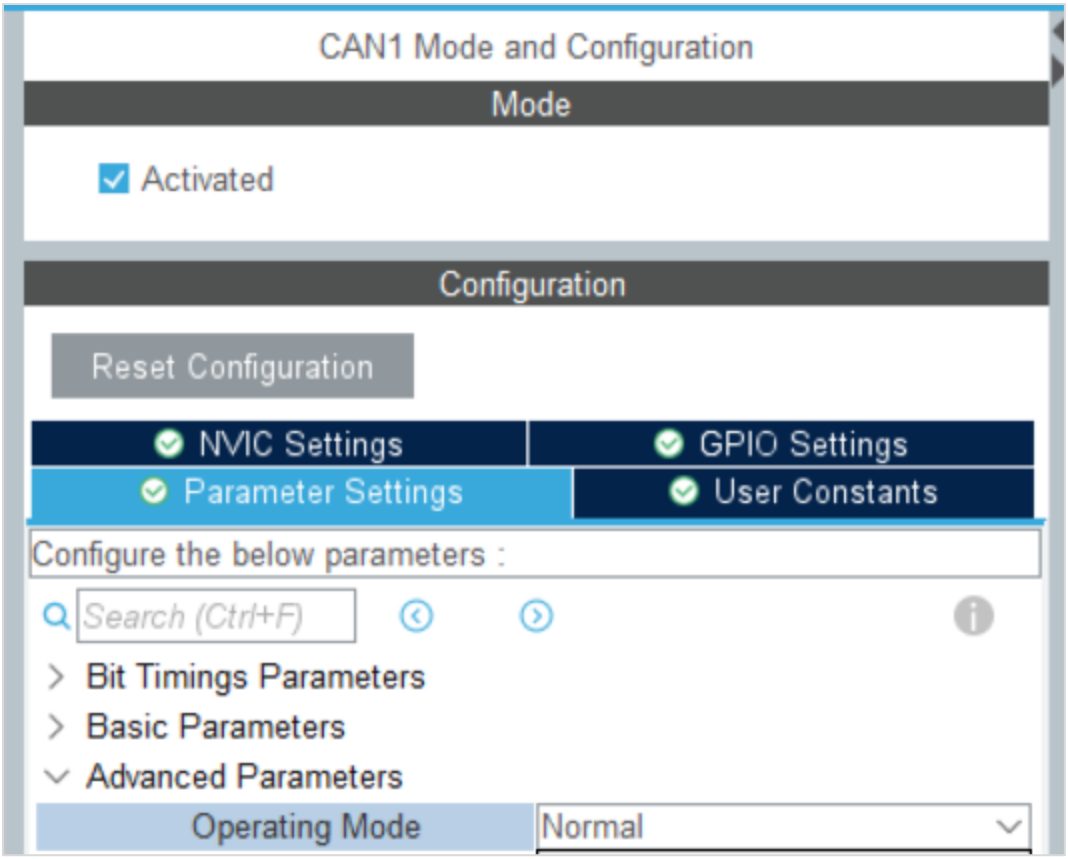


Figure 142 :CAN configuration in StmCubeMx Mode

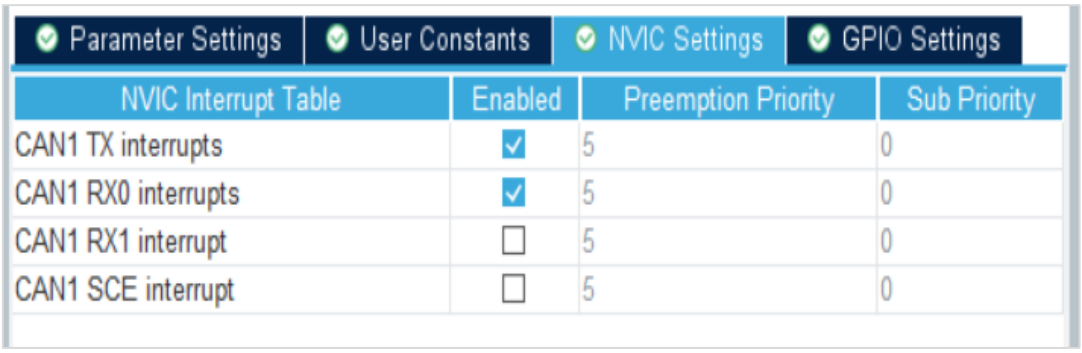


Figure 141 :CAN configuration in StmCubeMx interrupts

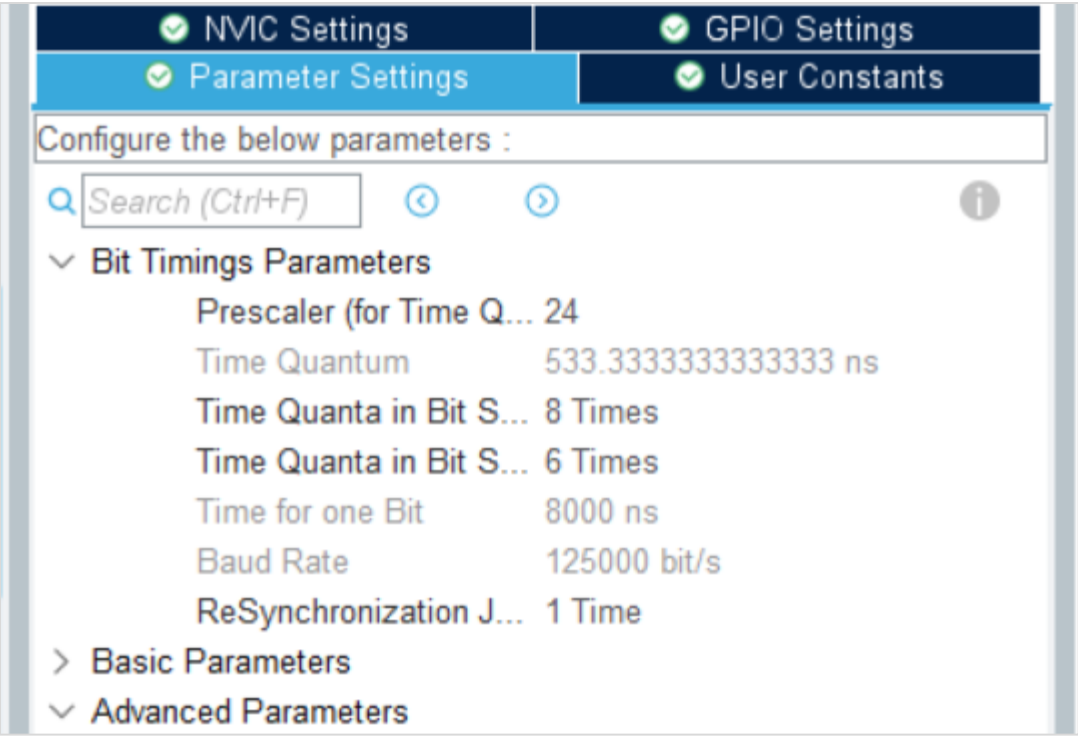


Figure 143 :CAN configuration in StmCubeMx Bit timing

- Prescaler: The prescaler is a value used to divide the system clock frequency to generate the time quantum. It determines the granularity of the time quantum and affects the overall bit rate of the CAN communication. A higher prescaler value leads to a lower time quantum and a higher bit rate, while a lower prescaler value results in a higher time quantum and a lower bit rate.

- Time Quanta in Bit Segment 1: The bit timing in the CAN bus is divided into multiple segments. Bit Segment 1 is the first segment and is responsible for the propagation time of the dominant bit. The Time Quanta in Bit Segment 1 refers to the number of time quanta allocated to this segment. It determines the length of Bit Segment 1 and influences the synchronization and sampling of the received data.

- Time Quanta in Bit Segment 2: Bit Segment 2 follows Bit Segment 1 and is responsible for the propagation time of the recessive bit. The Time Quanta in Bit Segment 2 refers to the number of time quanta allocated to this segment. It determines the length of Bit Segment 2 and affects the length of the inter-frame space (IFS) and the synchronization for the next bit.

The bit timing divided into 4 parts:

- Synchronization segment
- Propagation segment
- Phase segment1
- Phase segment2

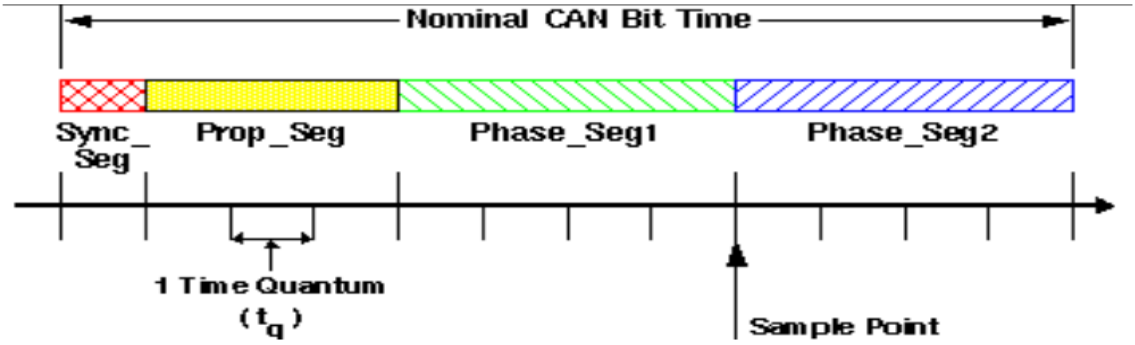


Figure 144 :CAN Bit timing

Total Bit Time = (Time Quanta in Bit Segment 1 + Time Quanta in Bit Segment 2+ Time Quanta in Sync_Seg+Prop_Seg) * Time Quantum

Parameter	Range	Remark
BRP	[1 .. 32]	defines the length of the time quantum t_q
Sync_Seg	1 t_q	fixed length, synchronization of bus input to system clock
Prop_Seg	[1 .. 8] t_q	compensates for the physical delay times
Phase_Seg1	[1 .. 8] t_q	may be lengthened temporarily by synchronization
Phase_Seg2	[1 .. 8] t_q	may be shortened temporarily by synchronization
SJW	[1 .. 4] t_q	may not be longer than either Phase Buffer Segment
This table describes the minimum programmable ranges required by the CAN protocol		

Table 3 :parameters of CAN Bit time

Baud Rate = System Clock / [(Prescaler) (Time Quanta in Bit Segment 1 + Time Quanta in Bit Segment 2+ Time Quanta in Sync_Seg+Prop_Seg)]

Note: We observed the CubeMX and ignored Propagation segment

Baud Rate = System Clock / [(Prescaler) (Time Quanta in Bit Segment 1 + Time Quanta in Bit Segment 2+ Time Quanta in Sync_Seg)]

For our Target baud rate 125 kbps, phase_seg1=6, phase_seg2=8, System_Clcok =45 Mbps

We found the prescaler =24.

If you want to test the software and the sender and receiver CAN without HW you can you use the mode of Loopback and Send any something and check you receive it or not.

Software CAN Code configuration

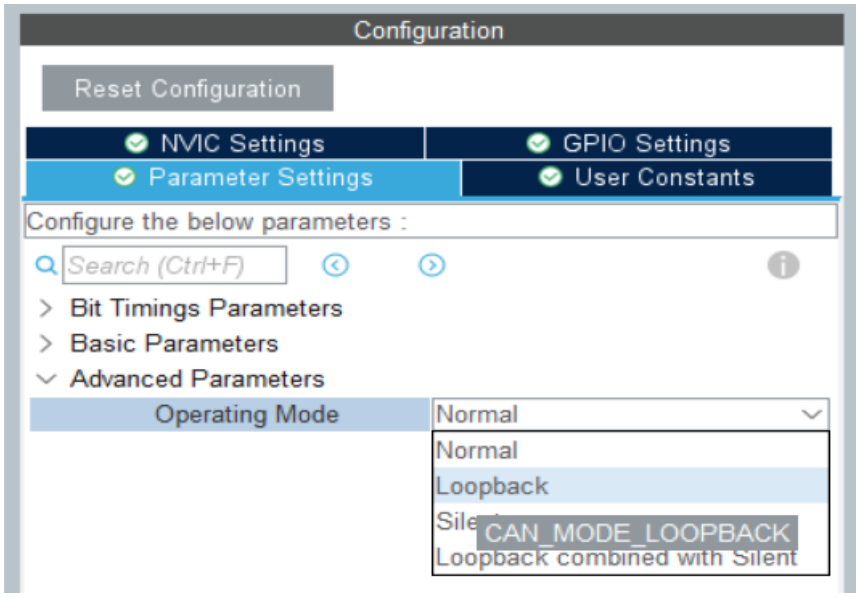


Figure 145 :CAN configuration in StmCubeMx Loopback

9.12. CAN global configuration code

```
CAN_HandleTypeDef hcan1;
CAN_FilterTypeDef sFilterConfig;
CAN_TxHeaderTypeDef TxHeader;
CAN_RxHeaderTypeDef RxHeader;
uint8_t TxData[8];
uint8_t RxData[8];
uint8_t RcvData;
uint32_t TxMailbox;
static void MX_CAN1_Init(void);
```

Figure 146 :CAN global configuration code

```
hcan1.Instance = CAN1;
hcan1.Init.Prescaler = 24;
hcan1.Init.Mode = CAN_MODE_NORMAL;
hcan1.Init.SyncJumpWidth = CAN_SJW_1TQ;
hcan1.Init.TimeSeg1 = CAN_BS1_8TQ;
hcan1.Init.TimeSeg2 = CAN_BS2_6TQ;
hcan1.Init.TimeTriggeredMode = DISABLE;
hcan1.Init.AutoBusOff = DISABLE;
hcan1.Init.AutoWakeUp = DISABLE;
hcan1.Init.AutoRetransmission = DISABLE;
hcan1.Init.ReceiveFifoLocked = DISABLE;
hcan1.Init.TransmitFifoPriority = DISABLE;
```

Figure 148 :CAN global configuration code

```
sFilterConfig.FilterBank = 0;
sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT;
sFilterConfig.FilterIdHigh = 0x0000;
sFilterConfig.FilterIdLow = 0x0000;
sFilterConfig.FilterMaskIdHigh = 0x0000;
sFilterConfig.FilterMaskIdLow = 0x0000;
sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO0;
sFilterConfig.FilterActivation = ENABLE;
sFilterConfig.SlaveStartFilterBank = 14;
```

Figure 149 :CAN configuration filters code

```
TxHeader.StdId = 0x321;
TxHeader.ExtId = 0x01;
TxHeader.RTR = CAN_RTR_DATA;
TxHeader.IDE = CAN_ID_STD;
TxHeader.DLC = 8;
TxHeader.TransmitGlobalTime = DISABLE;
TxData[0] = 1;
TxData[1] = 2;
TxData[2] = 3;
TxData[3] = 4;
TxData[4] = 5;
TxData[5] = 6;
TxData[6] = 7;
TxData[7] = 8;
```

Figure 147 :CAN Testing code for transmitting data

9.13. CAN Task

- CAN is configured using interrupt.
- We configure fifth task with higher priority and periodicity.
- The Task suspended initially, and the Callback of the ISR resume this suspension to run immediately and tack the action.
- The task receive flag 0, 1, 2 indicating:
  - Sending 1 represent 10 continuous frames in which the driver closes his eye.

    →It run a buzzer to warn the driver.

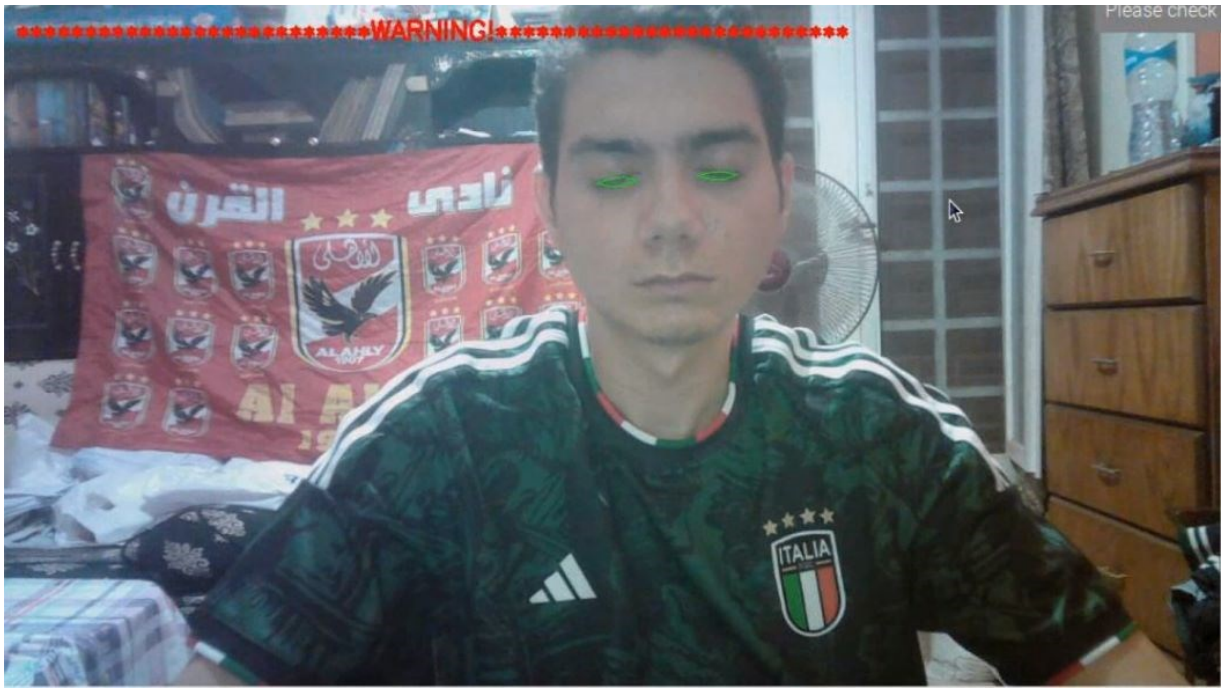


Figure 150 : drowsiness warning state

- After the driver open or recover from sleeping.

Figure 151 :Recovering from Warning

- Sending 2 represent 20 continuous frames in which the driver closes his eye.
- If the car located in the right or left lanes it stops immediately.
- If the car located in the middle lane, the algorithm turn the car to any side lane and the task suspend.

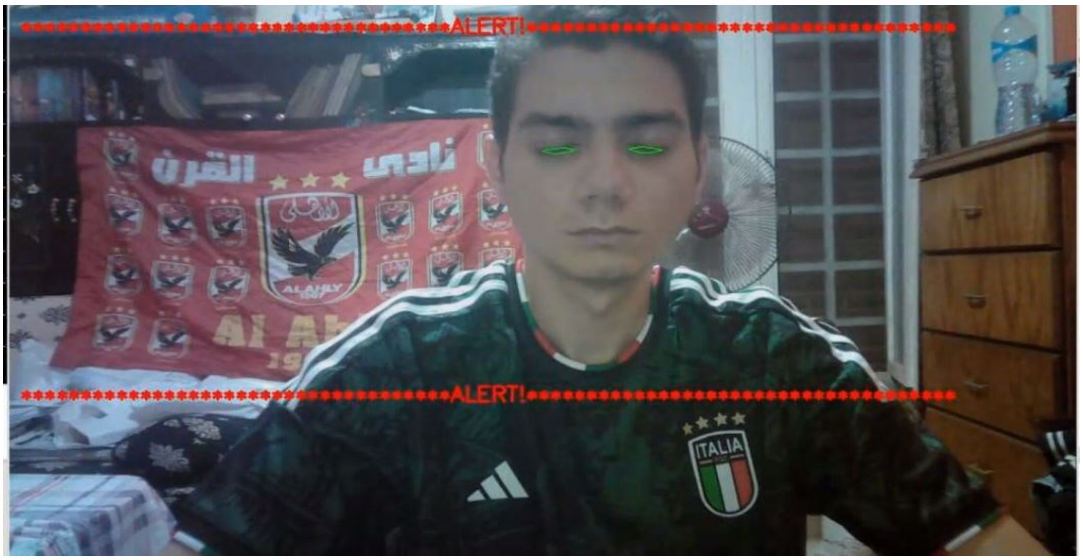


Figure 152 : drowsiness Alert Case

Chapter 10

SPI

10. SPI

Serial Peripheral Interface (SPI) is a synchronous serial communication interface specification used for short-distance communication, primarily in embedded systems. The interface was developed by Motorola in the mid-1980s and has become a de facto standard. Typical applications include Secure Digital cards and liquid crystal displays.

10.1. SPI PROTOCOL:

SPI has high-speed with full-duplex and synchronous communication bus, which is used for information transmission between microcontrollers and peripherals. SPI is mainly used for saving the chip ports and gap on Printed Circuit Boards(PCB) layout.

It works with Master-Slave configuration, in full duplex mode there will be single master device and single slave device. It requires only four lines, and the components used are Serial Data Input, Serial Data Output, Serial Clock (SCK), and Slave Select (SS).

When the master of SPI wants to send data to slave, then SS line will get low for selecting slave, the clock signal will be activated, so that both master and slave can use it at the same time. Master sends data to MOSI line (master's Serial Data Out and Slave's Serial Data Input) and it receives the data from MISO line (master's Serial Data Input and slave's Serial Data Output). Clock pulse was provided by Serial Clock and Serial Data Input, Serial Data Output, these all are based on the pulse to make the information transmission.

Data output is done through the Serial data output(SDO) line of master's either by rising or falling edge of the clock, and it is read by the slave in falling or rising edge followed. So, eight-bit data transfer needs at least eight times the clock signal changes.

10.1.1. Descriptions:

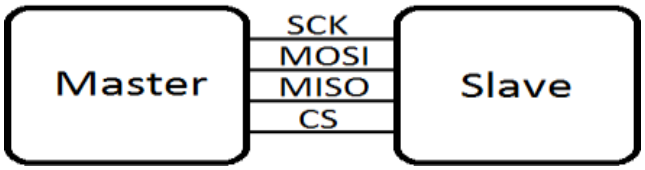


Figure 153 :SPI with Single Master& Single Slave SPI Signal

- Master In Slave Out (MISO):

Input of master and output of slave was configured as MISO line. It transfers serial data in only one direction, where in MSB is sent first. If the slave was not selected, then the MISO line will be placed in high impedance state.

Pin. Name	Master	Slave
SCK	Serial clock output from the master [SCK]	Input to the slave [SCK]
SDO	Serial data output from the master [MOSI]	Input to the slave [MOSI]
SDI	Serial data input from the slave[MISO]	Output from the slave [MISO]
SS	Optional slave selects [SS]	Slave Select [SS]

Table 1: SPI Interface Signals

- Master Out Slave In (MOSI)

Input to slave and output to master is done by MOSI line. Data is transmitted in single direction serially, with the MSB.

- Serial Clock (SCK)

The synchronization of data to and from the device is done using the serial clock, through MOSI & MISO signals. The duration of eight clock cycles, slave and master device are capable in exchanging the data a byte. The master device generates SCK, hence, it be an input to the device of slave.

- Slave Select (SS Bar)

The device of slave is selected using the select input line of slave, which is active low during data transfer and must stay low throughout the data transfer.

10.1.1.1. SPI Data transmission:

There are four modes of operation, they are from 0 through 3, they are used to control the data way which is clocked in or out of SPI device. SPI control register (SPCR) manages the configuration using two bits. The CPOL control bit specifies the clock polarity.

Different transfer formats are selected using the clock phase CPHA control bit. Same mode has to be maintained to ensure better communication between both master devices and slave devices, which can provide the required reconfiguration of master and to match the requirements of slaves with different peripheral.

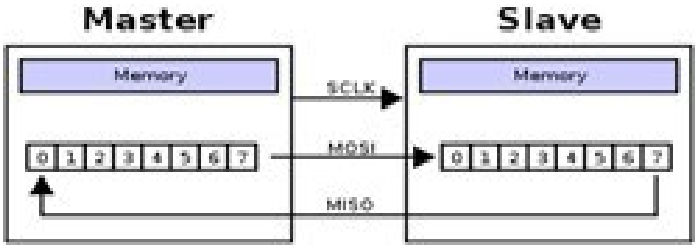


Figure 154 :SPI Data Transmission of 8-bit from Master to slave

Communication is always initiated by master, and will configure the clock first using that configure frequency that which must be less than or equal to frequency which is maximum then the slave device. The master will select a desired slave for communication by pulling a chip through SS line of a slave-peripheral that is particular to low state.

Shift register will organize for usage of data transfer with the given word sizes such as 8-bits in master device and slave device and they will connect in a ring. MOSI line is used by the master to shift the register value, then slave shifts the data in to shift register and then the data is sent to master by slave through the line MISO.

10.1.1.2. Receiving data

When you want to send data from the peripheral to the controller, you will use the separate MISO line. As the name suggests, data goes “in” the controller, and “out” of the peripheral. data will be sent back on the predetermined number of clock cycles. The controller will have to know beforehand whether the peripheral will return some data and how much. If it doesn’t, it won’t know how many clock signals it should send, and some data might get lost.

As Shown in Figure.149, when the peripheral sends capital E while it receives data from the controller. Capital E in ASCII is 01000101, so it will be sent as 10100010. As seen from the image below, the controller will generate two clock cycles. During the first, the controller will send data, and the peripheral will be idle. During the second cycle, the controller will be idle, and the peripheral will send some data back.

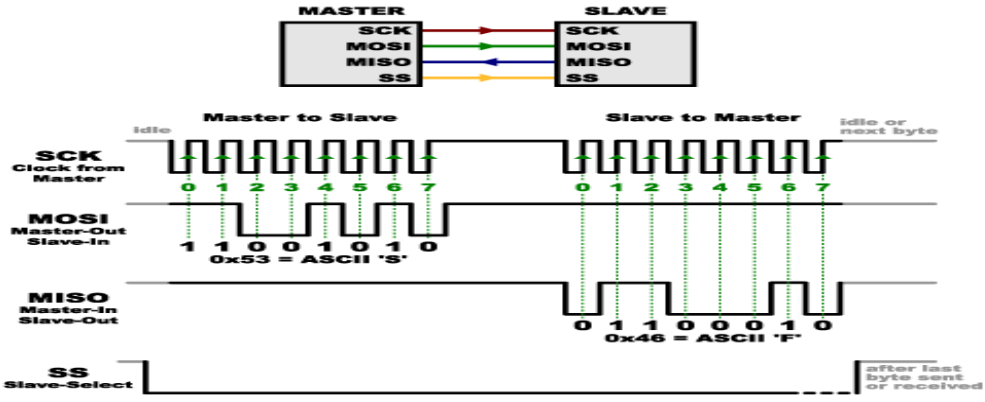


Figure 155 :Obtain SPI Data Receiving

10.1.1.3. Clock polarity and phase:

In addition to setting the clock frequency, the master must also configure the polarity and phase of the clock relative to the data as shown in figure.150. The SPI Block Guide names these two options CPOL and CPHA (for clock polarity and clock phase) respectively, a convention that most vendors

have also adopted.
The time chart is shown on the right. Synchronization is described in more detail below and applies to both master and slave devices.

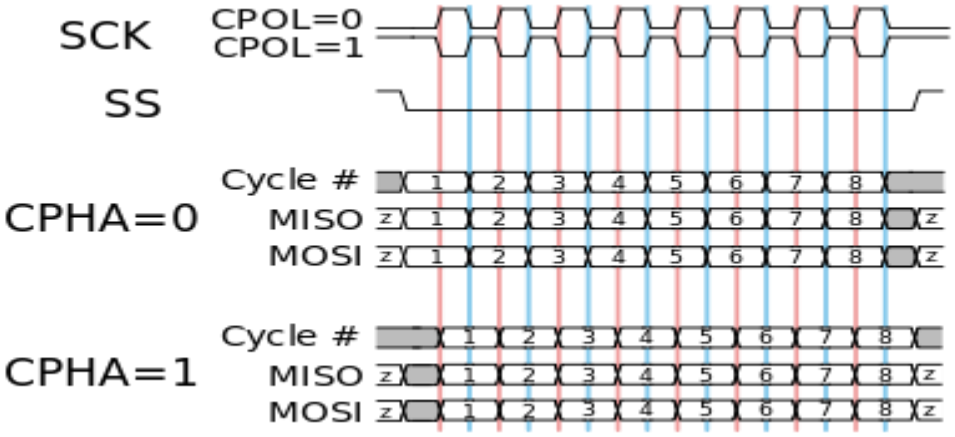


Figure 156 :Timing diagram showing clock polarity and phase. Red lines denote clock leading edges; and blue lines, trailing edges.

CPOL determines the polarity of the clock. The polarities can be converted with a simple inverter.

- CPOL=0 is a clock which idles at 0, and each cycle consists of a pulse of 1. That is, the leading edge is a rising edge, and the trailing edge is a falling edge.
- CPOL=1 is a clock which idles at 1, and each cycle consists of a pulse of 0. That is, the leading edge is a falling edge, and the trailing edge is a rising edge.

CPHA determines the timing (i.e. phase) of the data bits relative to the clock pulses. Conversion between these two forms is non-trivial.

- For CPHA=0, the "out" side changes the data on the trailing edge of the preceding clock cycle, while the "in" side captures the data on (or shortly after) the leading edge of the clock cycle. The outside holds the data valid until the trailing edge of the current clock cycle.

For the first cycle, the first bit must be on the MOSI line before the leading clock edge. An alternative way of considering it is to say that a CPHA=0 cycle consists of a half cycle with the clock idle, followed by a half cycle with the clock asserted.

- For CPHA=1, the "out" side changes the data on the leading edge of the current clock cycle, while the "in" side captures the data on (or shortly after) the trailing edge of the clock cycle. The outside holds the data valid until the leading edge of the following clock cycle. For the last cycle, the slave holds the MISO line valid until slave select is deasserted. An alternative way of considering it is to say that a CPHA=1 cycle consists of a half cycle with the clock asserted,

followed by a half cycle with the clock idle.

The MOSI and MISO signals are usually stable (at their reception points) for the half cycle until the next clock transition. SPI master and slave devices may sample data at different points in that half cycle.

10.1.1.4. MASTER WITH MULTIPLE SLAVES

1. Independent Slave Configuration

In the independent slave configuration, As Shown in Figure # , there is an independent chip select line for each slave. This is the way SPI is normally used. The master asserts only one chip select at a time.

Pull-up resistors between power source and chip select lines are recommended for systems where the master's chip select pins may default to an undefined state. When separate software routines initialize each chip select and communicate with its slave, pull-up resistors prevent other uninitialized slaves from responding.

Since the MISO pins of the slaves are connected together, they are required to be tri-state pins (high, low or high-impedance), where the high-impedance output must be applied when the slave is not selected. Slave devices not supporting tri-state may be used in independent slave configuration by adding a tri-state buffer chip controlled by the chip select signal. Since only a single signal line needs to be tristated per slave, one typical standard logic chip that contains four tristate buffers with independent gate inputs can be used to interface up to four slave devices to an SPI bus.

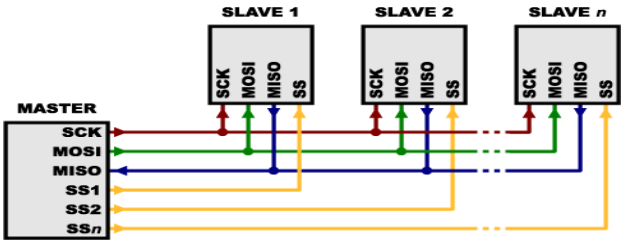


Figure 157 :Obtain Independent Slave Configuration

2. Daisy Chain Configuration

The first slave output being connected to the second slave input, etc. As shown in Figure.152, The SPI port of each slave is designed to send out during the second group of clock pulses an exact copy of the data it received during the first group of clock pulses. The whole chain acts as a communication shift register; daisy chaining is often done with shift registers to provide a bank of inputs or outputs through SPI. Each slave copies input to output in the next clock cycle until active low SS line goes high. Such a feature only requires a single SS line from the master, rather than a separate SS line for each slave.

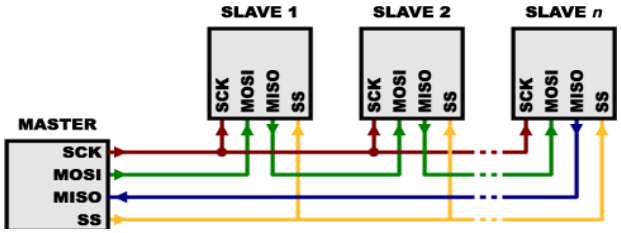


Figure 158 :Obtain Daisy Chain Configuration

Chapter 11

Real Time Operating System

RTOS

11. FREERTOS

11.1. Introduction

RTOS, a real-time operating system widely used in embedded systems and microcontrollers. real-time operating systems play a crucial role in systems that require precise and timely execution of tasks. They are designed to handle time-critical applications, where tasks must respond within specific deadlines. RTOS provides deterministic scheduling and resource management, ensuring that tasks are executed in a predictable and controlled manner. This is particularly important in domains such as automotive, aerospace, medical devices, industrial automation, and IoT, where safety, reliability, and real-time performance are critical.

RTOS offers benefits such as task scheduling, inter-task communication, synchronization mechanisms, and resource management, which enable developers to design complex systems with multiple tasks or threads. By using an RTOS like FreeRTOS, developers can focus on application logic and functionality while leveraging the built-in features and services provided by the operating system.

11.2. Introduction to FreeRTOS

FreeRTOS is an open-source real-time operating system designed for embedded systems and microcontrollers.

Key Features and Benefits of FreeRTOS:

- Task Management: Efficient creation and management of multiple tasks.

- Scheduling: Preemptive and cooperative scheduling mechanisms.

- Resource Management: Synchronization mechanisms for shared resource access.

- Inter-Task Communication: Various communication mechanisms for efficient data exchange.

- Memory Management: Dynamic memory allocation for optimized memory usage.

- Timer Services: Software timers for scheduling tasks or events.

- Portability: Support for a wide range of microcontrollers, architectures, and development tools.

- Scalability: Customizable to fit the specific needs of the embedded system.

Supported architectures and platforms:

FreeRTOS supports various architectures and platforms, including ARM Cortex-M series, AVR, MSP430, PIC32, Renesas RX, and more.

Comparison with other RTOS options:

When compared to other RTOS options, FreeRTOS stands out due to its open-source nature, portability, scalability, community support, and comprehensive documentation.

11.3. RTOS Fundamentals

11.3.1. Multitasking

The kernel is the core component within an operating system. Operating systems such as Linux employ kernels that allow users access to the computer seemingly simultaneously. Multiple users can execute multiple programs apparently concurrently. Each executing program is a task under control of the operating system. If an operating system can execute multiple tasks in this manner it is said to be multitasking. The use of a multitasking operating system can simplify the design of what would otherwise be a complex software application: The multitasking and inter-task communications features of the operating system allow the complex application to be partitioned into a set of smaller and more manageable tasks.

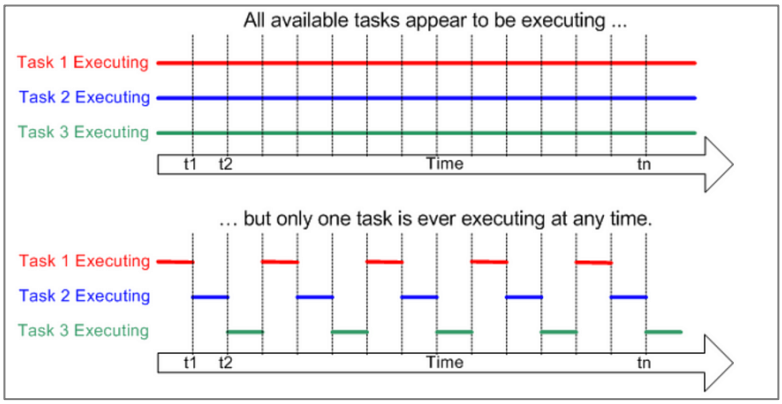


Figure 159: Contexts of Task in FreeRTOS

• The partitioning can result in easier software testing, work breakdown within teams, and code reuse.
• Complex timing and sequencing details can be removed from the application code and become the responsibility of the operating system.

11.3.2. Multitasking Vs Concurrency

A conventional processor can only execute a single task at a time - but by rapidly switching between tasks a multitasking operating system can make it appear as if each task is executing concurrently. This is depicted by Figure.153 which shows the execution pattern of three tasks with respect to time. The task names are color coded and written down the left hand. Time moves from left to right, with the colored lines showing which task is executing at any particular time. The upper diagram demonstrates the perceived concurrent execution pattern, and the lower the actual multitasking execution pattern.

11.3.3. Scheduling

The scheduler is the part of the kernel responsible for deciding which task should be executing at any

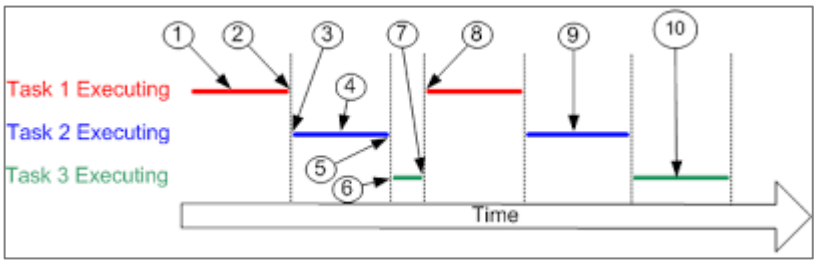


Figure 160:The Execution of Task in FreeRTOS

particular time. The kernel can suspend and later resume a task many times during the task lifetime.

The scheduling policy is the algorithm used by the scheduler to decide which task to execute at any point in time. The policy of a (nonreal time) multi user system will most likely allow each task a "fair" proportion of processor time. The policy used in real time embedded systems is described later. In addition to being suspended involuntarily by the RTOS kernel a task can choose to suspend itself. It will do this if it either wants to delay (sleep) for

a fixed period, or wait (block) for a resource to become available (e.g. a serial port) or an event to occur (eg. a key press). A blocked or sleeping task is not able to execute, and will not be allocated any processing time. Referring to the numbers in Figure.154:

• At (1) task 1 is executing.

• At (2) the kernel suspends task 1 ...

• ... and at (3) resumes task 2.

• While task 2 is executing (4), it locks a processor peripheral for it's own exclusive access.

• At (5) the kernel suspends task 2 ...

• ... and at (6) resumes task 3.

• Task 3 tries to access the same processor peripheral, finding it locked task 3 cannot continue so suspends itself at (7).

• At (8) the kernel resumes task 1.

• Etc.

• The next time task 2 is executing (9) it finishes with the processor peripheral and unlocks it .

• The next time task 3 is executing (10) it finds it can now access the processor peripheral and this time executes until suspended by the kernel.

11.3.4. Context Switching

As a task executes it utilizes the processor / microcontroller registers and accesses RAM and ROM just as any other program. These resources together (the processor registers, stack, etc.) comprise the task execution context. A task is a sequential piece of code - it does not know when it is going to get suspended or resumed by the kernel and does not even know when this has happened. Consider the example of a task being suspended

immediately before executing an instruction that sums the values contained within two processor registers. While the task is suspended other tasks will execute and may modify the processor register values. Upon resumption the task will not know that the processor registers have been altered - if it used the modified values the summation would result in an incorrect value. To prevent this type of error it is essential that upon resumption a task has a context identical to that immediately prior to its suspension. The operating system kernel is responsible for ensuring this is the case - and does so by saving the context of a task as it is suspended. When the task is resumed its saved context is restored by the operating system kernel prior to its execution. The process of saving the context of a task being suspended and restoring the context of a task being resumed is called context switching.

11.3.5. Real Time Applications

Real time operating systems (RTOS's) achieve multitasking using these same principals - but their objectives are very different to those of nonreal time systems. The different objective is reflected in the scheduling policy. Real time embedded systems are designed to provide a timely response to real world events. Events occurring in the real world can have deadlines before which the real time embedded system must respond and the RTOS scheduling policy must ensure these deadlines are met. To achieve this objective the software engineer must first assign a priority to each task. The scheduling policy of the RTOS is then to simply ensure that the highest priority task that is able to execute is the task given processing time. This may require sharing processing time "fairly" between tasks of equal priority if they are ready to run simultaneously.

11.3.6. Tasks in FreeRTOS

11.3.6.1. Concepts of Tasks

A real time application that uses an RTOS can be structured as a set of independent tasks. Each task executes within its own context with no coincidental dependency on other tasks within the system or the scheduler itself. Only one task within the application can be executing at any point in time and the real time scheduler is responsible for deciding which task this should be. The scheduler may therefore repeatedly start and stop each task (swap each task in and out) as the application executes. As a task has no knowledge of the scheduler activity it is the responsibility of the real time scheduler to ensure that the processor context (register values, stack contents, etc) when a task is swapped in is exactly that as when the same task was swapped out. To achieve this each task is provided with its own stack. When the task is swapped out the execution context is saved to the stack of that task so it can also be exactly restored when the same task is later swapped back in.

11.3.6.2. Task Summary

- Simple.
- No restrictions on use.
- Supports full preemption.
- Fully prioritized.
- Each task maintains its own stack resulting in higher RAM usage.

- Re-entrance must be carefully considered if using preemption

11.3.6.3. Task States

A task can exist in one of the following states:

Running: when a task is actually executing it is said to be in the Running state. It is currently utilizing the processor.

Ready: ready tasks are those that are able to execute (they are not blocked or suspended) but are not currently executing because a different task of equal or higher priority is already in the Running state.

Blocked: A task is said to be in the Blocked state if it is currently waiting for either a temporal or external event. For example, if a task calls vTaskDelay() it will block (be placed into the Blocked state) until the delay period has expired - a temporal event. Tasks can also block waiting for queue and semaphore events. Tasks in the Blocked state always have a 'timeout' period, after which the task will be unblocked. Blocked tasks are not available for scheduling.

Suspended: Tasks in the Suspended state are also not available for scheduling. Tasks will only enter or exit the suspended state when explicitly commanded to do so through the vTaskSuspend() and xTaskResume() API calls respectively. A 'timeout' period cannot be specified.

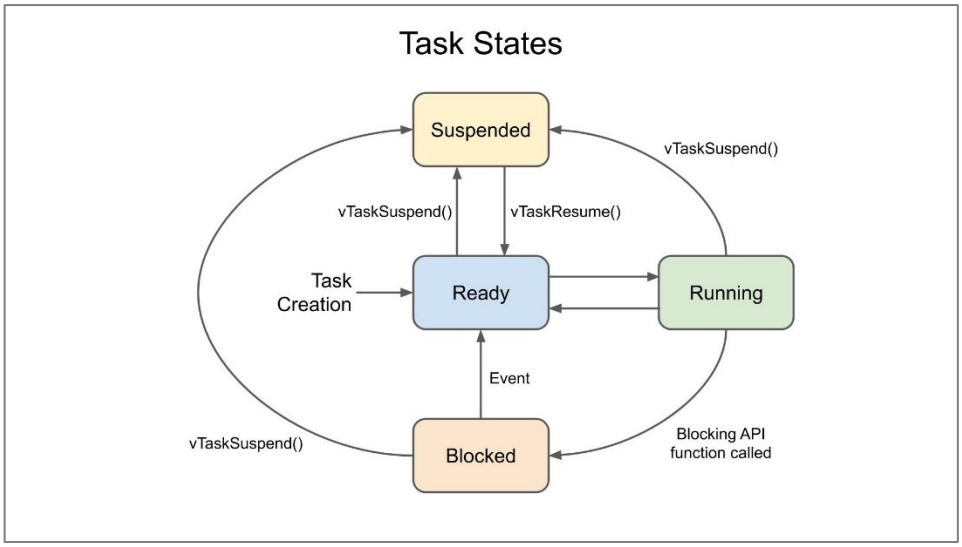


Figure 161:The Execution of Task in FreeRTOS

11.3.6.4. Task Priorities

Each task is assigned a priority from 0 to (configMAX PRIORITIES - 1). configMAX PRIORITIES is defined within **FreeRTOSConfig.h** and can be set on an application by application basis. The higher the value given to configMAX PRIORITIES the more RAM the FreeRTOS kernel will consume. Low priority numbers denote low priority tasks, with the default idle priority defined by tskIDLE PRIORITY as being zero. The scheduler will ensure that a task in the ready or running state will always be given processor time in preference to tasks of a lower priority that are also in the ready state. In other words, the task given processing time will always be the highest priority task that is able to run.

11.3.6.5. Implementing a Task

A task should have the following structure:

```
void vATaskFunction(void *pvParameters)

{
for (;;)
{
-- Task application code here. --
}
}
```

The type pdTASK CODE is defined as a function that returns void and takes a void pointer as it's only parameter. All functions that implement a task should be of this type. The parameter can be used to pass information of any type into the task - this is demonstrated by several of the standard demo application tasks. Task functions should never return so are typically implemented as a continuous loop. Again, see the RTOS demo application for numerous examples. Tasks are created by calling xTaskCreate() and deleted by calling vTaskDelete()

11.3.6.6. The Idle Task

The idle task is created automatically when the scheduler is started. The idle task is responsible for freeing memory allocated by the RTOS to tasks that have since been deleted. It is therefore important in applications that make use of the vTaskDelete() function to ensure the idle task is not starved of processing time. The activity visualization utility can be used to check the micro-controller time allocated to the idle task. The idle task has no other active functions so can legitimately be starved of micro-controller time under all other conditions. It is possible for application tasks to share the idle task priority. (tskIDLE PRIORITY)

11.3.7. Task Management

11.3.7.1. xTaskHandle

Type by which tasks are referenced. For example, a call to xTaskCreate returns (via a pointer parameter) an xTaskHandle variable that can then be used as a parameter to vTaskDelete to delete the task.

11.3.7.2. xTaskCreate

Prototype

```
portBASE_TYPE xTaskCreate(pdTASK_CODE pvTaskCode,
                          const portCHAR * const pcName,
                          unsigned portSHORT usStackDepth,
                          void *pvParameters,
                          unsigned portBASE_TYPE uxPriority,
                          xTaskHandle *pvCreatedTask);
```

Semantics

Create a new task and add it to the list of tasks that are ready to run.

Parameters

PvTaskCode: Pointer to the task entry function. Tasks must be implemented to never return (i.e. continuous loop).

pcName: A descriptive name for the task. This is mainly used to facilitate debugging. Max length defined by configMAX TASK NAME LEN.

usStackDepth: The size of the task stack specified as the number of variables the stack can hold - not the number of bytes. For example, if the stack is 16 bits wide and usStackDepth is defined as 100, 200 bytes will be allocated for stack storage. The stack depth multiplied by the stack width must not exceed the maximum value that can be contained in a variable of type size t.

pvParameters Pointer that will be used as the parameter for the task being created.

uxPriority: The priority at which the task should run.

pvCreatedTask: Used to pass back a handle by which the created task can be referenced.

Returns

pdPASS if the task was successfully created and added to a ready list, otherwise an error code defined in the file **projdefs.h**

11.3.7.3. vTaskDelete

Prototype

```
void vTaskDelete(xTaskHandle pxTask);
```

INCLUDE vTaskDelete must be defined as 1 for this function to be available. See the configuration section for more information.

Semantics

Remove a task from the RTOS real time kernels management. The task being deleted will be removed from all ready, blocked, suspended and event lists.

NOTE: The idle task is responsible for freeing the kernel allocated memory from tasks that have been deleted. It is therefore important that the idle task is not starved of microcontroller processing time if your application makes any calls to vTaskDelete(). Memory allocated by the task code is not automatically freed, and should be freed before the task is deleted. See the demo application file death.c for sample code that utilises vTaskDelete().

Parameters

pxTask the handler of the task to be deleted. Passing NULL will cause the calling task to be deleted.

11.3.7.4. vTaskDelay

Prototype

```
void vTaskDelay(portTickType xTicksToDelay);
```

INCLUDE vTaskDelay must be defined as 1 for this function to be available. See the configuration section for more information.

Semantics

Delay a task for a given number of ticks. The actual time that the task remains blocked depends on the tick rate. The constant portTICK RATE MS can be used to calculate real time from the tick rate - with the resolution of one tick period.

Parameters

xTicksToDelay the amount of time, in tick periods, that the calling task should block.

11.3.7.5. vTaskDelayUntil

Prototype

```
void vTaskDelayUntil(portTickType *pxPreviousWakeTime, portTickType xTimeIncrement);
```

IINCLUDE vTaskDelayUntil must be defined as 1 for this function to be available. See the configuration section for more information.

Semantics

Delay a task until a specified time. This function can be used by cyclical tasks to ensure a constant execution frequency. This function differs from vTaskDelay() in one important aspect: vTaskDelay() specifies a time at which the task wishes to unblock relative to the time at which vTaskDelay() is called, whereas vTaskDelayUntil() specifies an absolute time at which the task wishes to unblock. vTaskDelay() will cause a task to block for the specified number of ticks from the time vTaskDelay() is called. It is therefore difficult to use vTaskDelay() by itself to generate a fixed execution frequency as the time between a task unblocking following a call to vTaskDelay() and that task next calling vTaskDelay() may not be fixed [the task may take a different path though the code between calls, or may get interrupted or preempted a different number of times each time it executes]. Whereas vTaskDelay() specifies a wake time relative to the time at which the function is called, vTaskDelayUntil() specifies the absolute (exact) time at which it wishes to unblock. It should be noted that vTaskDelayUntil() will return immediately (without blocking) if it is used to specify a wake time that is already in the past.

Therefore a task using vTaskDelayUntil() to execute periodically will have to re-calculate its required wake time if the periodic execution is halted for any reason (for example, the task is temporarily placed into the Suspended state) causing the task to miss one or more periodic executions. This can be detected by checking the variable passed by reference as the pxPreviousWakeTime parameter against the current tick count. This is however not necessary under most usage scenarios. The constant configTICK RATE MS can be used to calculate real time from the tick rate - with the resolution of one tick period.

Parameters

PxPreviousWakeTime: Pointer to a variable that holds the time at which the task was last unblocked. The variable must be initialised with the current time prior to its first use (see the example below). Following this the variable is automatically updated within vTaskDelayUntil().

xTimeIncrement The cycle time period. The task will be unblocked at time (*pxPreviousWakeTime + xTimeIncrement). Calling vTaskDelayUntil with the same xTimeIncrement parameter value will cause the task to execute with a fixed interval period.

11.3.7.6. uxTaskPriorityGet

Prototype

```
unsigned portBASE_TYPE uxTaskPriorityGet(xTaskHandle pxTask);
```

INCLUDE vTaskPriorityGet must be defined as 1 for this function to be available. See the configuration section for more information.

Semantics

Obtain the priority of any task.

Parameters

pxTask Handle of the task to be queried. Passing a NULL handle results in the priority of the calling task being returned.

Returns

The priority of pxTask.

11.3.7.7. vTaskPrioritySet

Prototype

```
void vTaskPrioritySet(xTaskHandle pxTask, unsigned portBASE_TYPE uxNewPriority);
```

INCLUDE vTaskPrioritySet must be defined as 1 for this function to be available. See the configuration section for more information.

Semantics

Set the priority of any task. A context switch will occur before the function returns if the priority being set is higher

than the currently executing task.

Parameters

pxTask Handle to the task for which the priority is being set. Passing a NULL handle results in the priority of the calling task being set.

uxNewPriority the priority to which the task will be set.

11.3.7.8. vTaskSuspend

Prototype

```
void vTaskSuspend(xTaskHandle pxTaskToSuspend);
```

INCLUDE vTaskSuspend must be defined as 1 for this function to be available. See the configuration section for more information.

Semantics

Suspend any task. When suspended a task will never get any microcontroller processing time, no matter what its priority. Calls to vTaskSuspend are not accumulative - i.e. calling vTaskSuspend() twice on the same task still only requires one call to

Parameters

pxTaskToSuspend Handle to the task being suspended. Passing a NULL handle will cause the calling task to be suspended.

11.3.7.9. vTaskResume

Prototype

```
void vTaskResume(xTaskHandle pxTaskToResume);
```

INCLUDE vTaskSuspend must be defined as 1 for this function to be available. See the configuration section for more information.

Semantics

Resumes a suspended task. A task that has been suspended by one of more calls to vTaskSuspend() will be made available for running again by a single call to vTaskResume().

Parameters

pxTaskToResume Handle to the task being readied.

11.3.7.10. vTaskResumeFromISR

Prototype

```
portBASE_TYPE vTaskResumeFromISR(xTaskHandle pxTaskToResume);
```

INCLUDE vTaskSuspend and INCLUDE xTaskResumeFromISR must be defined as 1 for this function to be available. See the configuration section for more information.

Semantics

A function to resume a suspended task that can be called from within an ISR. A task that has been suspended by one of more calls to vTaskSuspend() will be made available for running again by a single call to xTaskResumeFromISR(). vTaskResumeFromISR() should not be used to synchronize a task with an interrupt if there is a chance that the interrupt could arrive prior to the task being suspended - as this can lead to interrupts being missed. Use of a semaphore as a synchronization mechanism would avoid this eventuality.

Parameters

pxTaskToResume Handle to the task being readied.

Returns

pdTRUE if resuming the task should result in a context switch, otherwise pdFALSE. This is used by the ISR to determine if a context switch may be required following the ISR.

11.3.8. Task Synchronization:

In a multitasking environment it is sometimes necessary to synchronize the order of operations between two or more tasks. For example, some tasks may require repeating another task and synchronizing access to shared memory before execution. The RTOS uses the following methods to achieve task synchronization and resource sharing.

11.3.8.1. Semaphores:

Semaphore Variable or abstract data type used to control access to a shared resource by multiple processes in a concurrent system, such as a multitasking operating system. There are two types of semaphores:

- **Counting semaphores**:

They are used for event counting and for resource management.

Prototype

```
SemaphoreHandle_t xSemaphoreCreateCounting(UBaseType_t uxMaxCount,
UBaseType_t uxInitialCount);
```

INCLUDE semphr. h must be defined as 1 for this function to be available.

configSUPPORT_DYNAMIC_ALLOCATION must be set to 1 in FreeRTOSConfig.h, or left undefined (in which case it will default to 1), for this RTOS API function to be available.

Semantics

Counting semaphores are generally used for two purposes: counts. In this usage scenario, the event

handler “gives” the semaphore (incrementing the semaphore count) each time an event occurs and the handler activity “takes”; Semaphore each time an event is processed (thereby decreasing the value of the semaphore count). The counter value is then the difference between the number of events that have occurred and the number of events that have been processed. In this case, it is desirable that the initial count is zero as shown in figure.156.

resource management:In this usage scenario, the value of the counter indicates the number of available resources. In order to take control of a resource, the activity must first acquire the semaphore, which reduces the value of the semaphore count. When the counter value reaches zero, there are no more free resources. If the job completes with a resource that "returns" the semaphore, it increments the semaphore count.In this case, it is desirable that the initial value of the counter is equal to the maximum value of the counter, which indicates that all resources are free.

Parameters

uxMaxCount: The maximum count value that can be reached. When the semaphore reaches this value it can no longer be 'given'.

uxInitialCount: The count value assigned to the semaphore when it is created

Returns

If the semaphore is created successfully then a handle to the semaphore is returned. If the semaphore cannot be created because the RAM required to hold the semaphore cannot be allocated then NULL is returned.

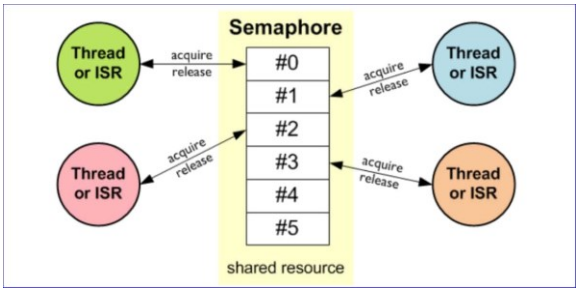


Figure 162: obtain Counting Semaphore process

• Binary semaphores:

used to mutually exclusion and synchronization targets.

Prototype

```
SemaphoreHandle_t xSemaphoreCreateBinary( void );
```

INCLUDE semphr.h must be defined as 1 for this function to be available.

configSUPPORT_DYNAMIC_ALLOCATION must be set to 1 in FreeRTOSConfig.h, or left undefined (in which case it will default to 1), for this RTOS API function to be available.

Semantics

The priority of a task that 'takes' a mutex can potentially be raised if another task of higher priority attempts to obtain the same mutex. The task that owns the mutex 'inherits' the priority of the task attempting to 'take' the same mutex. This means the mutex must always be 'given' back - otherwise the higher priority task will never be able to obtain the mutex, and the lower priority task will never 'disinherit' the priority. An example of a mutex being used to implement mutual exclusion is provided on the xSemaphoreTake() documentation page as shown in figure #.

Both mutex and binary semaphores are referenced by variables of type SemaphoreHandle_t and can be used in any task level API function that takes a parameter of that type. Unlike mutexes, binary semaphores can be used in interrupt service routines.

Returns

NULL The semaphore could not be created because there was insufficient FreeRTOS heap available.

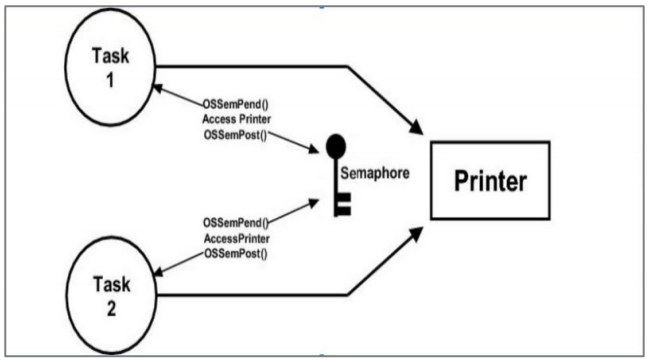


Figure 163 :obtain Binary Semaphore process

11.3.8.2. Mutex:

Mutex is a synchronization mechanism for enforcing limits on access to a resource in an environment where there are many threads of execution. A lock is designed to enforce a mutual exclusion concurrency control policy. As Shown in figure.158, When used for mutual exclusion the mutex acts like a token that is used to guard a resource. When a task wishes to access a resource it must first obtain (‘take’) the token. When it has finished with the resource it must ‘give’ the token back.

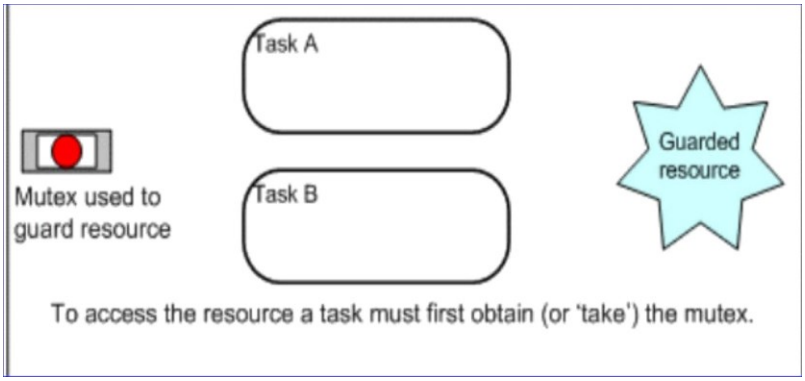


Figure 164 : obtain Mutex process

Prototype

```
SemaphoreHandle_t xSemaphoreCreateMutex( void )
```

INCLUDE semphr. h must be defined for this function to be available.

configSUPPORT_DYNAMIC_ALLOCATION and configUSE_MUTEXES must both be set to 1 in FreeRTOSConfig.h for xSemaphoreCreateMutex() to be available.

Semantics

The priority of a task that 'takes' a mutex will be temporarily raised if another task of higher priority attempts to obtain the same mutex. The task that owns the mutex 'inherits' the priority of the task attempting to 'take' the same mutex. This means the mutex must always be 'given' back - otherwise the higher priority task will never be able to obtain the mutex, and the lower priority task will never 'disinherit' the priority.

Mutexes are taken using xSemaphoreTake(), and given using xSemaphoreGive(). xSemaphoreTakeRecursive() and xSemaphoreGiveRecursive() can only be used on mutexes created using xSemaphoreCreateRecursiveMutex().

Mutexes and binary semaphores are very similar but have some subtle differences: Mutexes include a priority inheritance mechanism, binary semaphores do not. This makes binary semaphores the better choice for implementing synchronisation (between tasks or between tasks and an interrupt), and mutexes the better choice for implementing simple mutual exclusion.

Returns

If the mutex type semaphore was created successfully then a handle to the created mutex is returned. If the mutex was not created because the memory required to hold the mutex could not be allocated then NULL is returned.

- **xSemaphoreTake:**

Prototype

```
xSemaphoreTake( SemaphoreHandle_t xSemaphore, TickType_t xTicksToWait );
```

INCLUDE semphr.h must be defined for this function to be available.

Semantics

Macro to obtain a semaphore. The semaphore must have previously been created with a call to xSemaphoreCreateBinary(), xSemaphoreCreateMutex() or xSemaphoreCreateCounting().

This macro must not be called from an ISR. xQueueReceiveFromISR() can be used to take a semaphore from within an interrupt if required, although this would not be a normal operation. Semaphores use queues as their underlying mechanism, so functions are to some extent interoperable.

Parameters

xSemaphore:A handle to the semaphore being taken - obtained when the semaphore was created.

xTicksToWait: The time in ticks to wait for the semaphore to become available. The macro portTICK_PERIOD_MS can be used to convert this to a real time. A block time of zero can be used to poll the semaphore.

If INCLUDE_vTaskSuspend is set to '1' then specifying the block time as portMAX_DELAY will cause the task to block indefinitely (without a timeout).

Returns

pdTRUE if the semaphore was obtained. pdFALSE if xTicksToWait expired without the semaphore becoming available.

- **xSemaphoreGive:**

Prototype

```
xSemaphoreGive( SemaphoreHandle_t xSemaphore );
```

INCLUDE semphr.h must be defined for this function to be available.

Semantics

Macro to release a semaphore. The semaphore must have previously been created with a call to xSemaphoreCreateBinary(), xSemaphoreCreateMutex() or xSemaphoreCreateCounting().This must not be used from an ISR.

Parameters

xSemaphore :A handle to the semaphore being released. This is the handle returned when the semaphore was created.

Returns

pdTRUE if the semaphore was released. pdFALSE if an error occurred. Semaphores are implemented using queues. An error can occur if there is no space on the queue to post a message - indicating that the semaphore was not first obtained correctly.

11.3.9. Task Inter-Communication

11.3.9.1. Mailbox:

A mailbox is simply a storage location, large enough to hold a single variable of type ADDR, whose access is controlled so that it can be safely used for multiple tasks. A task can write to the mailbox. As shown in figure.159 It is then full and no tasks can be sent to it until a task has read the mailbox or the mailbox is reset. Attempting to send to a full mailbox or read from an empty mailbox may result in an error or pause.

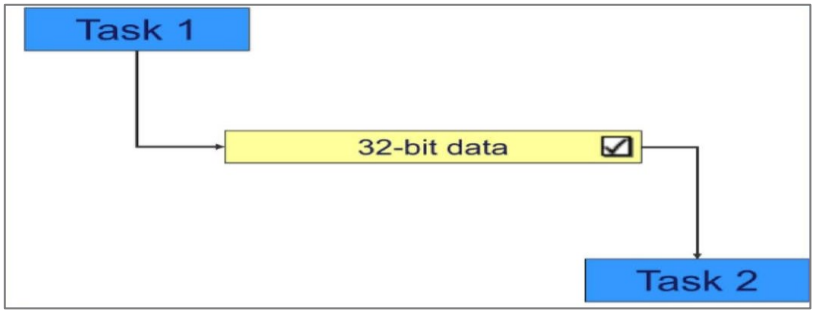


Figure 165 : obtain Mailbox process

11.3.9.2. Messages Queues:

A message queue is some fixed or maximum size buffer controlled by the RTOS and also a queue for tasks that are waiting for messages. The size and number of buffers are specified when creating the message queue.
Queue is a form of communication between main tasks. As shown in figure.160 they can be used to send messages between tasks and between interrupts and tasks. In most cases they are used as thread-safe FIFO (first in, first out) buffers, with new data sent towards the back of the queue.

Prototype

```
QueueHandle_t xQueueCreate( UBaseType_t uxQueueLength,
                            UBaseType_t uxItemSize );
```

INCLUDE semphr. h must be defined for this function to be available.

configSUPPORT_DYNAMIC_ALLOCATION must be set to 1 in FreeRTOSConfig.h, or left undefined (in which case it will default to 1), for this RTOS API function to be available.

Semantics

Macro to release a semaphore. The semaphore must have previously been created with a call to

xSemaphoreCreateBinary(), xSemaphoreCreateMutex() or xSemaphoreCreateCounting().This must not be used from an ISR.

Parameters

uxQueueLength :The maximum number of items the queue can hold at any one time.

uxItemSize :The size, in bytes, required to hold each item in the queue.

Returns

If the queue is created successfully then a handle to the created queue is returned. If the memory required to create the queue could not be allocated then NULL is returned.

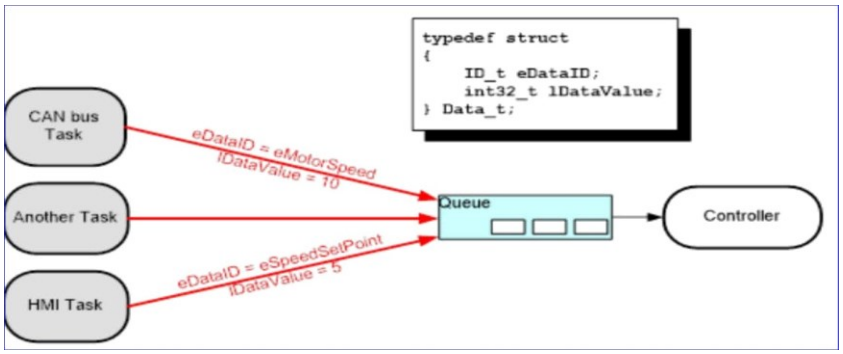


Figure 166 : obtain message queue process

11.4. Software FreeRTOS Code configuration:

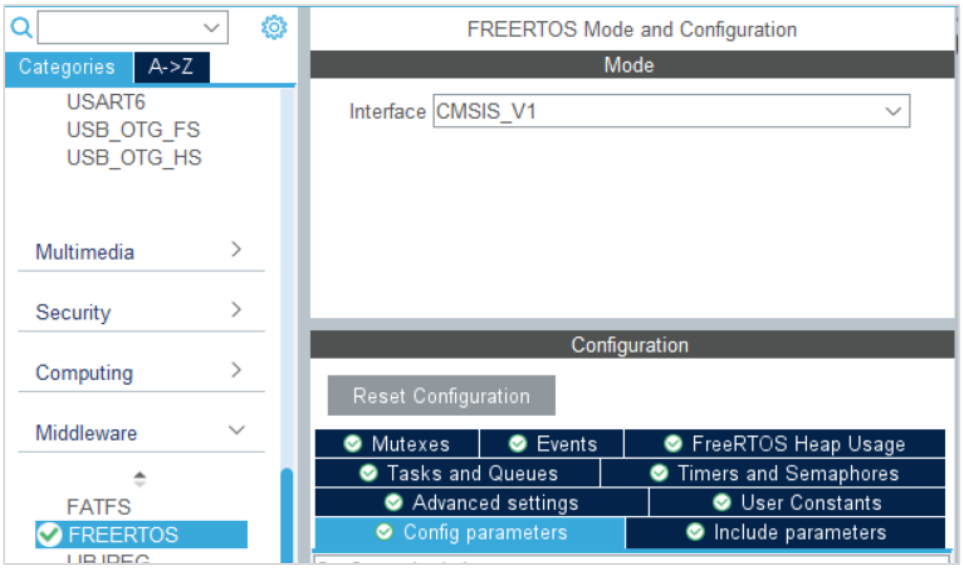


Figure 167 :FreeRTOS Cube MX configuration

We set all parameters to default value and configure all configurations manually, to create the tasks and queues as below; but we also create them manually.

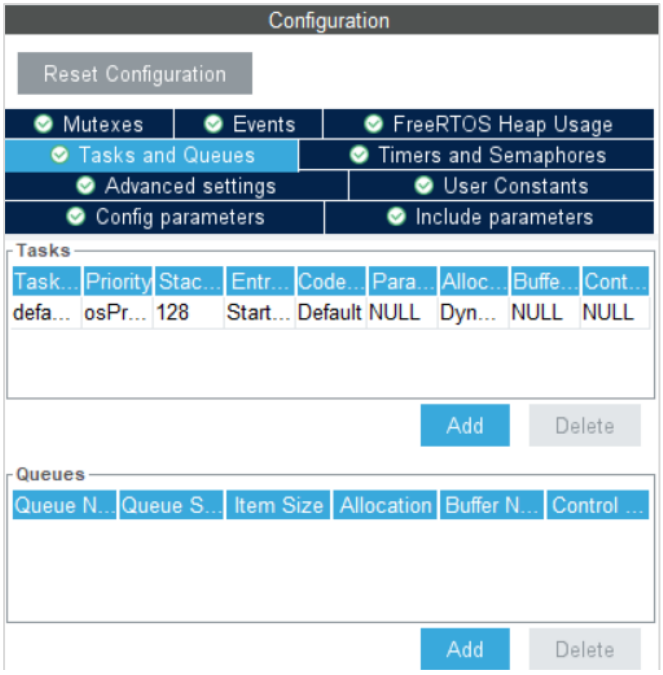


Figure 168 :FreeRTOS Cube MX configuration adding tasks and queues

11.5. FreertosConfig.h file configuration

```
#define configUSE_PREEMPTION                    1
#define configSUPPORT_STATIC_ALLOCATION         1
#define configSUPPORT_DYNAMIC_ALLOCATION        1
#define configUSE_IDLE_HOOK                     1
#define configUSE_TICK_HOOK                     0
#define configCPU_CLOCK_HZ                      ( SystemCoreClock )
#define configTICK_RATE_HZ                      ((TickType_t)1000)
#define configMAX_PRIORITIES                    ( 7 )
#define configMINIMAL_STACK_SIZE                ((uint16_t)256)
#define configTOTAL_HEAP_SIZE                   ((size_t)32768)
#define configMAX_TASK_NAME_LEN                 ( 16 )
#define configGENERATE_RUN_TIME_STATS           1
#define configUSE_TRACE_FACILITY                1
#define configUSE_STATS_FORMATTING_FUNCTIONS    1
#define configUSE_16_BIT_TICKS                  0
#define configUSE_MUTEXES                       1
#define configQUEUE_REGISTRY_SIZE               8
#define configCHECK_FOR_STACK_OVERFLOW          2
#define configUSE_RECURSIVE_MUTEXES             1
#define configUSE_MALLOC_FAILED_HOOK            1
#define configUSE_APPLICATION_TASK_TAG          1
#define configUSE_COUNTING_SEMAPHORES           1
#define configUSE_PORT_OPTIMISED_TASK_SELECTION 1

#define configMESSAGE_BUFFER_LENGTH_TYPE        size_t
/* Co-routine definitions. */
#define configUSE_CO_ROUTINES                   0
#define configMAX_CO_ROUTINE_PRIORITIES         ( 2 )

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */
#define INCLUDE_vTaskPrioritySet            1
#define INCLUDE_uxTaskPriorityGet           1
#define INCLUDE_vTaskDelete                 1
#define INCLUDE_vTaskCleanUpResources       0
#define INCLUDE_vTaskSuspend                1
#define INCLUDE_vTaskDelayUntil             0
#define INCLUDE_vTaskDelay                  1
```

Figure 169 :FreeRTOSConfig.h file configuration

1) Firstly, we use FreeRTOS to implement our Project, mainly we use 5 tasks:

   - first 4 tasks run sequentially to implement the algorithm ABS responding to Prescan (Simulink) data.

```
xTaskCreate(vTaskReceiveUART  , "Receive UART", configMINIMAL_STACK_SIZE*4  , NULL,1, &Task1Handle);
xTaskCreate(vTaskMakeAlgorithm, "Make Algorithm", configMINIMAL_STACK_SIZE*4, NULL,2, &Task2Handle);
xTaskCreate(vTaskUpdateABS    , "Update ABS", configMINIMAL_STACK_SIZE*4    , NULL,3, &Task3Handle);
xTaskCreate(vTaskSendUART     , "Send UART", configMINIMAL_STACK_SIZE*4     , NULL,4, &Task4Handle);
```

   - last task suspended all time and Interrupt will resume the suspension as an action in Callback function of the CAN ISR to take action responding to the algorithm of face detection data from the raspberry pi.

```
xTaskCreate(vTaskReceiveCAN   , "Receive CAN", configMINIMAL_STACK_SIZE*4   , NULL,5, &Task5Handle);
```

2) we use concept queue with fixed size, as the first 4 tasks run sequentially and result of one is the input for the follower, so we deliver the data between them using queues functions (xQueueSend, xQueueReceive).

```
xUARTRecieveDataQueue = xQueueCreate(10, sizeof(InputData_t ));
xMakeAlgorithmQueue   = xQueueCreate(10, sizeof(flags_t));
xUpdateABSQueue       = xQueueCreate(10, sizeof(ControlData_t));
```

3) before starting the schedular we suspend the fifth task until ISR resume it.

```
xTaskCreate(vTaskReceiveCAN   , "Receive CAN", configMINIMAL_STACK_SIZE*4   , NULL,5, &Task5Handle);
/* USER CODE END RTOS_THREADS */

vTaskSuspend( Task5Handle );
/* Start scheduler */
osKernelStart();
```

4) We use Mutex for sharing resource UART1 which may send receive in the same time and make problems

```
myMutexUARTHandle = xSemaphoreCreateMutex();
```

```
if( xSemaphoreTake( myMutexUARTHandle, ( TickType_t ) portMAX_DELAY) == pdTRUE )
{



    HAL_UART_Receive(&huart1, (uint8_t*)BufferUARTReceive,strlen(ourBuffer), 200);

    sscanf(BufferUARTReceive, "*%f;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f#",
           &ReceivedData.desiredSpeed, &ReceivedData.Heading,
           &ReceivedData.rangeOfLRR, &ReceivedData.relativeSpeedOfLRR,
           &ReceivedData.rangeOfSRR, &ReceivedData.relativeSpeedOfSRR,
           &ReceivedData.rangeOfFLR, &ReceivedData.relativeSpeedOfFLR, &ReceivedData.azimuthAngleOfFLR,
           &ReceivedData.rangeOfFRR, &ReceivedData.relativeSpeedOfFRR, &ReceivedData.azimuthAngleOfFRR,
           &ReceivedData.rangeOfBRR, &ReceivedData.relativeSpeedOfBRR, &ReceivedData.azimuthAngleOfBRR,
           &ReceivedData.rangeOfBLR, &ReceivedData.relativeSpeedOfBLR, &ReceivedData.azimuthAngleOfBLR  );

    xQueueSend(xUARTRecieveDataQueue, &ReceivedData, portMAX_DELAY);
    xQueueSend(xMakeAlgorithmQueue2,  &ReceivedData, portMAX_DELAY);

    xSemaphoreGive( myMutexUARTHandle );
```

- First task called UART Receive which receive the data coming from Simulink such as ranges and the relative speed of radars, then send it through queue to following Task.

- Second task called Make Algorithm which take the data from the queue ,that set from UART Receive task, then calculate the region, set the flags of regions then send the result of the flags through Second Queue.

- Third task called Update ABS which receive the result of the flags then take actions corresponding to the flags set before like braking, throttle, lane …. etc. Then send these actions to following function through the third queue.

- Fourth task called UART Send which receive the actions from queue then send these data to Simulink to take action into Prescan through UART.

- Fifth task is completely separated from the flow of above tasks and has the highest priority and periodicity, as it is suspended always until the interrupt of the CAN, the callback of ISR resume this suspension and make this task run immediately as it indicates very critical time representing the driver status and run the ABS algorithm then after finishing suspend the task again.

Chapter 12

Drowsiness Detection

12. Introduction:

12.1. Motivation:

Numerous studies have confirmed the correlation between driver drowsiness and the rising incidence of accidents, establishing it as a significant contributing factor. While accurately quantifying the precise number of drowsiness-related accidents remains challenging, it is highly probable that the current estimates are conservative. This highlights the importance of research endeavors aimed at mitigating the risks posed by drowsiness-induced accidents. To date, researchers have focused on developing models that establish connections between drowsiness and specific indicators associated with both the vehicle and the driver.

12.1.1. Drowsiness

Drowsiness refers to a transitional state between wakefulness and sleep, characterized by symptoms that significantly impact task performance. These symptoms include delayed response time, intermittent lack of awareness, and microsleeps (brief episodes of unintentional sleep lasting over 500 ms). In fact, prolonged fatigue can impair performance levels similarly to the effects of alcohol consumption. These symptoms pose a grave danger while driving as they greatly increase the likelihood of drivers missing road signs or exits, drifting into other lanes, or causing accidents by losing control of their vehicles. Drowsiness is considered an intermediate stage between being awake and being asleep, involving progressive impairment of awareness accompanied by a desire or inclination to sleep.

Historically, approaches to detecting drowsiness have made assumptions about relevant behaviors, focusing on factors such as blink rate, eye closure, and yawning. The automotive industry has also made attempts to develop various systems for predicting driver drowsiness, although only a few commercial products are currently available. However, these systems often neglect to consider driver performance and individual characteristics, despite the recognition that different individuals exhibit distinct driving styles. Our system is specifically designed to adapt to changes in the driver's behavior, acknowledging the variability among drivers

12.2. Problem Description:

Accidental crashes resulting in injuries have gained significant global attention, ranking as the 8th leading cause of death according to the World Health Organization (WHO) survey. Sleepiness plays a pivotal role in disrupting neurological function and stands as a major risk factor for road traffic accidents. Shockingly, approximately 1.3 million deaths occur each year worldwide as a consequence of road traffic accidents, contributing to a substantial 3% loss in the gross domestic product of many countries. The US National Highway Traffic Safety Administration has estimated that drowsiness is responsible for roughly 100,000 road accidents annually on a global scale. These accidents account for

over 1,500 fatalities, more than 70,000 injuries, and an enormous monetary loss of $12.5 billion each year. Remarkably, 1 in 4 vehicle accidents is attributed to drowsy driving, indicating the grave impact of this issue. Moreover, an alarming statistic reveals that 1 in 25 adult drivers admit to having fallen asleep at the wheel within the past 30 days. What's truly unsettling is that drowsy driving encompasses more than just falling asleep while driving. Even a momentary lapse of consciousness, where the driver fails to fully concentrate on the road, falls under the realm of drowsy driving

12.3. Problem Solution:

Due to the relevance of this problem, we believe it is important to develop a solution for drowsiness detection, especially in the early stages to prevent accidents. Drowsy driving is one of the major causes behind fatal road accidents. Driver fatigue has been the one of the main issue for countless mishaps due to tiredness, tedious, road condition, and unfavorable climate situations. If the drowsiness of the driver can be predicted at initial stages, and if the driver can be alerted of the same, then a number of accidents can be reduced.

There are signs that suggest a driver is drowsy, such as

1) Frequently yawning

2) Inability to keep eyes open

3) Swaying the head forward

4) Face complexion changes.

Recent studies have categorized drowsiness detection techniques into three main categories: physiological measures, vehicle-based measures, and face analysis.

- Physiological measures rely on assessing changes in the body's elements, such as heart rate, body temperature, and pulse rate, which can indicate fatigue in a driver. Commonly used physiological signals for assessing driver fatigue include ECG (electrocardiogram), EEG (electroencephalogram), EMG (electromyogram), and EOG (electrooculogram). However, a significant drawback of using physiological methods is the requirement for the driver to wear sensors, which can potentially hinder comfort and convenience.
- Vehicle-based measures detect drowsiness by monitoring the behavior of the vehicle itself, including steering wheel movement, random braking, speed variations, and more. Sensors are installed in vehicle components to track driving performance and identify patterns that may indicate drowsiness. However, a limitation of vehicle-based methods is that external factors like adverse weather conditions, road conditions, or the influence of medication can alter the vehicle's behavior, potentially affecting the accuracy of drowsiness detection.
- Behavioral measures or face analysis techniques rely on observing facial expressions and movements using machine learning and computer vision. By analyzing facial cues, such as eye closure, yawning, or changes in facial expressions, drowsiness can be detected. This approach utilizes advanced technologies in machine learning and computer vision (CV) to interpret and recognize signs of drowsiness based on facial characteristics.

Overall, these three categories provide different approaches for detecting drowsiness, each with its own advantages and limitations. Researchers continue to explore and refine these techniques to improve the accuracy and reliability of drowsiness detection systems.

Finally, CV-based methods offer the advantage of being nonintrusive, making them an attractive option for real-time monitoring systems aimed at detecting driver fatigue. These methods leverage various visual cues, such as eyelid movement, gaze movement, head movement, and facial expressions, to assess the driver's level of fatigue. By analyzing these visual cues, CV-based systems can provide timely and continuous monitoring of driver fatigue without requiring any intrusive sensors or equipment. This nonintrusive nature allows for a seamless integration of the detection system into the driving environment, ensuring minimal disruption to the driver's experience.

12.4. Description

A drowsiness detection system uses computer vision to analyze a driver's eyes in real-time. It measures the duration of eye closure and develops an algorithm to detect drowsiness in advance. If the percentage of eye closure below a predefined threshold (e.g., 22-25% within a specific time period), the system identifies drowsiness. To accomplish this, the system employs HAAR classifier Cascade files, which contain classifiers for detecting faces and eyes.

In our project, we will detect Drowsiness by Plug a camera into the car cabin to detect the Drowsiness, and then forward these images to Raspberry Pi to decide whether the driver is drowsy or not based on our models

We have three scenarios: -

▪ If the driver wasn't drowsy nothing will happen.

▪ The second one is if the driver was drowsy for 10 consecutive frames, we will get his attention back by warning him with a loud sound using (Buzzer), and send 1 on CAN bus.

▪ The third one is if the driver was drowsy for 20 consecutive frames, we will start braking at road side, and send 2 on CAN bus.

12.5. Requirements:

A. Hardware Requirements

- Processor – Raspberry Pi 3 Model B+.
- Camera – USB Webcam.
- SD Card – 32GB.
- Sound – Buzzer.

- CAN network to link the system to STM32F429.

B. Software Requirements

- Operating System – Raspberry Pi OS(Raspbian).
- Language – Python.
- Importing necessary libraries
    1. scipy.spatial.distance is imported from SciPy for calculating distances between eye landmarks.
    2. face_utils is imported from imutils for working with facial landmarks.
    3. imutils provides convenience functions for resizing and working with images.
    4. dlib is a library used for face and facial landmark detection.
    5. cv2 is OpenCV, a popular computer vision library.
    6. os is used for system-level operations.
    7. can is a library for interacting with CAN buses.
- Tools - VNC Viewer, PUTTY.

12.6. Methodology:

Our system comprises several modules, including video acquisition, face detection, eye detection, and drowsiness detection. Additionally, it incorporates a Raspberry Pi, which serves as a single-board computer, and a buzzer that functions as an alarm to alert drowsy drivers.

To begin, we use a USB webcam connected to the Raspberry Pi to acquire video footage, which is then converted into a series of images or frames. The first step in the process involves detecting the driver's face using OpenCV and the Haar algorithm. Subsequently, we perform eye detection within the identified facial region using a technique called Region of Interest (ROI).

Once the eyes have been successfully detected, we proceed to determine whether they are open or closed. If the eyes are consistently detected as open in each frame, no further action is taken. However, if the eyes are detected as closed and fall below a predetermined threshold value, it indicates that the driver is experiencing drowsiness. In such cases, a sound alarm is triggered to alert the driver.

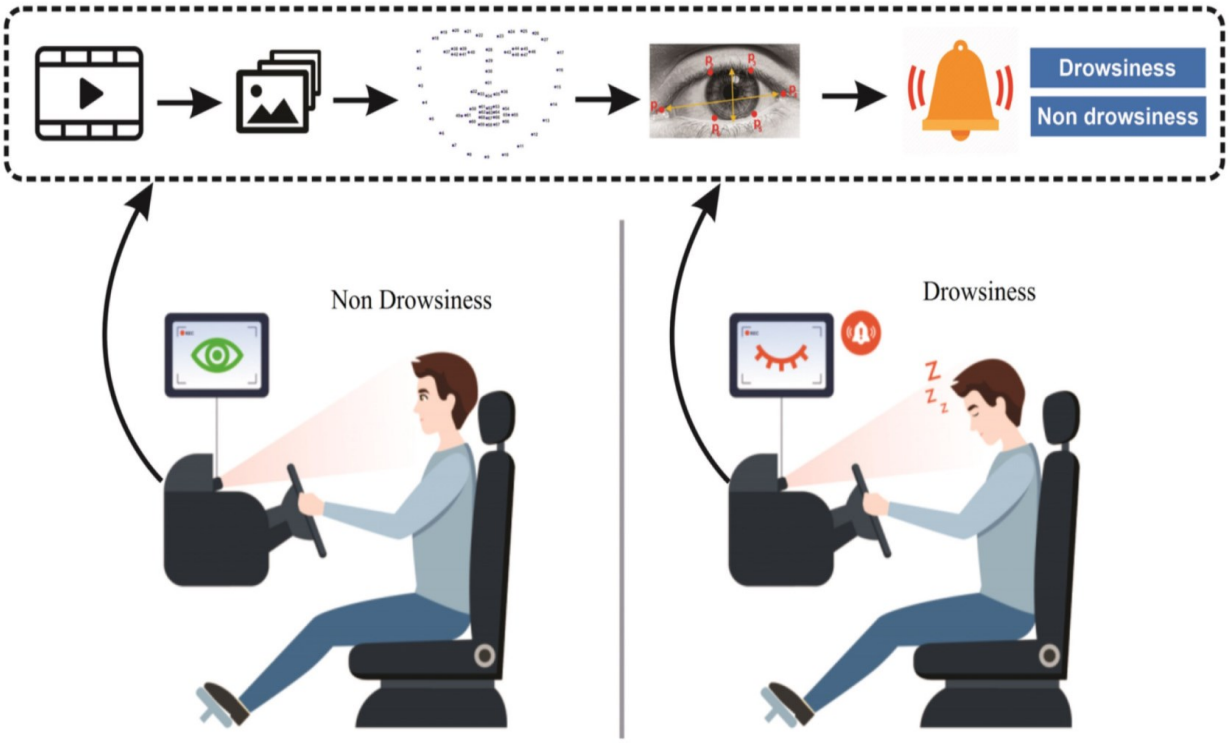


Figure 170: Frames are sent of the driver to update the driver's status

12.7. Analysis and Design:

12.7.1. Definitions:

Facial landmarks
are specific points on the face that correspond to key facial features such as the eyes, nose, mouth, and eyebrows. These landmarks are anatomical points that can be used to analyze and track facial expressions, shape, and alignment. in the code, we use the dlib library to detect and extract facial landmarks. The shape predictor model from dlib's "shape_predictor_68_face_landmarks.dat" file is employed for this purpose. The specific indices for the left and right eye landmarks are extracted from the FACIAL_LANDMARKS_68_IDXS dictionary.

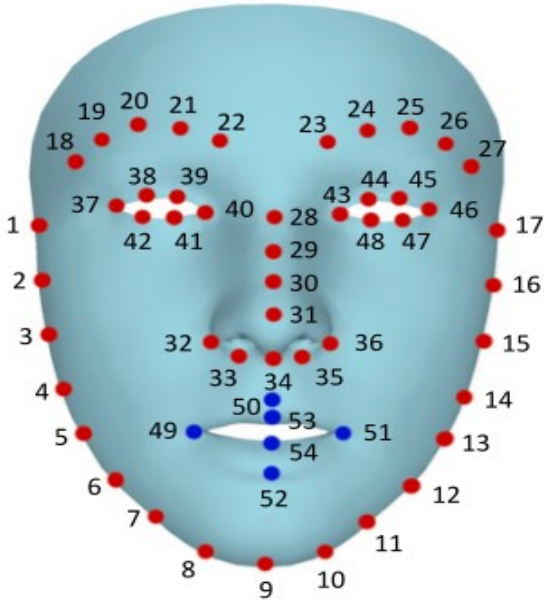


Figure 171 facial landmarks

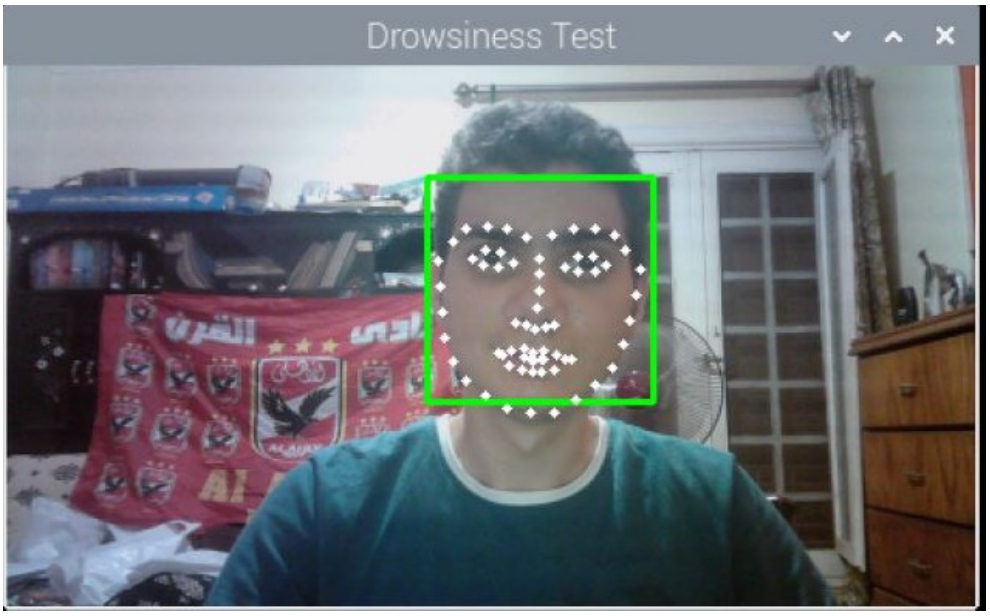


Figure 172 example of face landmarks

Euclidean distance

the Euclidean distance is used to calculate the Eye Aspect Ratio (EAR) and to measure the distances between specific points on the eye region.

EAR Calculation: The EAR is computed using the Euclidean distance between certain landmarks on the eye region. The formula for calculating the EAR is: EAR = (A + B) / (2C)

where A, B, and C are Euclidean distances between specific landmarks on the eye. These landmarks typically include the inner and outer corners of the eye and the top and bottom points of the eye region.

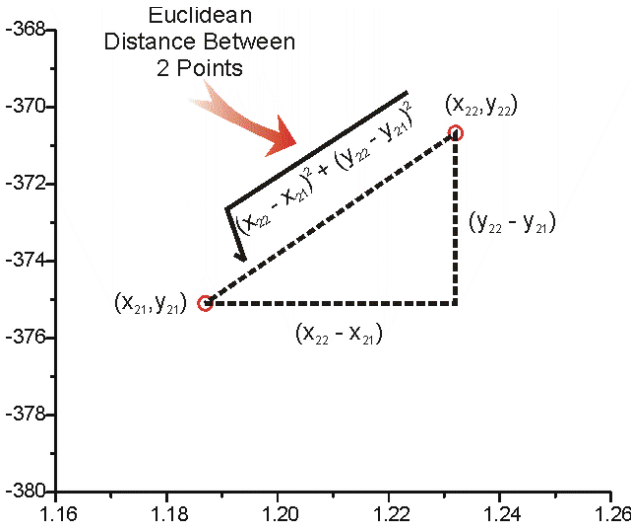


Figure 173 Euclidean distance

Distance calculation for eye regions:
The Euclidean distance is also used to measure the distances between different landmarks on the eye region. For example, in the code, the distances between the inner and outer corners of the eye are calculated to determine the open or closed state of the eye.

By utilizing the Euclidean distance, we can quantify the spatial relationships between various points on the eye region. This information is crucial for analyzing eye openness, determining drowsiness, and distinguishing between different eye states (open, closed, partially closed). The Euclidean distance provides a straightforward and effective way to measure these distances and derive meaningful insights about the eyes' condition in the context of drowsiness detection.

HAAR cascade

Haar Cascade is based on the concept of features which are proposed by Paul Viola and Michael Jones in their paper “Rapid Object Detection using a Boosted Cascade of Simple Features” in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It can be used to detect objects from an image or a video.

12.7.1.1. What is Viola Jones algorithm?

Viola Jones algorithm is named after two computer vision researchers who proposed the method in 2001, Paul Viola and Michael Jones in their paper, “Rapid Object Detection using a Boosted Cascade of Simple Features”. Despite being an outdated framework, Viola-Jones is quite powerful, and its application has proven to be exceptionally notable in real-time face detection. This algorithm is painfully slow to train but can detect faces in real-time with impressive speed.

Given an image (this algorithm works on grayscale image), the algorithm looks at many smaller subregions and tries to find a face by looking for specific features in each subregion. It needs to check many different positions and scales because an image can contain many faces of various sizes. Viola and Jones used Haar-like features to detect faces in this algorithm.

The Viola Jones algorithm has four main steps, which we shall discuss in the sections to follow:

1- Selecting Haar-like features.

2- Creating an integral image.

3- Running AdaBoost training.

4- Creating classifier cascades.

12.7.1.2. What are Haar-Like Features?

In the 19th century a Hungarian mathematician, Alfred Haar gave the concepts of Haar wavelets, which are a sequence of rescaled “square-shaped” functions which together form a wavelet family or basis. Voila and Jones adapted the idea of using Haar wavelets and developed the so-called Haar-like features.

Haar-like features are digital image features used in object recognition. All human faces share some universal properties of the human face like the eyes region is darker than its neighbour pixels, and the nose region is brighter than the eye region.

A simple way to find out which region is lighter or darker is to sum up the pixel values of both regions and compare them. The sum of pixel values in the darker region will be smaller than the sum of pixels in the lighter region. If one side is lighter than the other, it may be an edge of an eyebrow or sometimes the middle portion may be shinier than the surrounding boxes, which can be interpreted as a nose This can be accomplished using Haar-like features and with the help of them, we can interpret the different parts of a face.

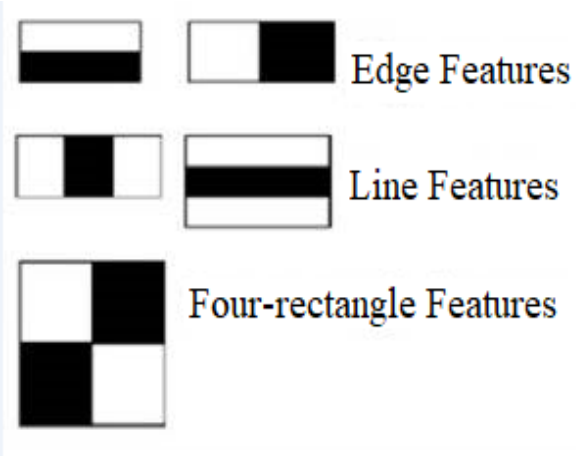


Figure 174: HAAR-Like features.

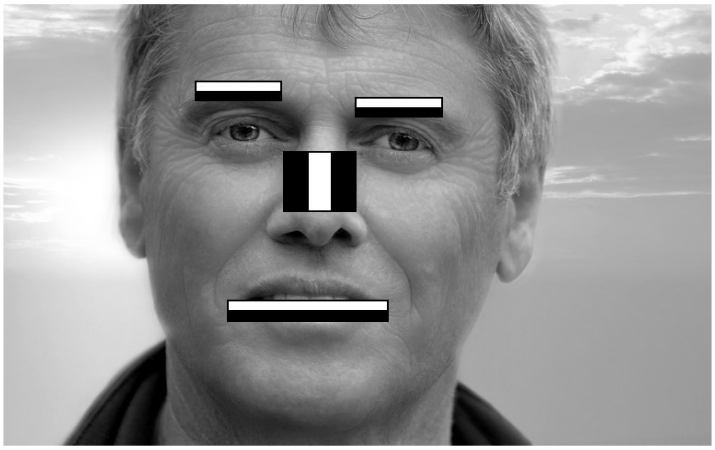


Figure 175: Applying HAAR-Like features on a person.

There are 3 types of Haar-like features that Viola and Jones identified in their research:

1- Edge features.

2- Line-features.

3- Four-sided features.

Edge features and Line features are useful for detecting edges and lines respectively. The four-sided features are used for finding diagonal features.

The value of the feature is calculated as a single number: the sum of pixel values in the black area minus the sum of pixel values in the white area. The value is zero for a plain surface in which all the pixels have the same value, and thus, provide no useful information.

12.7.1.3. What are Integral Images?

In the previous section, we have seen that to calculate a value for each feature, we need to perform computations on all the pixels inside that particular feature. In reality, these calculations can be very intensive since the number of pixels would be much greater when we are dealing with a large feature.

The integral image plays its part in allowing us to perform these intensive calculations quickly so we can understand whether a feature of several features fit the criteria.

An integral image (also known as a summed-area table) is the name of both a data structure and an algorithm used to obtain this data structure. It is used as a quick and efficient way to calculate the sum of pixel values in an image or rectangular part of an image.

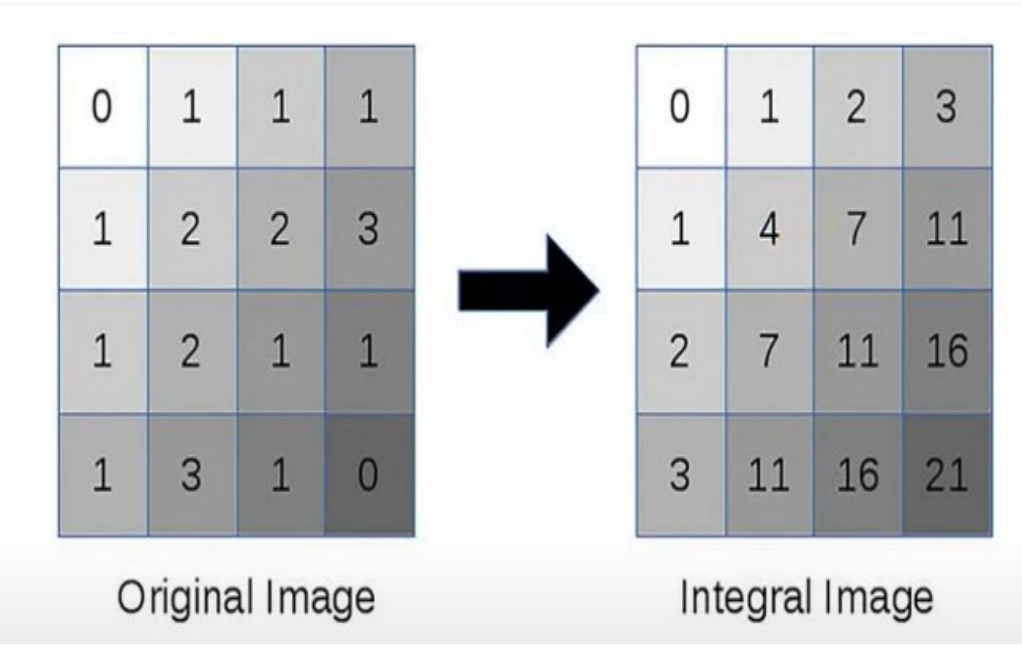


Figure 176: Original image and integral image.

Now, the rectangle of interest here is the cyanate colored which represents a haar-like feature from the massively high number of features, in order to get over the massive calculation of pixels as their numbers are enormous; after converting the original image to the integral image you will perform only three operations which are: 21+1-11-3

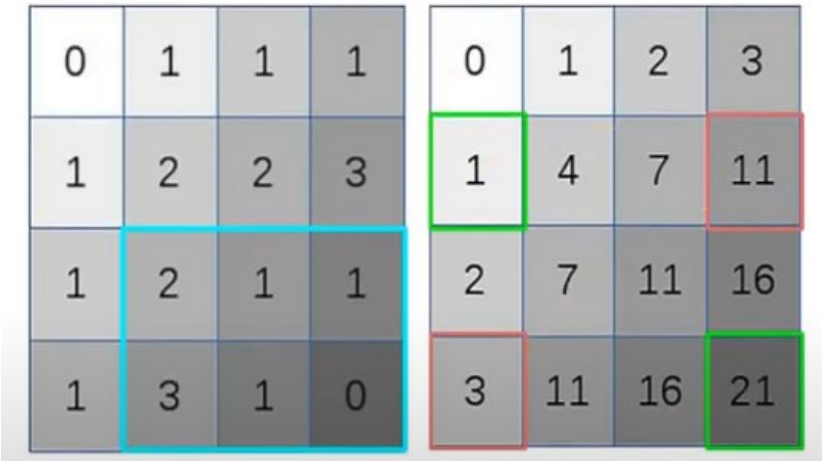


Figure 177: Resultant pixels with lower operations done.

21: represents summation of all pixels in the original image.

11: represents summation of all pixels above the rectangle of interest in the original image.

3: summation of the left column to the cyanate-colored rectangle.

12.7.1.4. How is AdaBoost used in viola jones algorithm?

Next, we use a Machine Learning algorithm known as AdaBoost. But why do we even want an algorithm?

The number of features that are present in the 24×24 detector window is nearly 160,000, but only a few of these features are important to identify a face. So we use the AdaBoost algorithm to identify the best features in the 160,000 features.

In the Viola-Jones algorithm, each Haar-like feature represents a weak learner. To decide the type and size of a feature that goes into the final classifier, AdaBoost checks the performance of all classifiers that you supply to it.

To calculate the performance of a classifier, you evaluate it on all subregions of all the images used for training. Some subregions will produce a strong response in the classifier. Those will be classified as positives, meaning the classifier thinks it contains a human face. Subregions that don't provide a strong response don't contain a human face, in the classifiers opinion. They will be classified as negatives.

The classifiers that performed well are given higher importance or weight. The final result is a strong classifier, also called a boosted classifier, that contains the best performing weak classifiers.

So when we're training the AdaBoost to identify important features, we're feeding it information in the form of training data and subsequently training it to learn from the information to predict. So ultimately, the algorithm is setting a minimum threshold to determine whether something can be classified as a useful feature or not.

12.7.1.5. What are Cascading Classifiers?

Maybe the AdaBoost will finally select the best features around say 2500, but it is still a time-consuming process to calculate these features for each region. We have a 24×24 window which we slide over the input image, and we need to find if any of those regions contain the face. The job of the cascade is to quickly discard non-faces, and avoid wasting precious time and computations. Thus, achieving the speed necessary for real-time face detection.

We set up a cascaded system in which we divide the process of identifying a face into multiple stages. In the first stage, we have a classifier which is made up of our best features, in other words, in the first stage, the subregion passes through the best features such as the feature which identifies the nose bridge or the one that identifies the eyes. In the next stages, we have all the remaining features.

When an image subregion enters the cascade, it is evaluated by the first stage. If that stage evaluates the subregion as positive, meaning that it thinks it's a face, the output of the stage is maybe.

When a subregion gets a maybe, it is sent to the next stage of the cascade and the process continues as such till we reach the last stage.

If all classifiers approve the image, it is finally classified as a human face and is presented to the user as a detection.

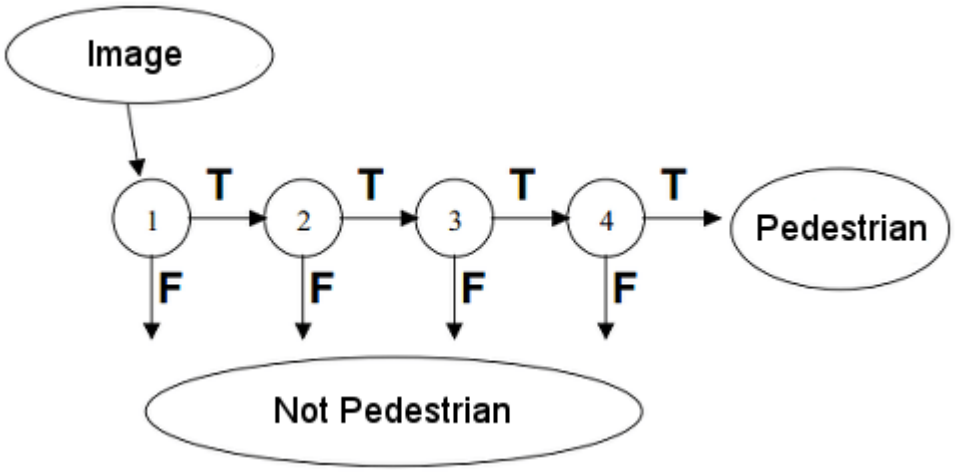


Figure 178: Applying cascaded classifier with different sizes on the image.

Now how does it help us to increase our speed? Basically, If the first stage gives a negative evaluation, then the image is immediately discarded as not containing a human face. If it passes the first stage but fails the second stage, it is discarded as well. Basically, the image can get discarded at any stage of the classifier.

Now, the result is as follows:

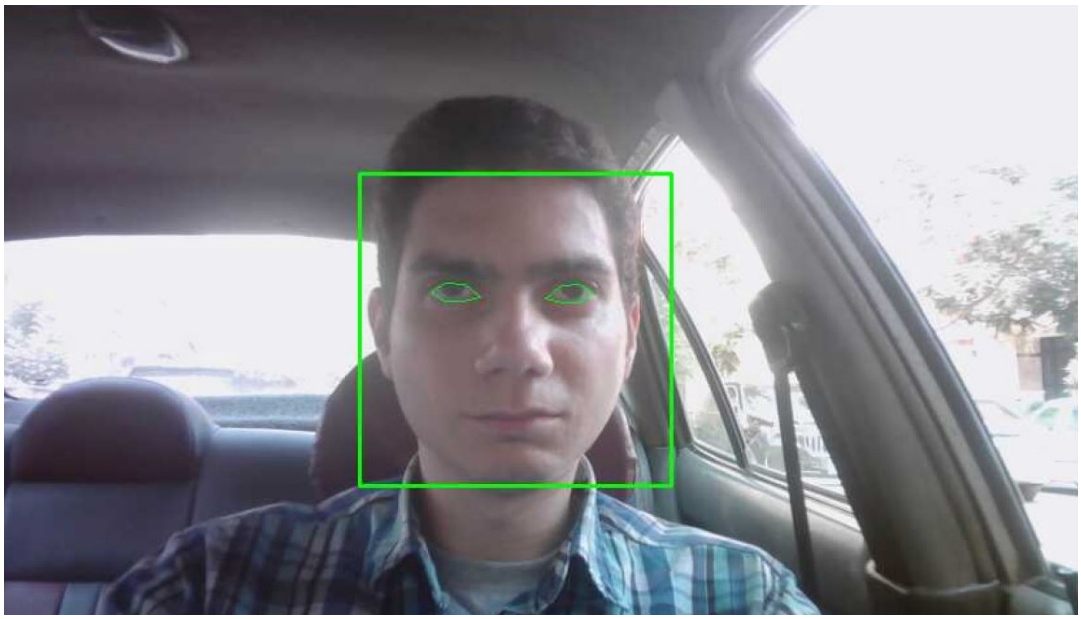


Figure 179: Face detection by HAAR-Like algorithm.

Approach

1) **Face Detection**: This module takes input from the camera and tries to detect a face in the video input. The detection of the face is achieved through the Haar classifiers mainly, the Frontal face cascade classifier. The face is detected in a rectangle format and converted to grayscale image and stored in the memory which can be used for training the mode.

2) Eye Detection: Since the model works on building a detection system for drowsiness we need to focus on the eyes to detect drowsiness.

Eye detection: is performed using facial landmarks obtained from the dlib library the code utilizes the dlib library and a pre-trained shape predictor model to detect facial landmarks.

Extracting Eye Regions: Once the facial landmarks are detected, the code extracts the left and right eye regions from the landmarks. This is achieved by specifying the appropriate indices for the eye regions within the facial landmark array.

Eye Aspect Ratio (EAR): After extracting the eye regions, the code can calculate the Eye Aspect Ratio (EAR). The EAR is a measure of eye openness and is used to determine if the eyes are open or closed. It is calculated based on the distances between specific points on the eye region, such as the inner corner, outer corner, and top and bottom points.

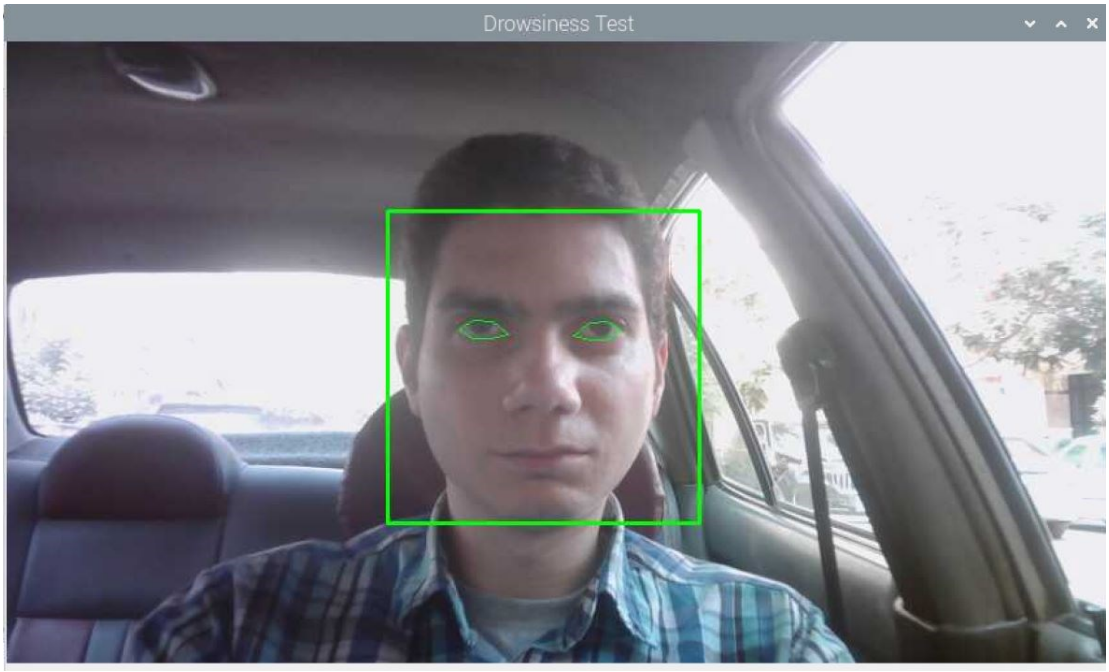


Figure 180 face and eye detection

3) Face Tracking: Due to the real-time nature of the project, we need to track the faces continuously for any form of distraction. Hence the faces are continuously detected during the entire time.

4) Eye Tracking: The input to this module is taken from the previous module the eyes state is determined through EAR algorithm.

5) Drowsiness detection: A predefined threshold value is set to determine the threshold below which the eyes are considered closed and indicative of drowsiness. If the calculated EAR falls below this threshold, it suggests that the eyes are closed or partially closed, indicating drowsiness. There is an alert levels based on the duration of low EAR values. If the EAR remains below the threshold for a certain number of frames, the code triggers different alert levels.

a. Warning Level: If the EAR remains below the threshold for a moderate duration (10 frames), a warning message is displayed on the frame to alert the user.

b. Full Brake Level: If the EAR remains below the threshold for an extended duration (20 frames), indicating prolonged drowsiness, a full brake action is triggered. This action involves sending a command through the CAN bus to initiate full braking in a connected vehicle system.

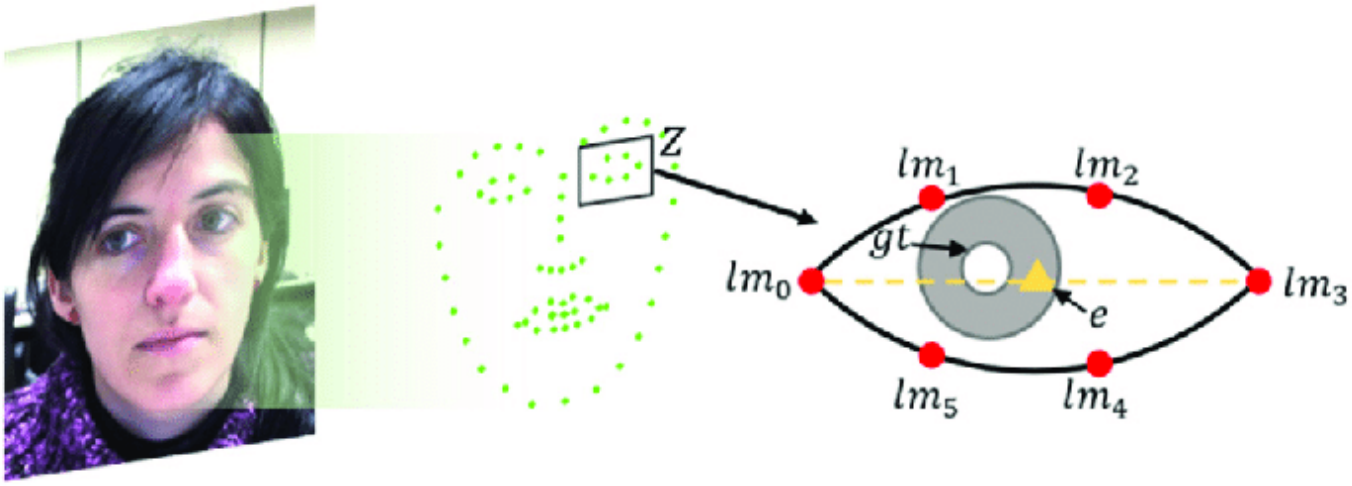


Figure 181: Points extraction from the face landmarks to define the EAR.

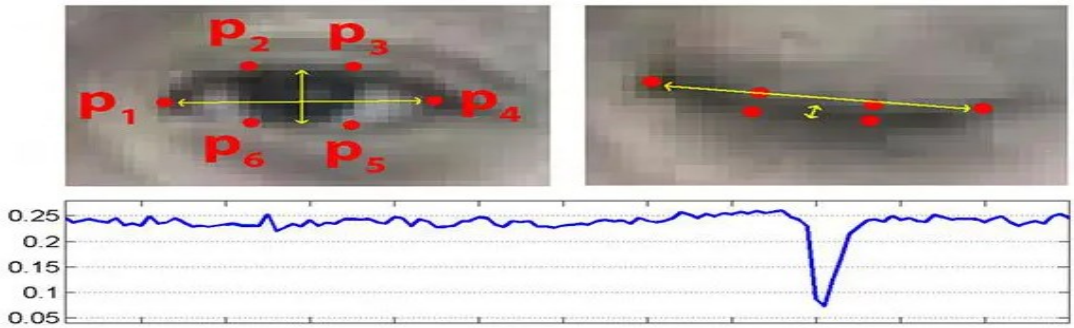


Figure 182: Top-left: A visualization of eye landmarks when the eye is open. Top-right: Eye landmarks when the eye is closed. Bottom: Plotting the eye aspect ratio over time. The dip in the eye aspect ratio indicates

EAR Calculation Formula:

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

Where,

A is the distance between the 2-points (p2 and p6).

B is the distance between the 2-points (p3 and p5).

C is the distance between 2-points (p1 and p4).

The Eye Aspect Ratio provides a numerical value that represents the degree of eye openness. As the eyes close or partially close, the EAR value decreases. It can be used as a measure to detect drowsiness or fatigue, as drowsiness often results in a decrease in the Eye Aspect Ratio.

6) Architecture Diagram

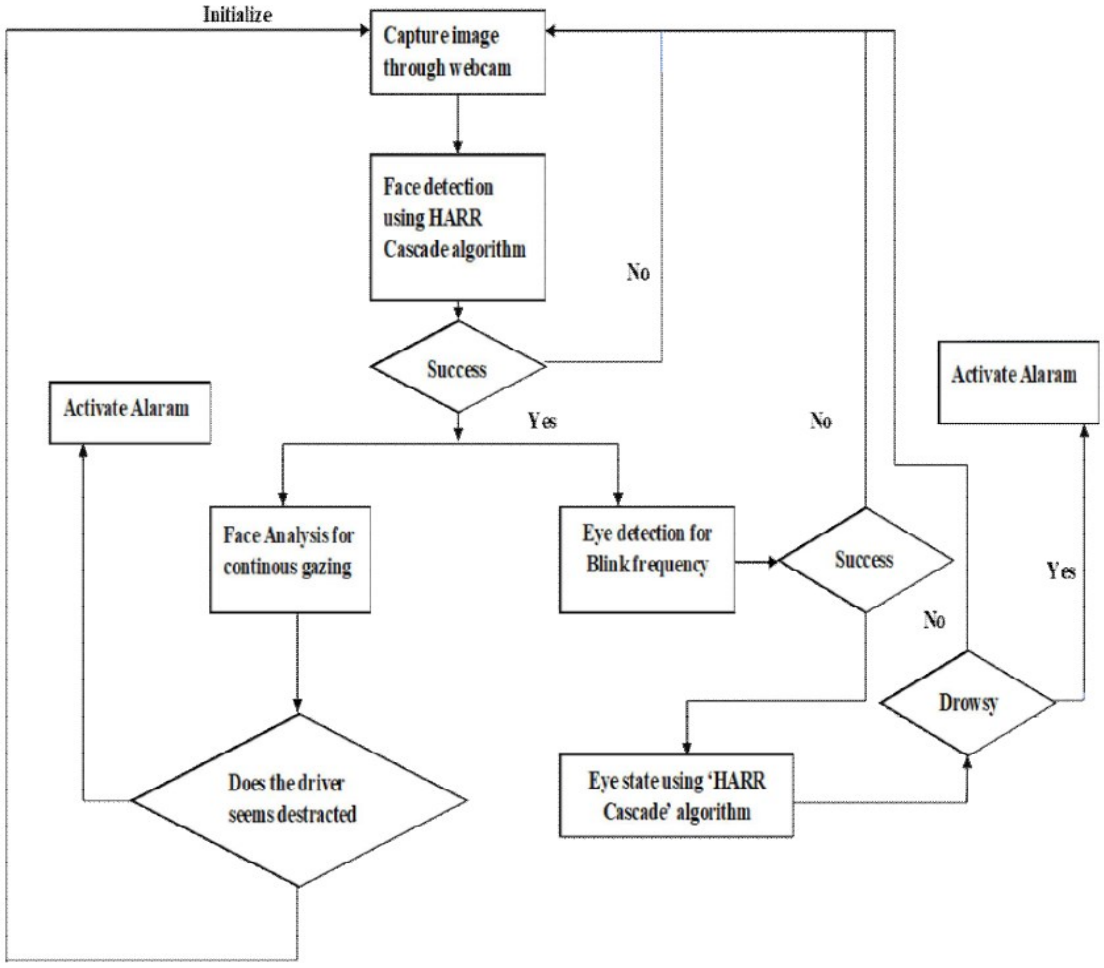


Figure 183: Architecture diagram.

12.8. Experiments & Results

12.8.1. Morning.

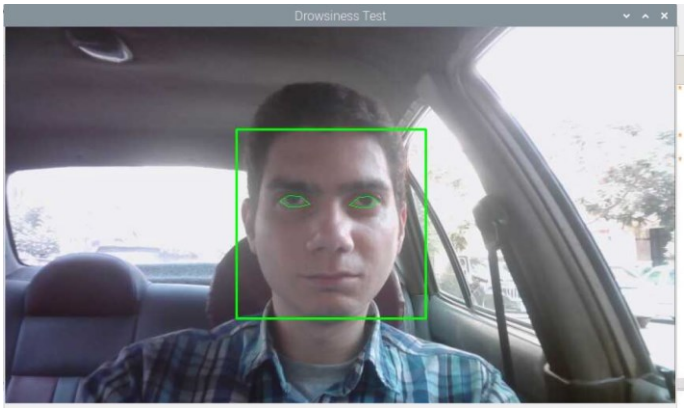


Figure 184 Normal state morning

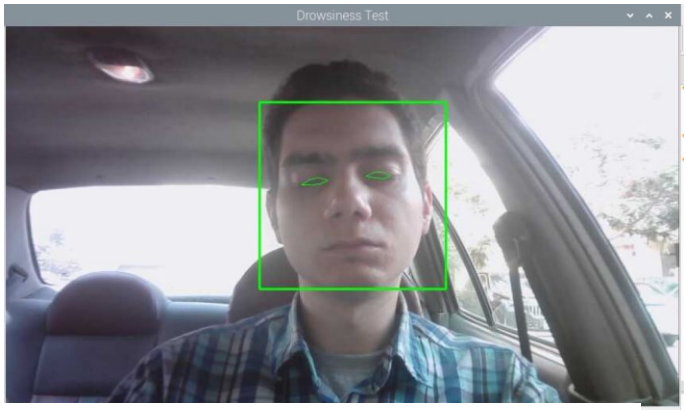


Figure 186 Closed eye state morning

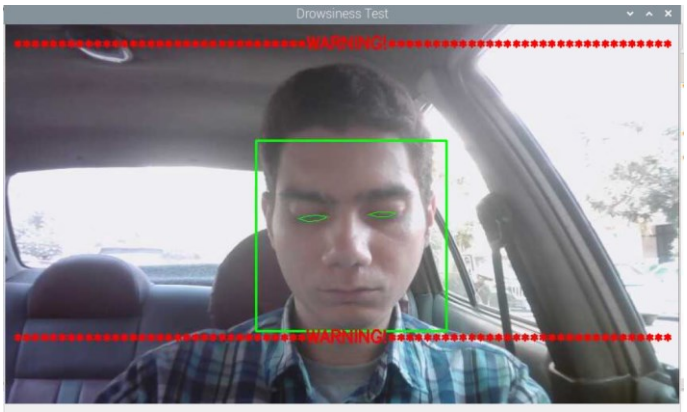


Figure 187 Warning state morning

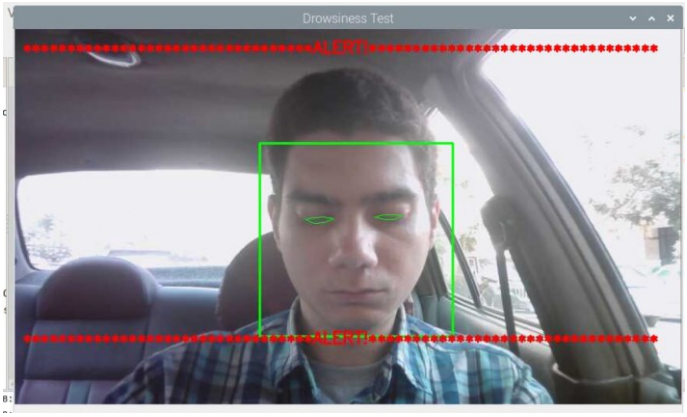


Figure 185 Alert state morning

The code successfully executed and provided results for drowsiness detection in the daylight. The implemented algorithms accurately detected and alerted for drowsiness based on the Eye Aspect Ratio (EAR) metric. The code properly differentiated between normal, warning, and full brake levels based on the duration of low EAR values. The morning test results demonstrated the effectiveness of the code in identifying drowsiness and initiating appropriate actions. The drowsiness detection system performed reliably and effectively during the morning scenario, highlighting its potential for real-world applications. Further evaluation and testing are recommended to ensure consistent performance across different time periods and scenarios.

12.8.2. Night.

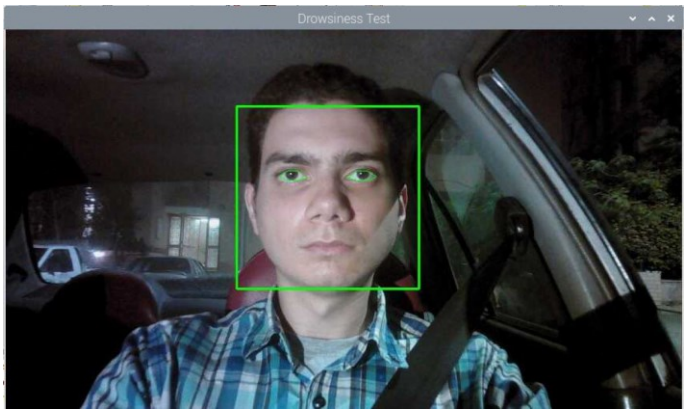


Figure 189 Normal state night

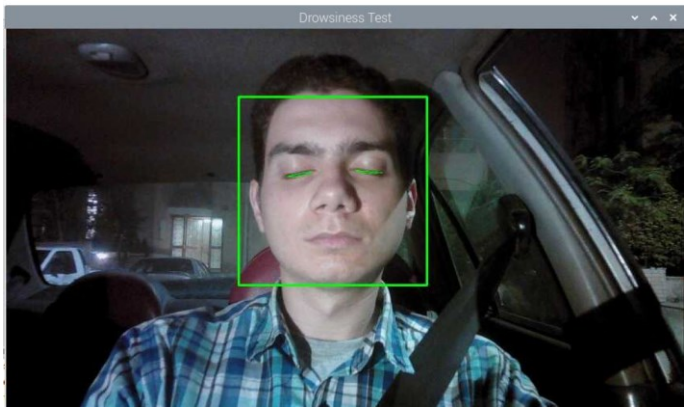


Figure 188 Closed eye state morning

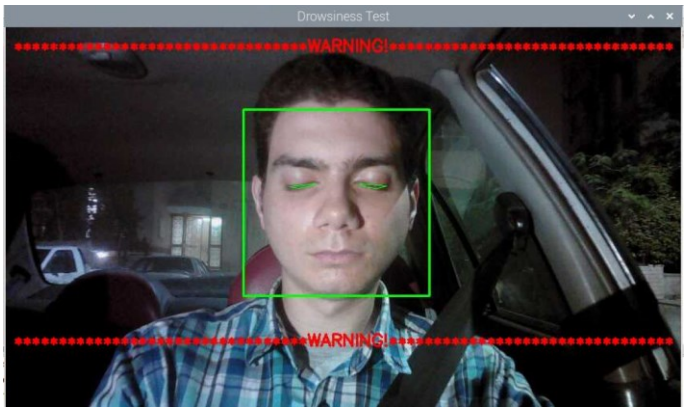


Figure 191 Warning state night

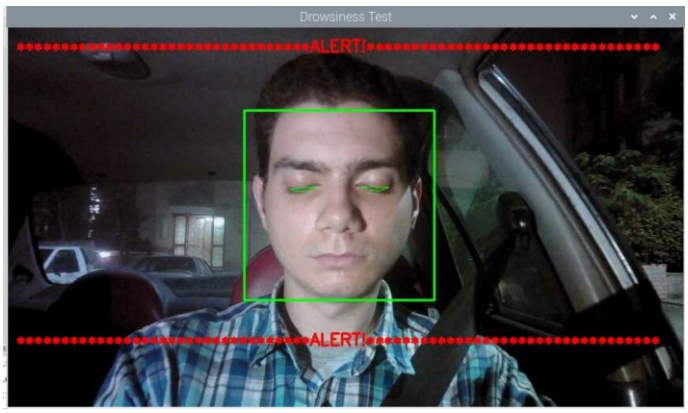


Figure 190 Alert state morning

Results The code successfully executed and provided results for drowsiness detection during nighttime conditions. The implemented algorithms accurately detected and alerted for drowsiness based on the Eye Aspect Ratio (EAR) metric, even in low light conditions. The code properly differentiated between normal, warning, and full brake levels based on the duration of low EAR values, ensuring appropriate responses to drowsiness. The nighttime test results demonstrated the effectiveness of the code in identifying drowsiness and initiating appropriate actions, highlighting its reliability in different lighting conditions. The drowsiness detection system performed consistently and effectively during the nighttime scenario, indicating its potential for real-world implementation in various settings. Further evaluation and testing are recommended to assess performance across different time periods, lighting conditions, and scenarios to ensure its robustness and accuracy.

12.8.3. low light conditions

Figure 192 Warning state low light

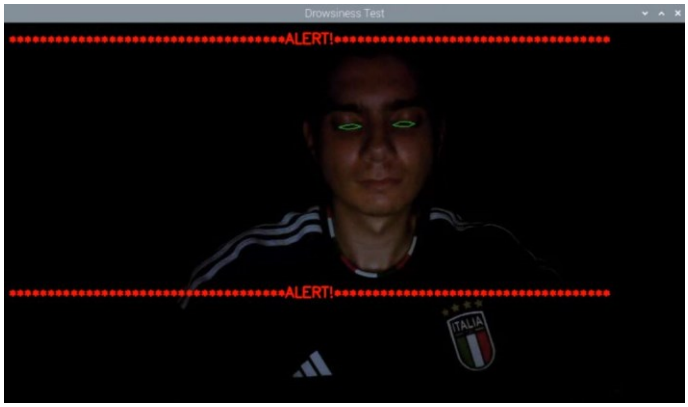


Figure 193 Alert state low light

Results The code successfully executed and provided results for drowsiness detection during low light intensity conditions. The implemented algorithms accurately detected and alerted for drowsiness based on the Eye Aspect Ratio (EAR) metric, even in low light conditions. The code properly differentiated between normal, warning, and full brake levels based on the duration of low EAR values, ensuring appropriate responses to drowsiness. The nighttime test results demonstrated the effectiveness of the code in identifying drowsiness and initiating appropriate actions, highlighting its reliability in different lighting conditions. The drowsiness detection system performed consistently and effectively during the nighttime scenario, indicating its potential for real-world implementation in various settings. Further evaluation and testing are recommended to assess performance across different time periods, lighting conditions, and scenarios to ensure its robustness and accuracy.

12.9. Observations

After evaluating the drowsiness detection system, the following observations can be made:

Accuracy: The system demonstrates a reliable level of accuracy in detecting drowsiness based on the Eye Aspect Ratio (EAR) metric. It effectively distinguishes between normal, warning, and full brake levels, providing appropriate responses to varying levels of drowsiness.

Real-Time Performance: The system operates in real-time, continuously monitoring the driver's eye movements and promptly alerting for drowsiness. The response time between detecting drowsiness and triggering the corresponding action is sufficiently fast, ensuring timely interventions.

Lighting Conditions: The system showcases robustness across different lighting conditions, including both daytime and nighttime scenarios. It effectively detects and alerts for drowsiness even in low light conditions, emphasizing its adaptability to varying environmental settings.

Hardware Integration: The integration of the Raspberry Pi and other components, such as the webcam and buzzer, facilitates a cohesive system. The use of the CAN bus for communicating with a connected vehicle system enables additional safety measures, such as initiating full brake actions when necessary.

Potential for Real-World Application: The overall performance of the system indicates its potential for real-world implementation in various contexts. It can contribute to enhancing driver safety by proactively detecting drowsiness and issuing appropriate alerts or interventions.

processing time: We have noticed that with reducing the width of the frame, the results become more accurate and faster than it was with a large frame, and this is because the number of pixels in the image decreases, so the time to process decreased which make the system faster.

Further Testing and Refinement: While the system demonstrates promising results, additional testing and refinement are recommended. Evaluating its performance across diverse scenarios, analyzing its robustness to various eye-related factors, and considering user feedback can contribute to further improving the system's effectiveness and reliability.

12.10. CONCLUSIONS

In conclusion, the drowsiness detection project presented a comprehensive system for real-time monitoring and detection of driver drowsiness. By leveraging computer vision techniques, facial landmarks, and the Eye Aspect Ratio (EAR), the system successfully identified signs of drowsiness based on the closed or partially closed state of the driver's eyes. The integration of a Raspberry Pi, webcam, and buzzer facilitated seamless hardware implementation, while the utilization of the CAN bus allowed for communication with a connected vehicle system to initiate appropriate actions. The created driver drowsiness system is able to quickly identify drowsy driving behaviors which was created based on the driver’s eye closure, is able to distinguish between regular eye blinking and drowsiness and can identify drowsiness while driving. The suggested technique can stop accidents brought on by drivers who are drowsy.

Chapter 13

Distraction Detection

13. Distraction Detection

13.1. Motivation:

Recently, the number of road accidents has been increased worldwide due to the distraction of the drivers. This rapid road crush often leads to injuries, loss of properties, even deaths of the people. Therefore, it is essential to monitor and analyze the driver's behavior during the driving time to detect the distraction and mitigate the number of road accident. To detect various kinds of behavior like- using cell phone, talking to others, eating or lack of concentration during driving.

13.2. Distraction

Distracted driving is anything that inhibits a driver from paying full attention to the task of driving, or inhibits them from being fully engaged to adequately respond to changes in the driving environment.

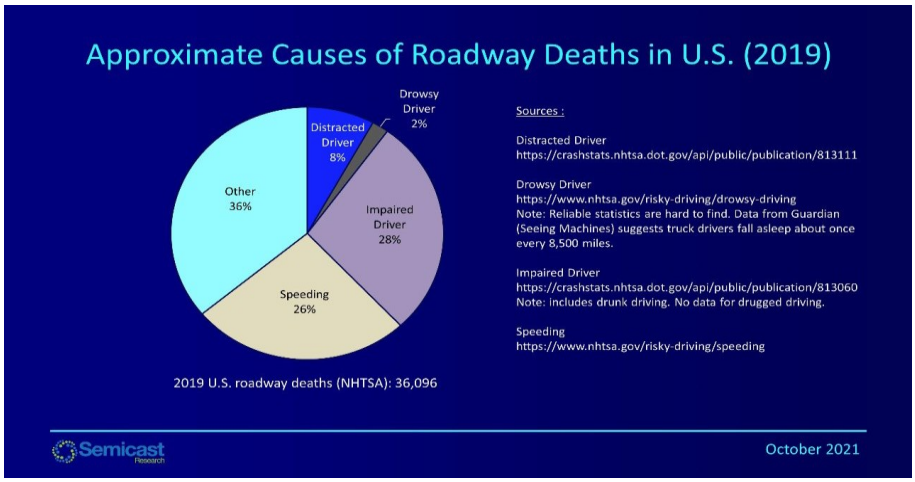


Figure 194 How distracted drivers cause more accidents than drowsy drivers

According to the National Highway Traffic Safety Administration, distractions typically fall into three categories:

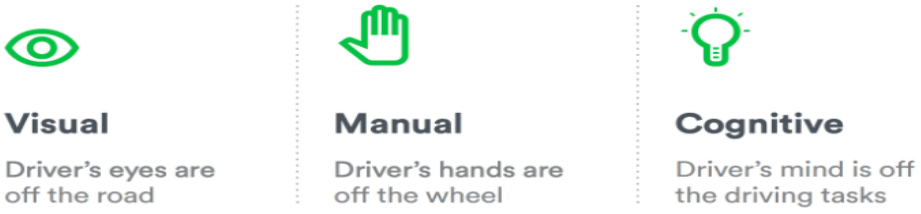


Figure 195: Distraction three main categories.

13.3. Problem Description:

According to WHO (World Health Organization, 2020), the number of deaths caused by road accidents is approximately 1.35 million each year and on average, 64 people die every day. Road accident is a major problem in many developing countries including Bangladesh. More than 5000 people were killed in road accidents across Bangladesh in 2019, an unprecedented increase from the previous year's projections. The number of deaths in road crashes increased to 5227, with 788 more people killed than in 2018.

A driver may be engaging in one or more distracted driving behaviors at any given time. The result is the same: heightened risk of a collision, injury, or even fatality. There's little argument that distracted driving is one of the fastest growing threats to safe driving today and cause of collisions in the United States. Nearly 95% of serious traffic collisions are due to human error, with over 70% of commercial fleet collisions involving distracted drivers. The National Safety Council reports on a typical day, more than 700 people are injured in distracted driving crashes. These statistics are true indicators of how safely drivers are using the roadways and why keeping commercial fleet drivers and bystanders safe on the road is becoming a bigger and more costly challenge for many fleet managers and safety leaders.

Distracted driving comes in a variety of forms. These behaviors include instances when a driver is looking down or away from the road ahead for a period of time long enough to lose situational awareness of the forward driving scene like:

- Talking on a cell phone.
- Texting.
- Eating.
- Smoking.
- Using a tablet.
- Reading paperwork.
- Programming an in-vehicle infotainment system.

13.4. Problem Solution:

To detect driver distraction, most video telematics and dashcam solutions today require driver video to be uploaded to the cloud for analysis (i.e. human review) before any distraction determination is made. There are significant shortcomings to this approach:

1. The time lag, or latency, resulting from data transmission from the vehicle to the cloud and back again prevents real-time alerting. This means the driver doesn't get a chance to act in time to prevent the incident, and worse, the supervisor knows something has happened even before the driver does (since many systems on the market don't even let the driver know data was captured).

2. Drivers fear in-vehicle video will be used against them, and will find ways to block the system from working as intended. When this occurs, they're missing out on a chance to be exonerated from the collisions that are not their fault, as well as a chance to learn. This would be possible if the driver was notified of the risk first, in real-time, while they still had an opportunity to change their behavior or avoid the risk.

Reducing distracted driving and fleet collisions requires more than safety policies, traditional driver training, and physics-based ABS systems, and that's can be made by detecting the distraction of the driver and rapidly alert the driver of distraction until the driver focuses again on the road.

13.5. Description

Gaze tracking is the process of measuring the point of gaze. Gaze tracking technology is used in research on the human visual system, in psychology, in psycholinguistics, marketing, product design, and countless other mediums.

This can be done by the dlib, OpenCV libraries found in python.

Dlib is a general-purpose cross-platform software library written principally for C++, but a number of its tools can be used in python applications. This software library is responsible for the development of thousands of machine learning models. Dlib's pre-trained model for facial detection and 68 facial landmarks is used in this project when defining a face object.

Our approach in this part is gaze detection, we first specified the regions of eyes from the facial landmarks that we have used in the drowsiness detection part.

Figure 196: Eyes region localization.

We will split the detection into 3 looking positions: left, center and right.

Figure 197: Dividing the position of the iris into 3 positions.

By looking at the image above, it's that the sclera (white part of the eye) fills the right part of the eye when the eye is looking at the left, the opposite happens when it's looking to the right and when it's looking to the center the white is well balanced between left and right.

The idea is to split the eye in two parts and to find out in which of the two parts there is more sclera visible.

Technically to detect the sclera we convert the eye into grayscale, we find a threshold and we count the white pixels.

Figure 198: Defining a threshold to detect the iris of the eye where it's looking at.

We divide the white pixels of the left part and those of the right part and we get the gaze ratio. Normally both the eyes look in the same direction, so if we correctly detect the gaze of a single eye, we detect the gaze of both eyes.

By creating a function called get_gaze_ratio(eye_points, facial_landmarks), which first get the x and y coordinates for each point of the eye landmarks, then the facial landmarks of the whole face to locate the location of the eyes, we will do that for left eye and right eye.

Now, we will calculate the gaze ratio as follows:

```
# Gaze detection
gaze_ratio_left_eye = get_gaze_ratio([36, 37, 38, 39, 40, 41], landmarks)
gaze_ratio_right_eye = get_gaze_ratio([42, 43, 44, 45, 46, 47], landmarks)
gaze_ratio = (gaze_ratio_right_eye + gaze_ratio_left_eye) / 2
```

Figure 199: Gaze ratio calculations.

And once we have the gaze ratio we can display on the screen the result.
We found out that, the if the gaze ratio is smaller than 1 when looking to the right side and greater than 1.7 when the eyes are looking to the left side.

```
if gaze_ratio <= 1:
    cv2.putText(frame, "RIGHT", (50, 100), font, 2, (0, 0, 255), 3)
    new_frame[:] = (0, 0, 255)
elif 1 < gaze_ratio < 1.7:
    cv2.putText(frame, "CENTER", (50, 100), font, 2, (0, 0, 255), 3)
else:
    new_frame[:] = (255, 0, 0)
    cv2.putText(frame, "LEFT", (50, 100), font, 2, (0, 0, 255), 3)
```

Figure 200: Detecting where the eyes are looking at.

13.6. Detecting the driver's distraction

Now it's time to apply our algorithm to detect the distraction of the driver. When the gaze ratio is between 1 and 1.7; then the eyes are centered.

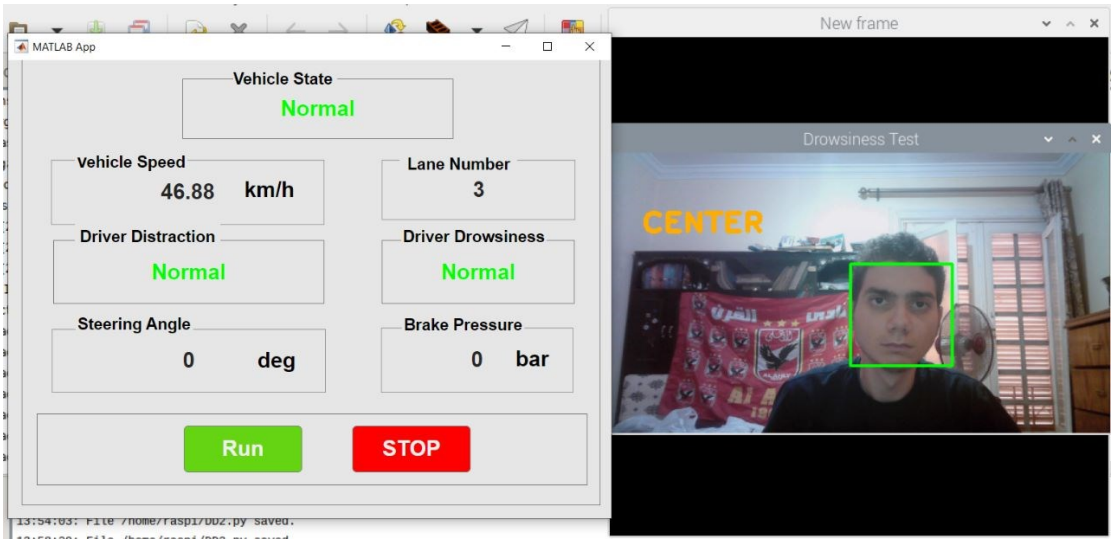


Figure 201: Normal status when the driver's eyes are centered

When the ratio of the gaze is less than or equal 1 then the driver is looking at the right, else if the driver's gaze ratio is more than 1.7 then the driver is looking at the left.

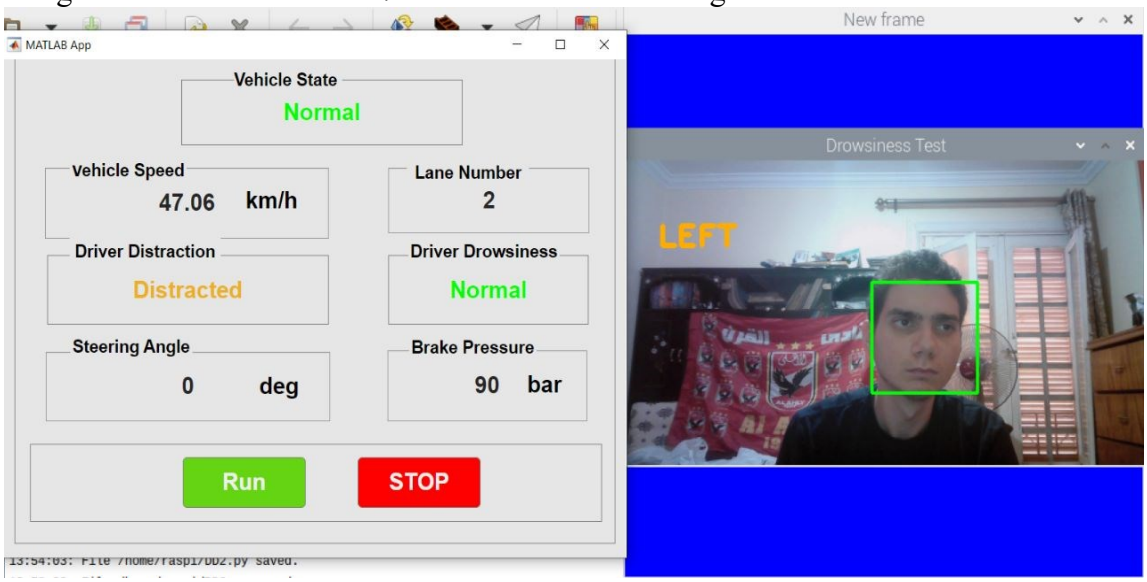


Figure 202: Detecting the eyes to be looking at the left when the gaze ratio was more than 1.7.

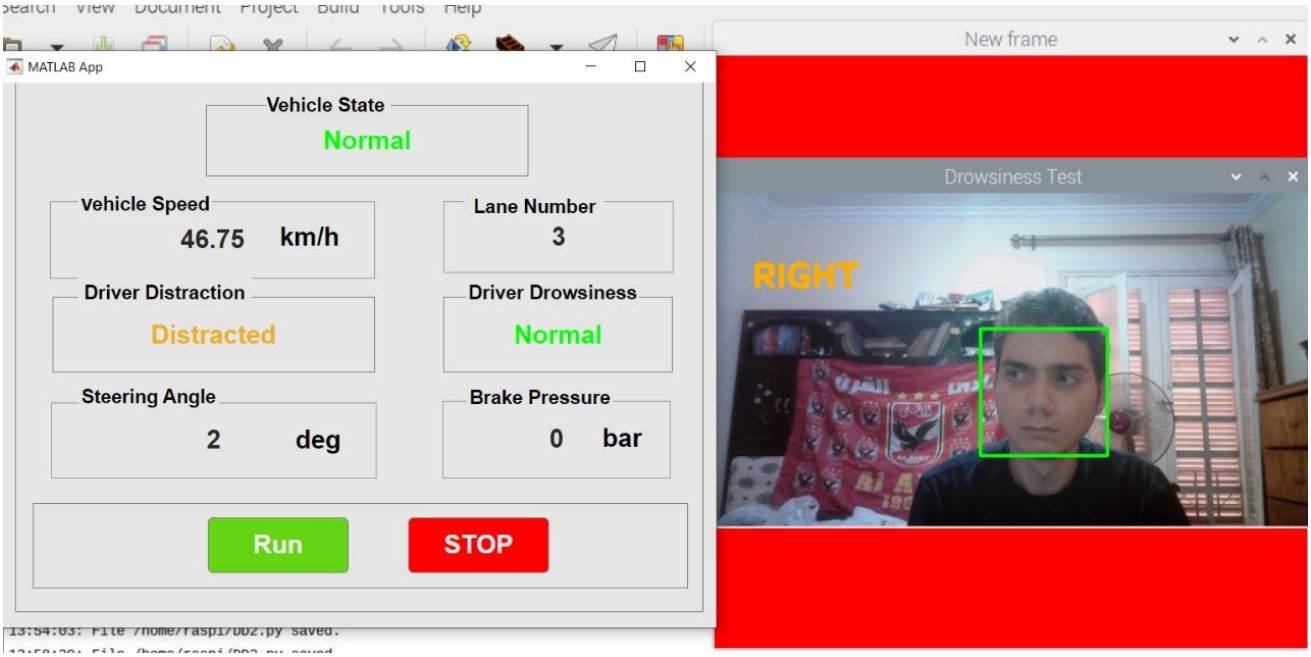


Figure 203: Detecting the driver to be looking at the right when the gaze ratio is less than 1.

Chapter 14
ABS Android Application

14. ABS Application

14.1. App overview:

Figure 204 App overview

The main aim of our application is to save the driver in cases of fainting or an accident happened, a notification sends to Hospitals, then the ready one to save him take the mission and sends an ambulance to the location of the car.

Our application consists of two types of users:

- Driver: we ask him to fill some information like phone number, emergency phone number, and medicine he takes.
- Hospital: fill some information like name, phone, ...

communication techniques:

We Send notification to Hospital Using firebase FCM.

Note: Application support dark mood and light mood.

14.2. Technolgy used

14.2.1. Firebase

Firebase is a comprehensive mobile and web development platform provided by Google. It offers a range of services and tools that simplify the process of building and managing applications. With Firebase, developers can easily handle tasks like data storage, user authentication, real-time database synchronization, and push notifications. It provides a scalable and flexible backend infrastructure, allowing developers to focus on creating exceptional user experiences rather than worrying about server management. Firebase is widely used in the industry for its ease of integration, robust features, and ability to accelerate application development.

14.2.1. Flutter

Flutter is an open-source UI framework developed by Google for creating cross-platform applications. It enables developers to build high-performance, visually appealing apps for iOS, Android, web, and desktop from a single codebase. With Flutter's hot-reload feature, developers can instantly see changes in real-time, speeding up the development process. Its rich widget library and customizable UI components allow for seamless app design and smooth user experiences. Flutter has gained popularity due to its ability to deliver native-like performance and its extensive community support, making it an ideal choice for building feature-rich, cross-platform applications.

14.3. Project Flow

step 1: when our drowsiness and distraction algorithm detect the driver is asleep, it sends this status to discovery board to take action and stop the car, then raspberry read some data from Simulink like

- position of the car (x,y) coordinates.
- car Id.

step 2: send this data to firebase server and using FCM (firebase cloud messaging) we send notification to hospital.

```
db = firestore.client()
doc_ref = db.collection('emergency').document()
uid = doc_ref.id
#Push data to a specific collection
data = {'x': f"{x}",
        'y': f"{y}",
        'carId': f"{carId}",
        'uid': f"{uid}",
        'status': False,
        'user': {'uid': "NULL"},
        'hospital': {'uid': "NULL"},

        }
doc_ref.set(data)

message = messaging.Message(
    notification=messaging.Notification(
        title='New Accident',
        body=f"Driver with CarID '{carId}' make Accident , Try to Help",
    ),
    topic='all'
)
```

Figure 205: send data from raspberry to firebase

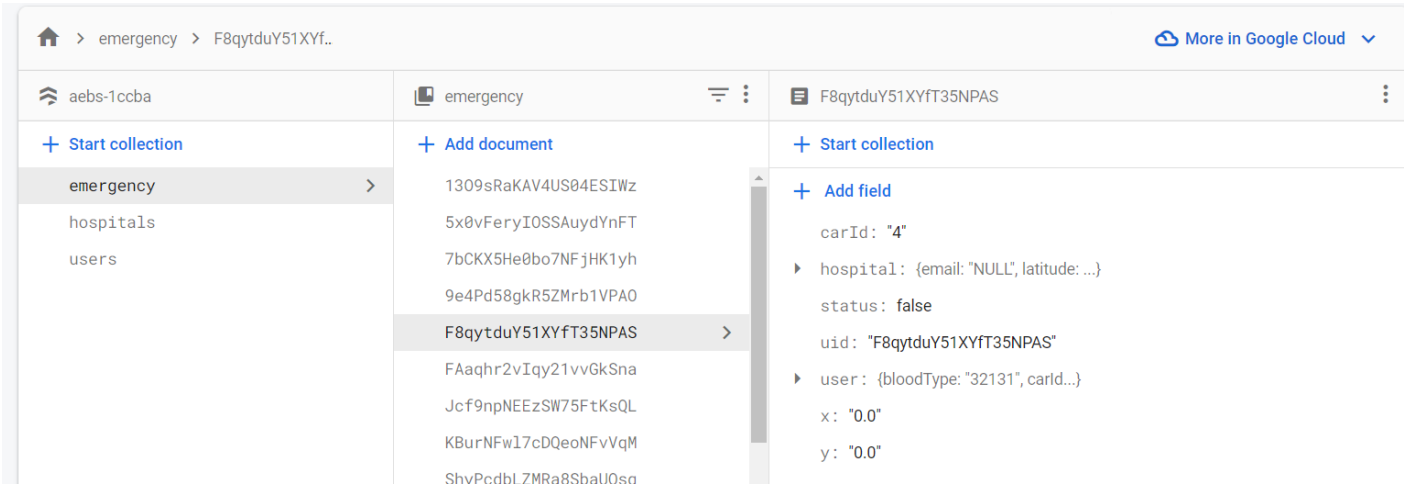


Figure 206: emergency request data field in data base

Step 3: notification send to hospital and the available one can accept the request to help the driver, then the request disappear from other hospital's account.

Notes:

- information of user we show in request page, we get it depend on carID, which field was required from user during registration process in application.
- Each 'emergency collection' in data base has variable called hospital, which is a Jason contain information of hospital which is initially NULL. And after certain hospital accept the request, this variable is updated with information of this hospital and will appear in history page of hospital.
- We send notification from raspberry to topic "all", in application side we make subscription in this topic after login process from hospital account.

14.4. Applcation pages details

14.4.1. Login Page

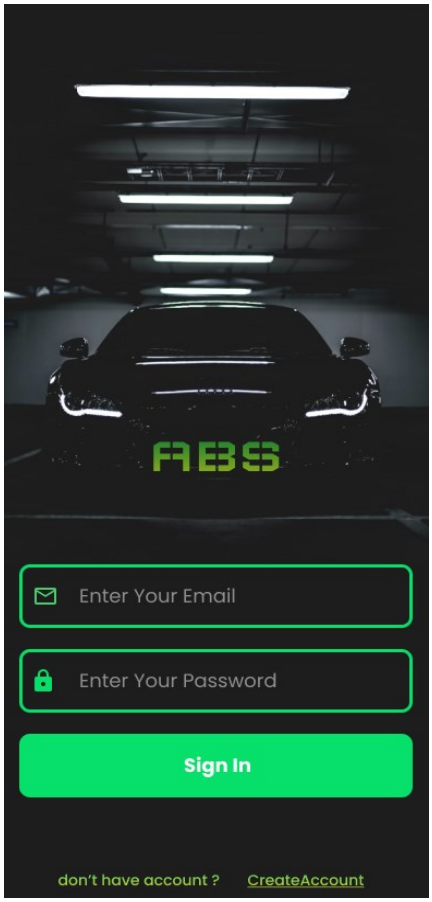


Figure 207 :Login page

This page represent the login interface system for hospital:

- new users, must create a new account.
- registered users, must enter their mail and password correctly to login to their profiles.

14.5. Create New Account:

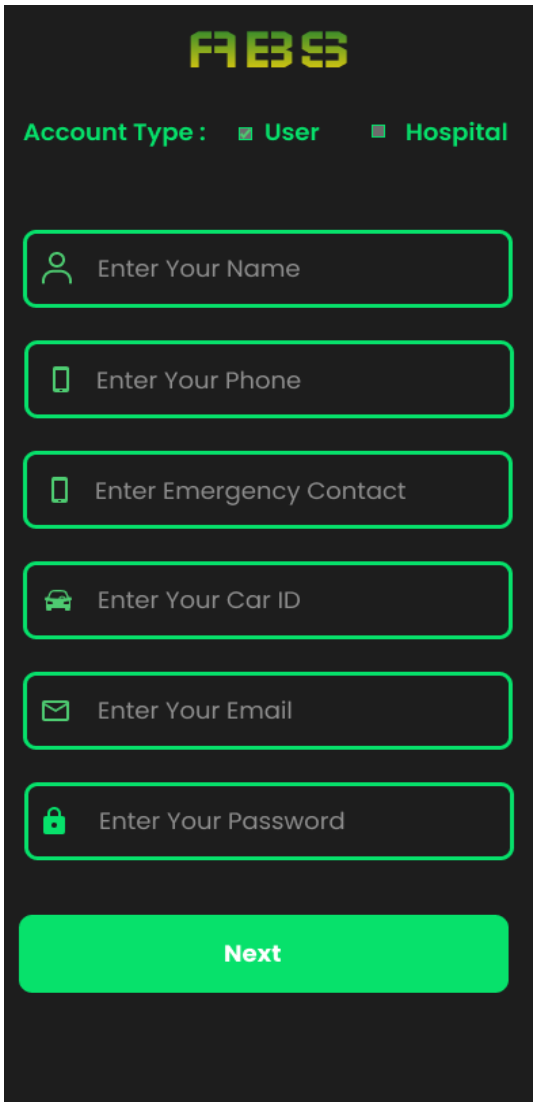


Figure 209 :Create New account for driver

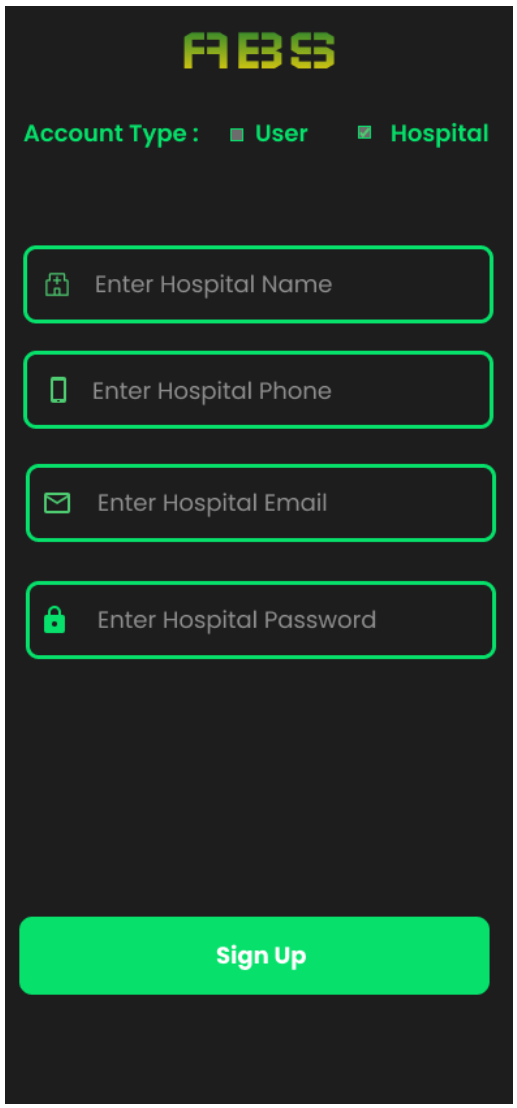


Figure 209 :Create New account for Hospital

These pages for create a new accounts for both driver and Hospital:

- For the driver, there are some important fields that need to be filled, such as: Driver Name, Phone Number, Car ID, Email, and Mail Password.
- For Hospital System, there are some important fields that need to be filled such as: Hospital Name, Phone Number, Email, Mail Password.

14.6. Complement of User Information related to User Sign Up:

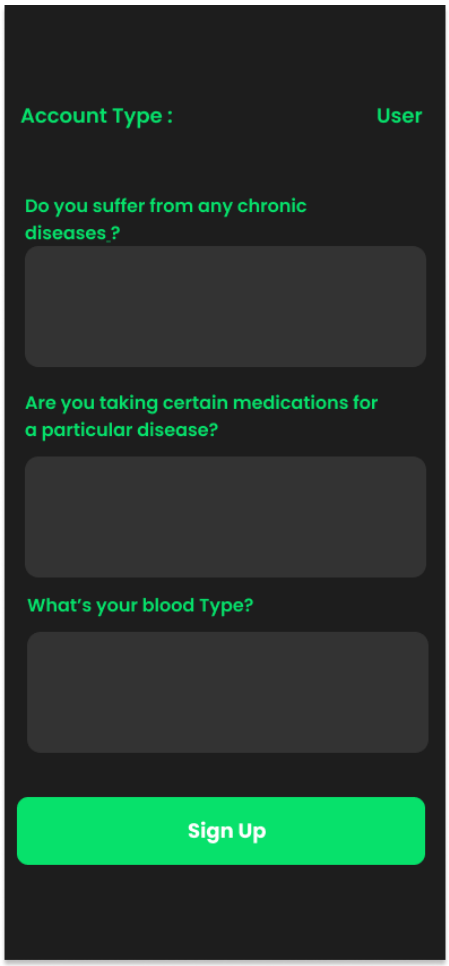


Figure 210 :Complement of User Information related to User Sign Up

In case of emergencies, it is essential for the hospital to be aware of the driver's medical condition. The following questions provide the hospital with necessary information regarding the driver:

- Does the driver suffer from any chronic diseases?
- Does the driver take any specific medications for a particular condition?
- What is the driver's blood type?

These details will enable the hospital to handle difficult situations more effectively

14.7. Hospital related Data

14.7.1. Hospital Profile

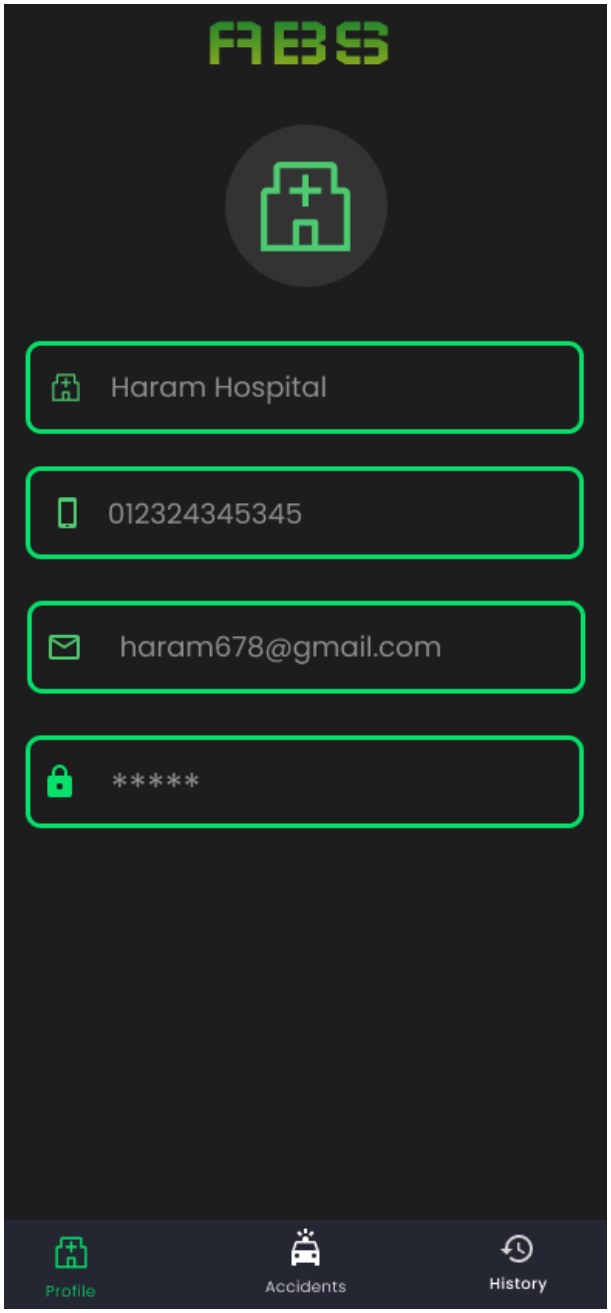


Figure 211 :Hospital Profile

14.7.2. Hospital Emergency Accidents:

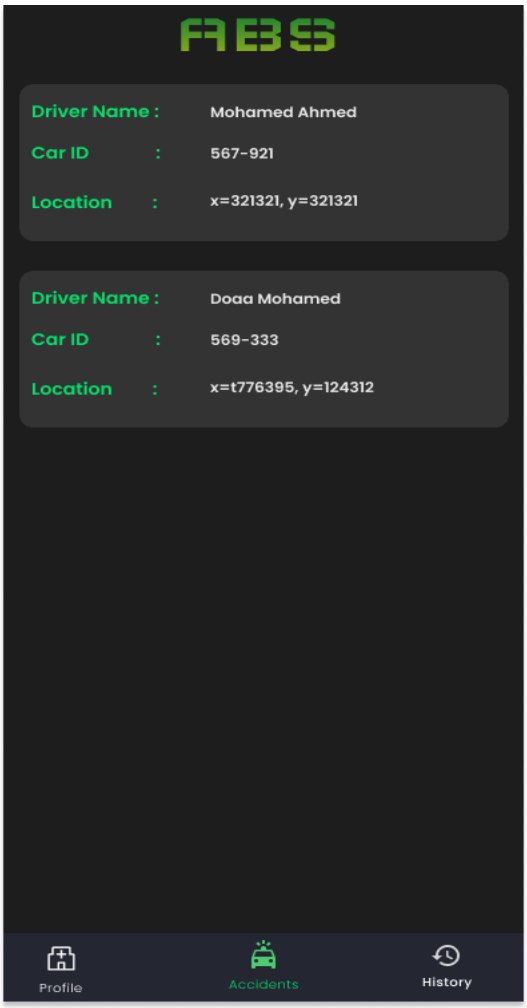


Figure 212 :Hospital Emergency Accidents

This page serves as a platform to receive notifications from vehicles alerting the driver about potential dangers. It displays cards belonging to all users who have requested automatic emergency assistance from nearby hospitals.

Each card contains user information that was registered during the account creation process. This information is crucial for hospitals to locate the specific vehicle and provide efficient medical assistance

14.7.2.1. Hospital Emergency Accidents Card details

This information broadcasts for all hospitals and first one accepts this card, it will remove from accidents to history page .

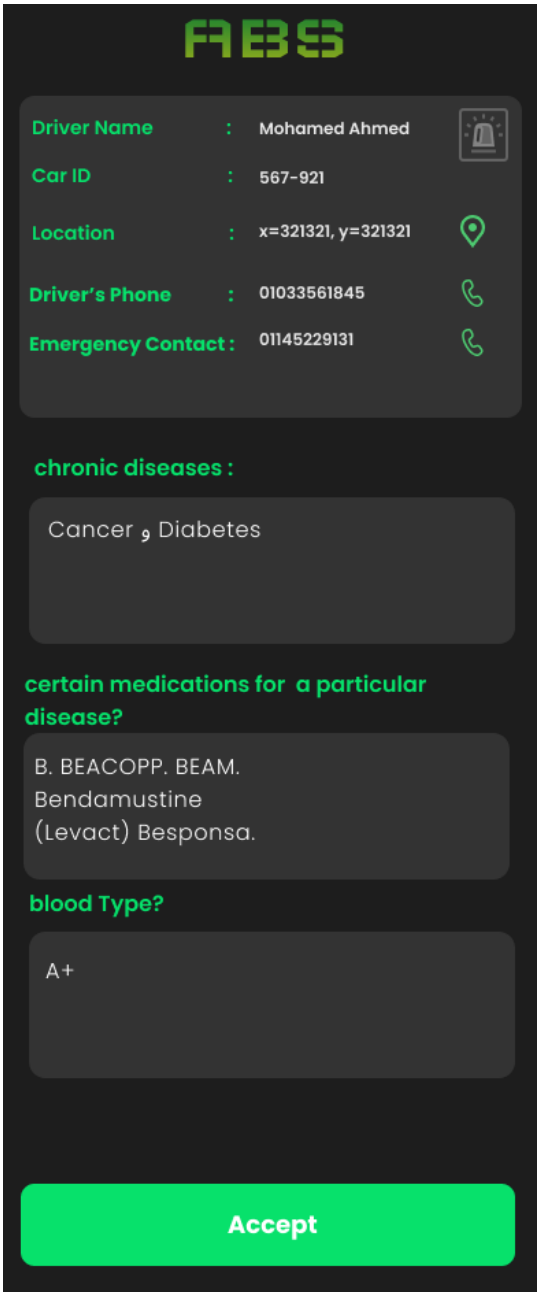


Figure 213 :Hospital Emergency Accidents Card details

References

[1]. Market leading RTOS (real time operating system) for embedded systems with internet of things extensions (2023) FreeRTOS. Available at: https://www.freertos.org/ (Accessed: 10 July 2023).

[2]. How do I update a GUI designed in app designer with data from a run... (no date) MATLAB Answers - MATLAB Central. Available at: https://www.mathworks.com/matlabcentral/answers/446302-how-do-i-update-a-gui-designed-in-app-designer-with-data-from-a-running-simulink-model (Accessed: 10 July 2023).

[3]. (No date a) Design & verification of serial peripheral interface (SPI) protocol. Available at: https://www.ijrte.org/wp-content/uploads/papers/v8i6/F7356038620.pdf (Accessed: 10 July 2023).

[4]. (No date b) Understanding microchip S can module bit timing. Available at: https://ww1.microchip.com/downloads/en/AppNotes/00754.pdf (Accessed: 10 July 2023).

[5]. Admin (2023) How to use can protocol in STM32 " controllerstech, ControllersTech. Available at: https://controllerstech.com/can-protocol-in-stm32/ (Accessed: 10 July 2023).

[6]. Can bit segments? (2018) Solved: Can bit segments? - STMicroelectronics Community. Available at: https://community.st.com/t5/stm32cubemx-mcu/can-bit-segments/m-p/357097 (Accessed: 10 July 2023).

[7]. Is it possible to start and stop a simulink simulation from inside ... (no date) MATLAB Answers - MATLAB Central. Available at: https://www.mathworks.com/matlabcentral/answers/97557-is-it-possible-to-start-and-stop-a-simulink-simulation-from-inside-a-gui (Accessed: 10 July 2023).

[8]. Piembsystech (2020) Can protocol bit time calculation, PiEmbSysTech. Available at: https://piembsystech.com/can-bit-time-calculation/ (Accessed: 10 July 2023).

[9]. Raspberry Pi (no date) Buy A raspberry pi 3 model B, Raspberry Pi. Available at: https://www.raspberrypi.com/products/raspberry-pi-3-model-b/ (Accessed: 10 July 2023).

[10]. Simin (no date) Simulate Simulink model - MATLAB sim - MathWorks India. Available at: https://in.mathworks.com/help/simulink/slref/sim.html (Accessed: 10 July 2023).

[11]. STM32F429/439 (no date) STMicroelectronics. Available at: https://www.st.com/en/microcontrollers-microprocessors/stm32f429-439.html (Accessed: 10 July 2023).

[12].Wang, C.-S., Liu, D.-Y. and Hsu, K.-S. (2018) Simulation and application of cooperative driving sense systems using prescan software - Microsystem Technologies, SpringerLink. Available at: https://link.springer.com/article/10.1007/s00542-018-4164-z?utm_source=getftr&utm_medium=getftr&utm_campaign=getftr_pilot (Accessed: 10 July 2023).

[13]. Youness (2020) How to connect Raspberry Pi to can bus, Hackster.io. Available at: https://www.hackster.io/youness/how-to-connect-raspberry-pi-to-can-bus-b60235 (Accessed: 10 July 2023).

[14]. 1080p USB webcam (no date) Manhattan Products. Available at: https://manhattanproducts.eu/products/manhattan-en-1080p-usb-webcam-462006 (Accessed: 10 July 2023).

[15]. Tolgakarakurt (no date) Tolgakarakurt/canbus-MCP2515-raspi: MCP2515 canbus module installation guide on RaspberryPi, GitHub. Available at: https://github.com/tolgakarakurt/CANBus-MCP2515-Raspi (Accessed: 10 July 2023).

[16]. Raspberry at can bus via levelshifter (no date) Raspberry at CAN bus via levelshifter. Available at: https://fritzing.org/projects/raspberry-at-i2c-via-levelshifter (Accessed: 10 July 2023).

[17].Lee, J., Kim, G. and Kim, B. (2019) Study on the improvement of a collision avoidance system for curves, MDPI. Available at: https://www.mdpi.com/2076-3417/9/24/5380/htm (Accessed: 10 July 2023).

[18].Everything you need to know about semaphores and mutexes (2023) Beningo Embedded Group. Available at: https://www.beningo.com/everything-you-need-to-know-about-semaphores-and-mutexes/# (Accessed: 10 July 2023).

[19]. (No date) Dart packages. Available at: https://pub.dev/ (Accessed: 12 July 2023).

[20]. Build and release an Android app (no date) Flutter. Available at: https://docs.flutter.dev/deployment/android (Accessed: 12 July 2023).

[21]. Flutterfire overview: Flutterfire (no date) FlutterFire Blog RSS. Available at: https://firebase.flutter.dev/docs/overview/ (Accessed: 12 July 2023).

[22]. (2008). Automated Emergency Brake System: Technical requirements, costs and benefits. (1st ed., pp. 1-117). Published Project Report.

[23]. Drowsiness and distraction detection:

Pooja D.C., Sara Aziz, Shakuntala Koujalagi, Shilpa B. H., Mr. Vasanth Kumar N.T. (2022). Driver Drowsiness Detection Using OpenCV and Raspberry Pi. International Journal for Research in Applied Science & Engineering Technology (IJRASET), Volume 10 (Issue VII). 1-7. https://doi.org/10.22214/ijraset.2022.45288.

[24]. Face detection using Haar Cascades (no date) OpenCV. Available at: https://docs.opencv.org/4.0.0/d7/d8b/tutorial_py_face_detection.html (Accessed: 1 July 2023).

[25]. Team, G.L. (2023) Face detection using Viola Jones algorithm, Great Learning Blog: Free Resources what Matters to shape your Career! Available at: https://www.mygreatlearning.com/blog/viola-jones-algorithm/ (Accessed: 1 July 2023).

[26]. there, S.C. (2019) Eye gaze detection 2 – gaze controlled keyboard with python and opencv p.4, Pysource. Available at: https://pysource.com/2019/01/17/eye-gaze-detection-2-gaze-controlled-keyboard-with-python-and-opencv-p-4/ (Accessed: 1 July 2023).