Information Technology Institute

Embedded Systems 4-Months Graduation Project

SmartDrive

Submitted By:

Ahmed Abdelnaser shaker Aboraya
Ahmed Bahaa Taha Mahdy Nasr
Khaled Ibrahim Abdulaziz Hamed
Mohamed Ahmed Abd El Aliem Gwaily
Muhammad Sameer Awad Alkahwagy

Supervised By:

Eng. Youssef Nofal

Eng. Nour Hasan

Table of Contents

1 Preface

It is with great pleasure that we present to you our graduation project, SmartDrive.
After four months of hard work, dedication, and collaboration, we are thrilled to share the innovative ideas and solutions that we have developed during our time at the Information Technology Institute - ITI.

This project is the result of our passion for embedded systems and our desire to make a meaningful impact on the world. Through SmartDrive, we have explored the latest technologies used in advanced driving assistance systems and gained a comprehensive understanding of the challenges and opportunities that exist in this field.

We believe that our project has the potential to make a positive impact on the automotive industry, and we are excited to share our findings with you. We hope that this book will serve as a valuable resource for anyone who is interested in embedded systems and advanced driving assistance systems.

We would like to express our deepest gratitude to our supervisors and mentors, who have provided us with invaluable guidance and support throughout the course of this project. We would also like to thank our families, friends, and loved ones, who have supported us emotionally and provided us with the motivation and encouragement to pursue our goals.

Finally, we would like to thank you, our readers, for taking the time to read our graduation project book. We hope that our work will inspire you and that you will join us in our quest to make the world a better place.

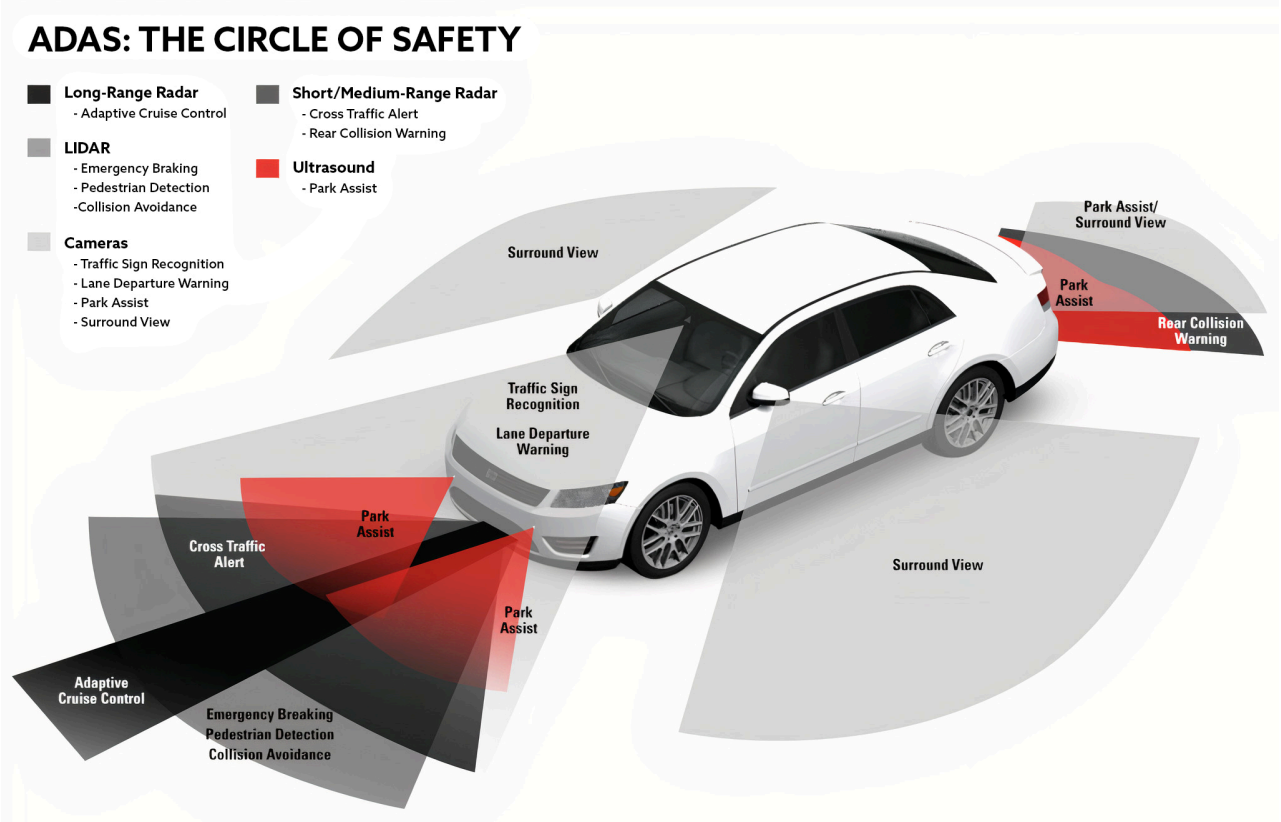
2 Introduction

Road safety is a crucial issue that impacts millions of people worldwide. Despite the significant advancements in automotive technology and road infrastructure, road accidents continue to be a major concern, resulting in millions of fatalities and injuries each year. These accidents are often caused by human error, such as speeding, distracted driving, and driver fatigue, as well as other contributing factors such as poor road conditions, inadequate traffic control measures, and vehicle malfunctions. Addressing these issues requires a comprehensive approach, including education, enforcement of traffic laws, and the development and implementation of advanced safety technologies. Technologies such as Advanced Driver Assistance Systems (ADAS) have proven to be effective in improving road safety by providing drivers with real-time information and assistance to avoid accidents. By working together to improve road safety, we can reduce the number of accidents and make our roads safer for everyone.

1 Fig.1.1

Advanced Driver Assistance Systems (ADAS) have become increasingly important in the automotive industry, providing drivers with increased safety, comfort, and convenience on the road. However, despite the significant advancements in ADAS technology, road accidents remain a major problem, causing millions of deaths and injuries each year.

One of the main causes of road accidents is driver error, which can be attributed to a variety of factors, including fatigue, distractions, and lack of attention. ADAS systems have the potential to significantly reduce the risk of accidents by providing drivers with real-time information and assistance, allowing them to make better decisions on the road. However, current ADAS systems often lack the reliability, accuracy, and user-friendliness needed to make a significant impact on road safety.



2 Fig.1.2

Our graduation project aims to address this problem by developing and implementing a comprehensive ADAS system that incorporates several key features, including Adaptive Cruise Control (ACC), Automatic Emergency Braking (AEB), Lane Keeping Assist System (LKAS), and Lane Departure Warning (LDW). Our system will be designed to be reliable, effective, and user-friendly, providing drivers with real-time information and assistance to reduce the risk of accidents caused by driver error.

Our project requires a deep understanding of the principles of automotive engineering, software development, and user interface design. We will use our knowledge of these fields to design and implement a system that is reliable, effective, and user-friendly. The successful implementation of our project has the potential to contribute to the advancement of ADAS technology, making driving safer and more efficient for everyone on the road. We hope that our project will inspire further research and development in this exciting field, and we look forward to sharing our results with the academic and automotive communities.

3 Background

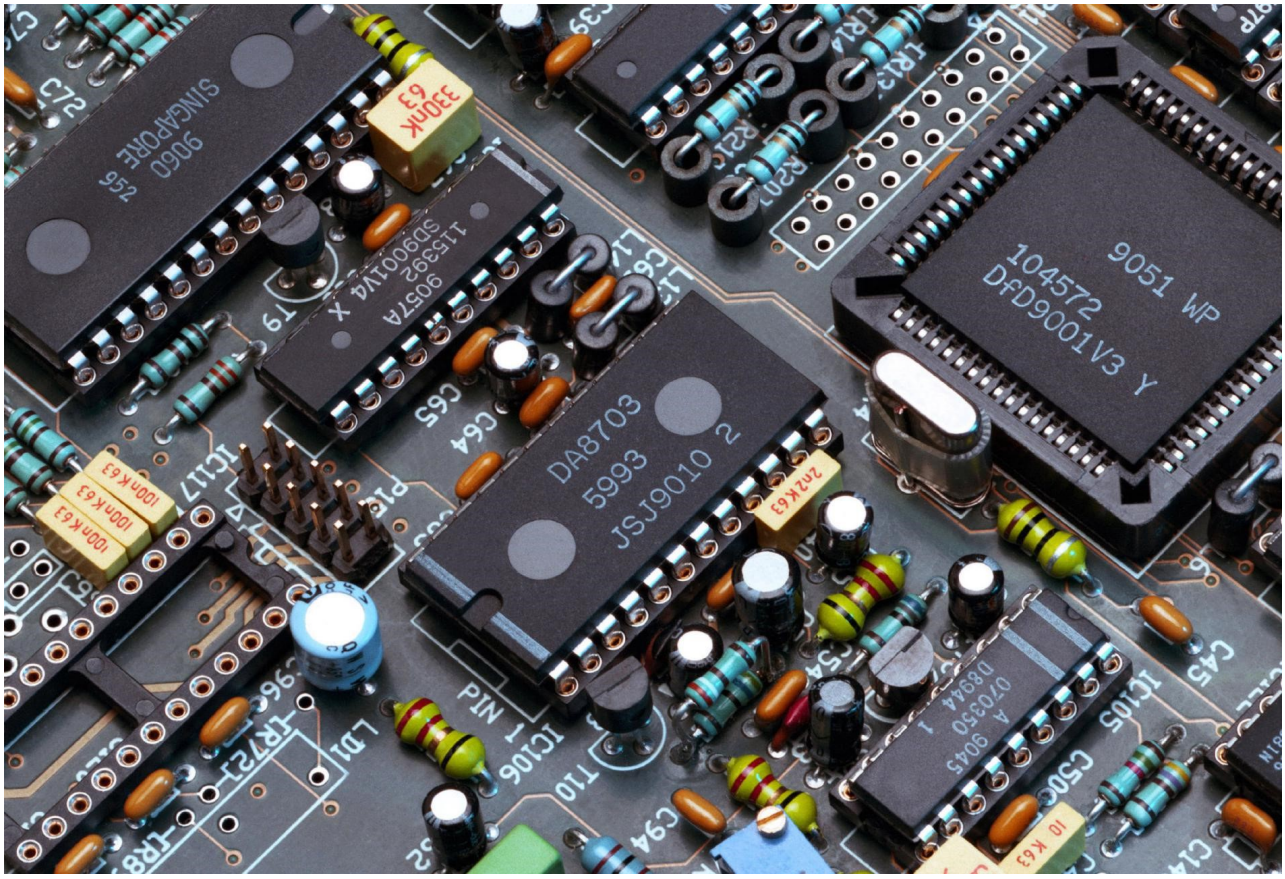
3.1 History of Embedded Systems

Embedded systems have been around since the earliest days of computing. In the 1960s and 1970s, embedded systems were primarily used in industrial control applications, such as process control and factory automation. These early embedded systems were typically built using discrete components and were designed to perform specific tasks.

In the 1980s, the development of microprocessors and microcontrollers led to a significant increase in the capabilities of embedded systems. These devices were smaller, more powerful, and more affordable than their predecessors, making them suitable for a wide range of applications.

In the 1990s and 2000s, the widespread adoption of the Internet and the development of wireless communication technologies led to a new era of embedded systems. Embedded systems were now being used in a wide range of applications, including consumer electronics, medical devices, and automotive systems.

Today, embedded systems are an essential part of our daily lives, powering everything from smartphones and smart homes to cars and airplanes. As technology continues to advance, we can expect to see even more sophisticated embedded systems that will revolutionize the way we live, work, and interact with the world around us.



3 Fig.2.1

3.2 Important Concepts

Embedded systems are computer systems that are designed to perform a specific function within a larger system. They are typically small, low-powered devices that are optimized for performance and efficiency. Here are some key concepts related to embedded systems:

1. Real-time Systems: Real-time systems are embedded systems that must respond to external events within a specific time frame. These systems are often used in safety-critical applications, such as in automotive or medical devices, where failure to respond quickly could result in serious consequences.
2. Microcontrollers: Microcontrollers are small, low-powered computer chips that are used to control the behavior of embedded systems. They typically include a processor, memory, and input/output ports, and are often programmed in C or assembly language.
3. Sensors: Sensors are devices that detect changes in the environment and provide input to the embedded system. Common types of sensors used in embedded systems include temperature sensors, pressure sensors, and motion sensors.
4. Actuators: Actuators are devices that convert electrical signals from the embedded system into physical actions. Common types of actuators used in embedded systems include motors, solenoids, and relays.
5. Communication Protocols: Embedded systems often need to communicate with other devices or systems. Communication protocols, such as I2C, SPI, and UART, are used to establish a standardized method of communication between devices.
6. Power Management: Embedded systems are often designed to operate on battery power or other limited power sources. Effective power management techniques are critical for ensuring that the system operates efficiently and reliably.

3.3 Embedded Systems in Automotive Industry

Nowadays Embedded systems are playing an increasingly important role in the automotive industry, providing drivers and passengers with a range of advanced features and functions. Here are some examples of modern embedded systems in cars:

1. Advanced Driver Assistance Systems (ADAS): ADAS uses a combination of sensors, cameras, and software to provide drivers with real-time information and assistance, allowing them to make better decisions on the road and avoid accidents. Features such as Automatic Emergency Braking (AEB), Lane Departure Warning (LDW), and Blind Spot Monitoring (BSM) are all examples of ADAS.
2. Infotainment Systems: Infotainment systems provide drivers and passengers with a range of entertainment and communication options, including music and video playback, navigation, and smartphone integration. Many modern infotainment systems also include voice recognition and gesture control features.
3. Telematics Systems: Telematics systems use wireless communication technologies to provide remote access to a range of vehicle data, including location, speed, and fuel consumption. This data can be used to improve vehicle performance and maintenance, as well as to provide drivers with real-time information about traffic conditions and potential hazards.
4. Engine Management Systems: Engine management systems use sensors and software to monitor and control various aspects of engine performance, including fuel injection, ignition timing, and exhaust emissions. These systems help to optimize engine performance, improve fuel efficiency, and reduce emissions.
5. Climate Control Systems: Climate control systems use sensors and software to monitor and control the temperature, humidity, and air quality inside the vehicle. These systems provide drivers and passengers with a comfortable and healthy environment, regardless of the weather conditions outside.

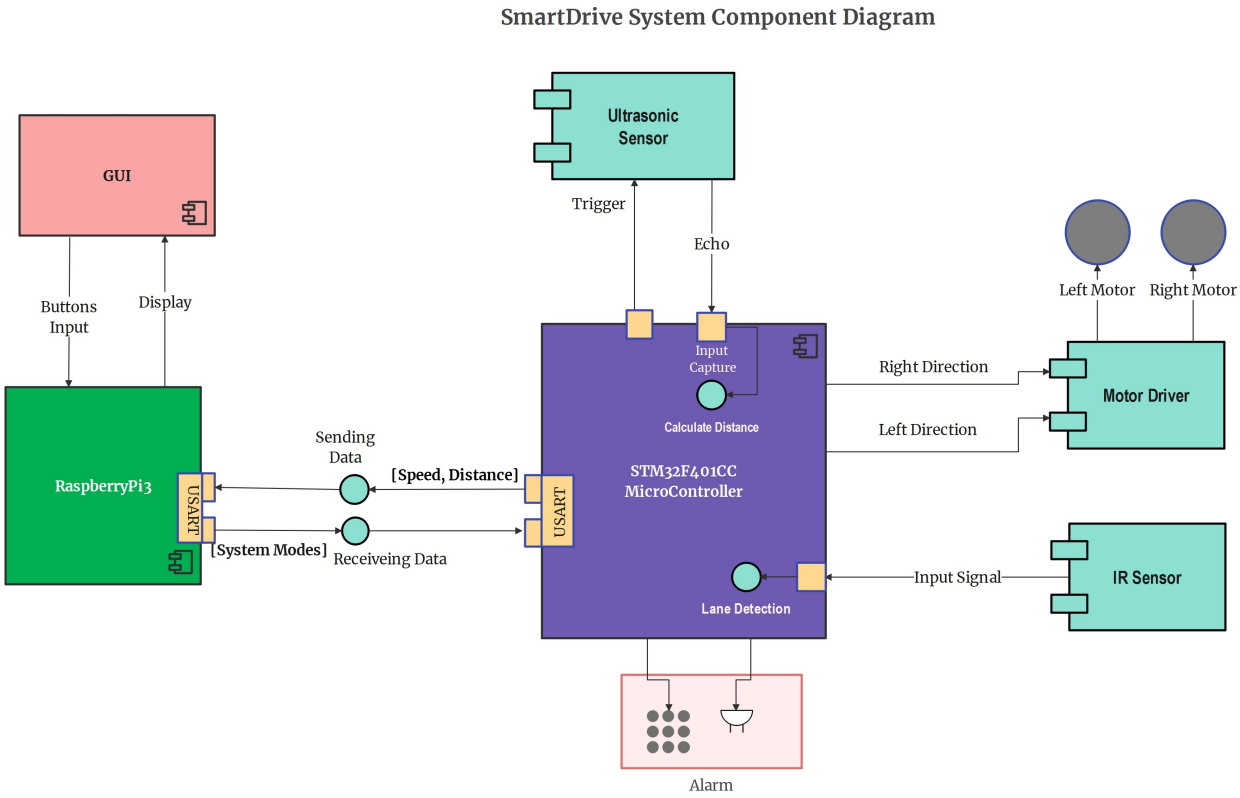
These are just a few examples of the many embedded systems in modern cars. As technology continues to advance, we can expect to see even more sophisticated systems that will enhance the driving experience and improve safety on the road.

3.4 Embedded Platforms

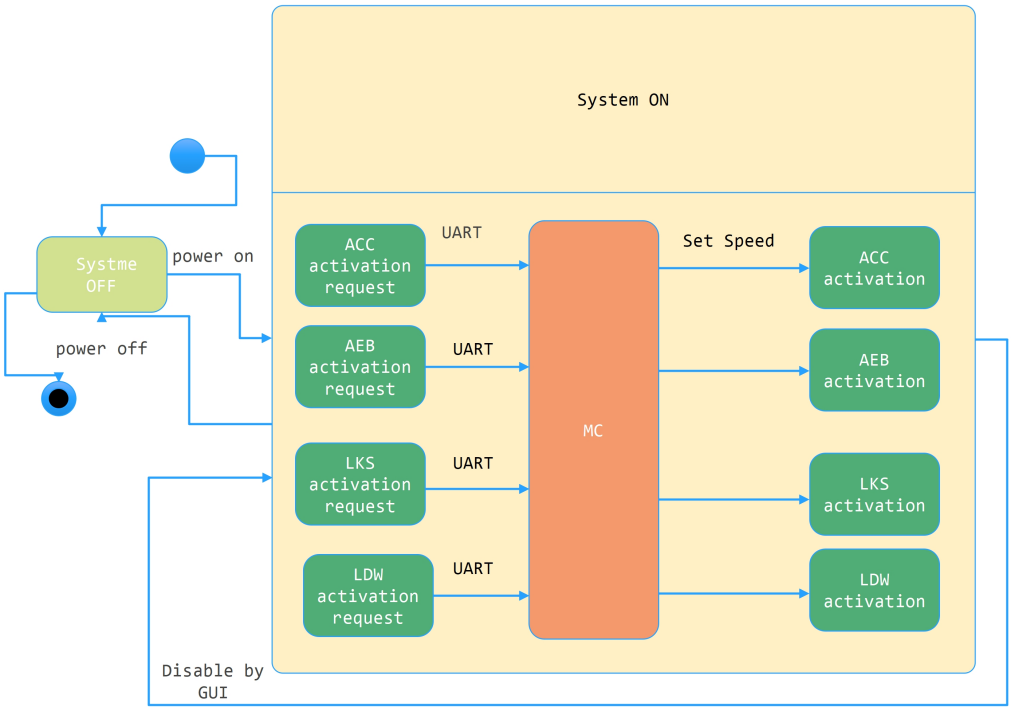
There are several embedded platforms that are commonly used in the development of embedded systems. Here are some of the most common embedded platforms:

1. Arduino: Arduino is an open-source platform that provides a range of microcontrollers and development boards for building embedded systems. Arduino is popular due to its ease of use and low cost, making it a great choice for hobbyists and small-scale projects.
2. Raspberry Pi: Raspberry Pi is a series of single-board computers that are designed for a range of applications, including education, hobbyist projects, and industrial automation. Raspberry Pi is popular due to its low cost, versatility, and large community of users and developers.
3. BeagleBone: BeagleBone is a series of single-board computers that are similar to Raspberry Pi. BeagleBone is popular due to its high performance, large number of input/output pins, and ability to run multiple operating systems.
4. STM32: STM32 is a series of microcontrollers that are designed for a range of applications, including industrial automation, automotive systems, and consumer electronics. STM32 is popular due to its low power consumption, high performance, and strong ecosystem of development tools and resources.
5. Texas Instruments LaunchPad: Texas Instruments LaunchPad is a series of development boards that are designed for use with the company's microcontrollers. LaunchPad is popular due to its low cost, ease of use, and strong community of users and developers.
6. PIC: PIC is a series of microcontrollers that are designed for a range of applications, including automotive systems, industrial automation, and medical devices. PIC is popular due to its low power consumption, small size, and strong ecosystem of development tools and resources.

4 ADAS Features



4 SmartDrive system Diagram



5 Introduction:
Advanced Driver Assistance Systems (ADAS) have revolutionized the automotive industry by integrating cutting-edge technologies into vehicles to enhance safety, convenience, and overall driving experience. ADAS encompasses a wide range of features and functionalities that assist drivers in various aspects of their journey, ultimately aiming to reduce the risk of accidents and improve road safety. From collision avoidance to adaptive cruise control and lane-keeping assistance, ADAS has become a crucial component in modern vehicles.

Importance of ADAS:

1. Safety Enhancement: ADAS technologies play a pivotal role in improving safety on the roads. By utilizing sensors, cameras, and advanced algorithms, ADAS features can detect potential dangers, warn drivers, and even take proactive measures to prevent accidents. This includes automatic emergency braking, forward collision warning, blind-spot detection, and lane departure warning systems. These technologies act as an extra set of eyes and reflexes, significantly reducing the likelihood of collisions caused by human error or inattentiveness.
2. Accident Prevention: ADAS technologies are designed to anticipate and mitigate potential hazards on the road. Through real-time monitoring of the vehicle's surroundings, ADAS systems can identify dangerous situations, such as sudden obstacles, pedestrians, or erratic traffic behavior, and provide timely alerts or intervene autonomously. By assisting drivers in avoiding accidents, ADAS helps to save lives and prevent injuries.
3. Driver Assistance and Comfort: ADAS features not only prioritize safety but also enhance driving convenience and comfort. Adaptive cruise control maintains a set distance from the vehicle ahead, reducing driver fatigue during long trips. Lane-keeping assistance systems provide gentle steering inputs to keep the vehicle centered within the lane, reducing driver effort. These features improve overall driving comfort and reduce the cognitive load on the driver, resulting in a more relaxed and enjoyable driving experience.

4. Traffic Efficiency: ADAS technologies contribute to optimizing traffic flow and improving overall traffic efficiency. Features such as traffic sign recognition, intelligent navigation systems, and predictive analytics can help drivers make informed decisions, select the best routes, and optimize fuel consumption. By minimizing congestion and improving traffic management, ADAS contributes to reducing emissions and enhancing the sustainability of transportation systems.
5. Stepping Stone to Autonomous Vehicles: ADAS serves as a stepping stone toward the development of fully autonomous vehicles. By gradually introducing autonomous functionalities, such as autonomous parking or highway driving assistance, ADAS systems are paving the way for a future where vehicles can operate independently without human intervention. This transition is expected to bring further advancements in safety, efficiency, and accessibility in the realm of transportation.

In conclusion, ADAS technologies are revolutionizing the automotive industry by prioritizing safety, enhancing driving convenience, and contributing to overall traffic efficiency. With its potential to prevent accidents, reduce human error, and pave the way for autonomous vehicles, ADAS is a significant leap forward in making roads safer and more efficient for drivers, passengers, and pedestrians alike.

Adaptive Cruise Control ACC(see page 12)

Automatic Emergency Braking AEB(see page 27)

Lane Departure Warning (LDW)(see page 46)

Lane Keeping Assist System (LKAS)(see page 61)

4.1 Adaptive Cruise Control ACC

4.1.1 Introduction

Adaptive Cruise Control system (ACC) is a system that reduces the driving burden on the driver. The ACC system primarily supports four driving modes on the road and controls the acceleration and deceleration of the vehicle in order to maintain a set speed or to avoid a crash. This paper proposes more accurate methods of detecting the preceding vehicle by radar while cornering, with consideration for the vehicle sideslip angle, and also of controlling the distance between vehicles. By making full use of the proposed identification logic for preceding vehicles and path estimation logic, an improvement in driving stability was achieved.

Adaptive Cruise Control (ACC) is an advanced automotive technology designed to reduce the driving burden on the driver and enhance road safety. It is particularly useful in mitigating the risks associated with car accidents and preventing collisions. ACC systems maintain a set speed for the vehicle and automatically adjust it to match the speed of the preceding vehicle while maintaining a safe distance.

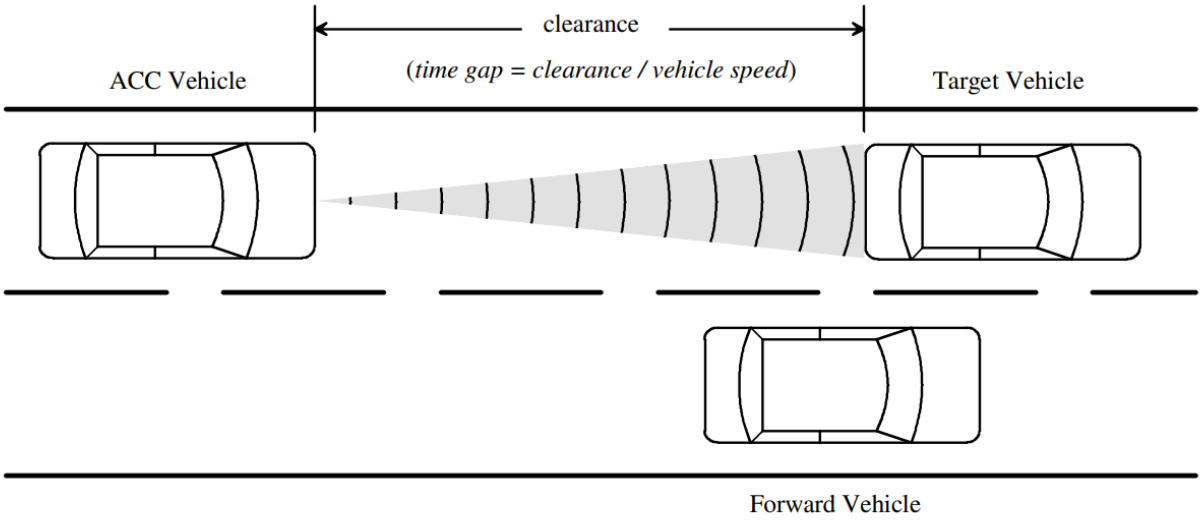
The primary objective of ACC is to provide intelligent control over vehicle acceleration and deceleration to ensure a consistent speed and prevent potential collisions. The system achieves this by utilizing various sensors, such as millimeter wave radar, to detect the presence of preceding vehicles on the road. By continuously monitoring the position and speed of the preceding vehicle, ACC can calculate the appropriate acceleration or deceleration required to maintain a safe following distance.

ACC systems take into account factors like the vehicle's turning radius, yaw rate, and steering angle to estimate the position of the preceding vehicle after each radar scan. This estimation helps the system maintain the appropriate distance from the preceding vehicle, even when there are changes in the driving conditions, such as curves or variations in speed.

One key aspect of ACC is its ability to adjust the following distance based on the behavior of the second vehicle ahead. If the preceding vehicle performs sudden braking or acceleration, the ACC system responds accordingly to maintain a safe and comfortable driving experience for the driver. By considering the actions of the second vehicle ahead, ACC can avoid unnecessary jerky motions and improve ride quality.

Overall, ACC technology plays a crucial role in enhancing road safety by reducing the risk of rear-end collisions and providing a more relaxed driving experience. It allows drivers to maintain a safe distance from the preceding vehicle, adapt to changing traffic conditions, and reduce the cognitive load associated with constant speed adjustments. ACC systems have become increasingly prevalent in modern vehicles, contributing to improved driving stability and accident prevention.

4.1.2 Definitions and Physical Overview



6 Figure 1 – ACC Vehicle Relationships

***Adaptive Cruise Control (ACC)* –** An enhancement to a conventional cruise control system which allows the ACC vehicle to follow a forward vehicle at an appropriate distance.

***ACC vehicle* –** the subject vehicle equipped with the ACC system.

***active brake control* –** a function which causes application of the brakes without driver application of the brake pedal.

***clearance* –** distance from the forward vehicle's trailing surface to the ACC vehicle's leading surface.

***forward vehicle* –** any one of the vehicles in front of and moving in the same direction and traveling on the same roadway as the ACC vehicle.

***set speed* –** the desired cruise control travel speed set by the driver and is the maximum desired speed of the vehicle while under ACC control.

4.1.2.1 system states

ACC off state – direct access to the 'ACC active' state is disabled.

ACC standby state – system is ready for activation by the driver.

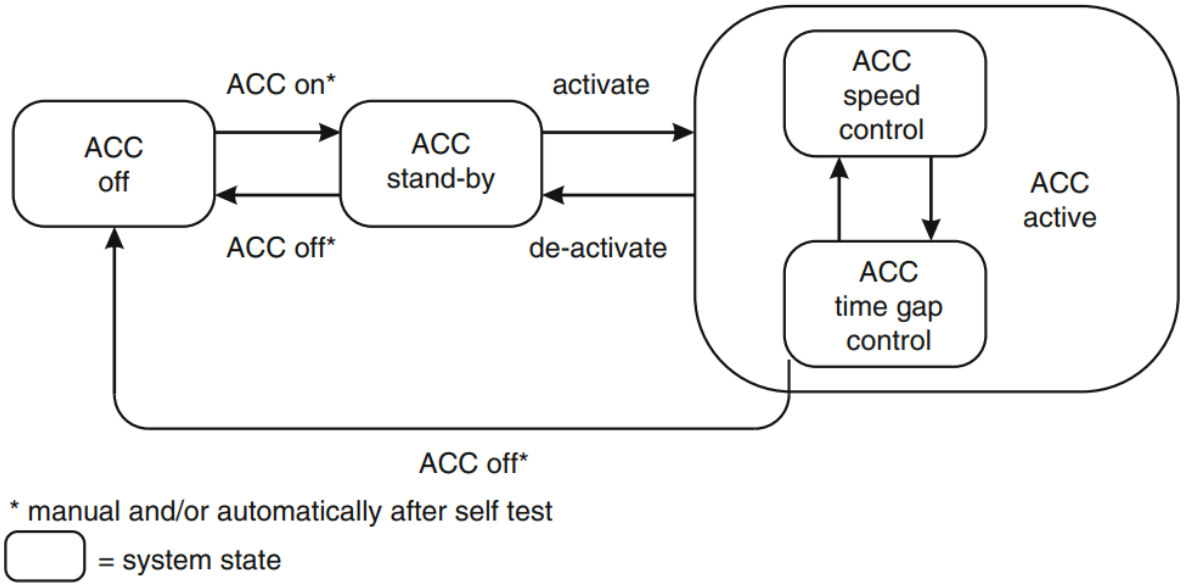
ACC active state – the ACC system is in active control of the vehicle's speed.

ACC speed control state – a substate of 'ACC active' state in which no forward vehicles are present such that the ACC system is controlling vehicle speed to the 'set speed' as is typical with conventional cruise control systems.

ACC time gap control state – a substate of 'ACC active' state in which time gap, or headway, between the ACC vehicle and the target vehicle is being controlled.

target vehicle – one of the forward vehicles in the path of the ACC vehicle that is closest to the ACC vehicle.

time gap – the time interval between the ACC vehicle and the target vehicle. The 'time gap' is related to the 'clearance' and vehicle speed by: *time gap = clearance / ACC vehicle speed*



7 Figure 2 - States and transitions according to ISO15622

4.1.3 Primary ACC Modes

The ACC system supports four control modes, which are described below and shown in Figure 1[1].

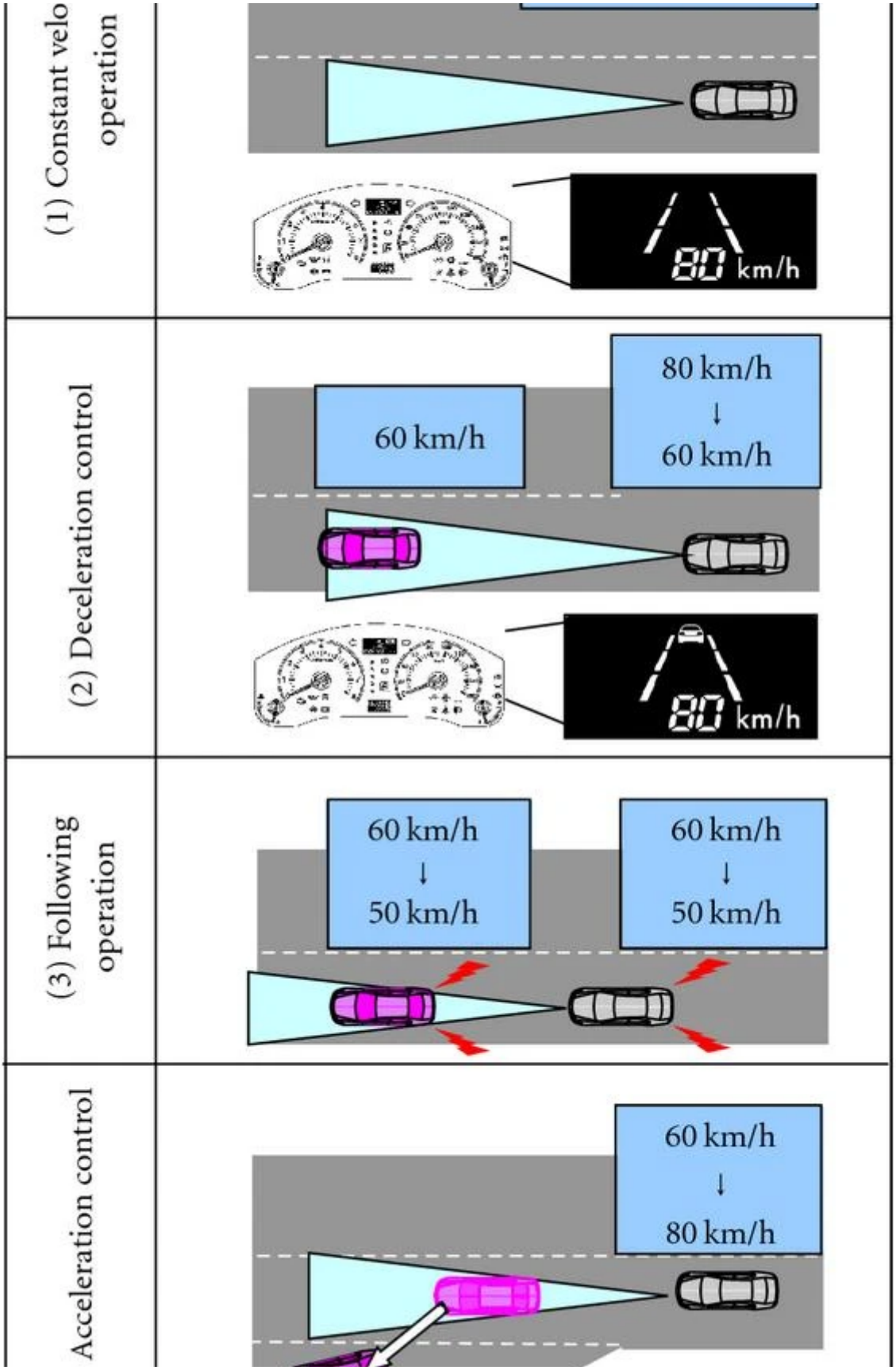
(1) *Constant velocity control*: when there are no vehicles straight ahead, or when there is a large distance between the driver's vehicle and the preceding vehicle, the system maintains a constant vehicle velocity.

(2) *Deceleration control*: when a vehicle traveling ahead at a slower speed is detected, the system uses the throttle to decelerate the driver's vehicle. If this deceleration is insufficient, the system uses the brake to decelerate the vehicle.

(3) *Following control*: when the driver's vehicle is following behind the preceding vehicle, the system controls the throttle and brake so that the time interval between the vehicles (which corresponds to a distance between the vehicles that is proportional to the velocity of the driver's vehicle) is the time which was set by the driver.

(4) *Acceleration control*: when, due to a lane change, there is no longer a vehicle ahead of the driver's vehicle, the system accelerates the vehicle up to the velocity set by the driver, and then maintains a constant velocity. Furthermore, when the driver's vehicle approaches a vehicle ahead of it without slowing down enough, an alert buzzer and display prompt the driver to apply the brakes or take other appropriate action.

1 https://asp-eurasipjournals.springeropen.com/articles/10.1155/2010/295016#Fig1


(1) Constant velo operation
80 km/h
(2) Deceleration control
60 km/h
80 km/h
↓
60 km/h
80 km/h
(3) Following operation
60 km/h
↓
50 km/h
60 km/h
↓
50 km/h
Acceleration control
60 km/h
↓
80 km/h

8 Figure 1

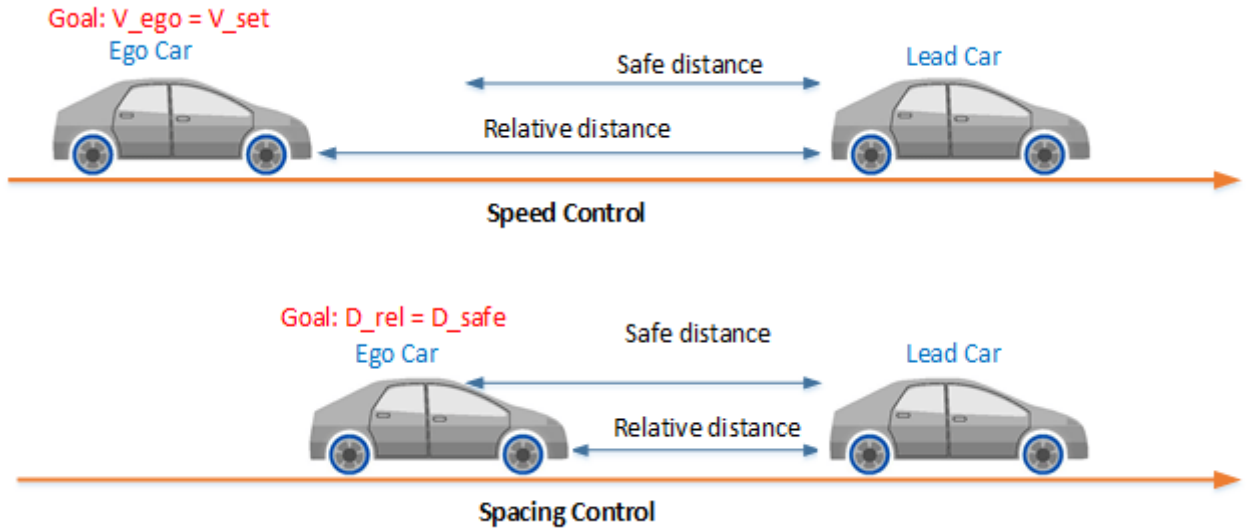
An Example Adaptive Cruise Control System block in MathWorks Simulink:

A vehicle (ego car) equipped with adaptive cruise control (ACC) has a sensor, such as radar, that measures the distance to the preceding vehicle in the same lane (lead car). The sensor also measures the relative velocity of the lead car. The ACC system operates in the following two modes:

Speed control: The ego car travels at a driver-set speed.

Spacing control: The ego car maintains a safe distance from the lead car.

The ACC system decides which mode to use based on real-time radar measurements. For example, if the lead car is too close, the ACC system switches from speed control to spacing control. Similarly, if the lead car is further away, the ACC system switches from spacing control to speed control. In other words, the ACC system makes the ego car travel at a driver-set speed as long as it maintains a safe distance.



9 Figure 2

4.1.4 ACC Requirements

4.1.4.1 Functional Requirements for Adaptive Cruise Control (ACC) According to ISO 15622:

1. **Under free cruising conditions:**
   - Provide constant speed control with high control comfort, minimizing longitudinal jerk and swinging.
   - Maintain high control quality with no significant deviation from the set speed.
   - Implement cruise control with brake intervention when the desired speed is lowered or during incline travel.
2. **When following another vehicle:**
   - Implement vehicle-following control with adjustable swinging damping to prevent copying speed fluctuations from the vehicle ahead.

- Maintain the set time gap between the vehicles, allowing gradual "falling back" when the interval is greatly shortened due to vehicles cutting in.
- Provide control dynamics that meet the driver's expectations.
- Ensure convoy stability when following other ACC-equipped vehicles.
- Allow for adequate acceleration capability for dynamic following situations.
- Enable deceleration in the majority of pursuit driving situations (90%) in moving traffic.
- Automatically detect and respond to cut-in or cut-out situations within a defined distance range, creating a target-seeking corridor.

3. **When approaching:**
    - Promptly adjust the speed for slow approaches to maintain the desired distance.
    - Provide a predictable deceleration course for faster approaches, allowing the driver to assess whether intervention is needed due to inadequate ACC deceleration.
    - "Fall back" in a standard driving manner if the vehicle gets closer than the desired clearance.
4. **Functional limits:**
    - Disable control at very low speeds, below a minimum speed threshold (ISO 15622: vlow <= 5 m/s) and hand over control to the driver.
    - Set a minimum set speed (vset,min) above 7 m/s (>=30 km/h speedometer values).
    - Maintain a time gap (tmin) not below 1 second in the steady state.
    - Prioritize driver intervention, deactivate ACC when the brake pedal is depressed, and override ACC when the accelerator pedal is depressed.
    - Allow the driver to set the desired speed (vset) and desired time gap (tset).
    - Facilitate appropriate handover of longitudinal control to the driver in the event of system failure, particularly during deceleration.
    - Limit acceleration within the range of Amin = -3.5 m/s^2 to Amax = 2.5 m/s^2.

4.1.4.2 Hardware Requirements:

1. **4WD Robot Car:**
    - A 4-wheel drive (4WD) robot car platform capable of accommodating the necessary components and providing stability and maneuverability.
    - Sufficient space and mounting points for integrating the sensors, microcontroller, speed sensor, servo motor, and other required hardware.
2. **RPLidar Sensor or Ultrasonic Sensor:**
    - RPLidar Sensor: A 360-degree laser range-finding sensor capable of accurately detecting obstacles and measuring distances in real-time. It should have a suitable scanning range and resolution for the application.
    - Ultrasonic Sensor: Multiple ultrasonic sensors placed strategically on the robot car to provide accurate distance measurement and obstacle detection in the vicinity.
3. **LM393 OPTO-SPEED SENSOR:**
    - A speed sensor capable of measuring the robot car's current speed accurately. This can be accomplished using an encoder attached to the wheels or other appropriate speed sensing mechanisms.
4. **STM32F401 Microcontroller:**
    - An STM32F401 microcontroller board or a compatible development board with sufficient processing power and I/O capabilities to handle sensor data processing, control algorithms, and communication with other components.
    - The microcontroller should have the necessary peripherals for interfacing with the sensors, speed sensor, servo motor, and LCD display.
5. **Servo Motor:**
    - A servo motor capable of controlling the robot car's steering mechanism. The motor should provide accurate and precise steering control based on the control algorithms implemented in the system.
6. **LCD Display:**

- An LCD display module for providing real-time feedback to the driver, displaying the current speed, detected obstacles, system status, and other relevant information.

7. **Power Supply:**
   - A reliable power supply system capable of providing sufficient voltage and current to power the entire adaptive cruise control system, including the microcontroller, sensors, servo motor, and LCD display.
   - The power supply should be stable and able to handle peak power demands during operation.
8. **Communication Interfaces:**
   - The necessary communication interfaces to facilitate data exchange between the microcontroller and other components, such as USB, UART, SPI, or I2C interfaces.
   - These interfaces will enable sensor data acquisition, control signals for the servo motor, and communication with external devices or a computer for debugging and configuration.

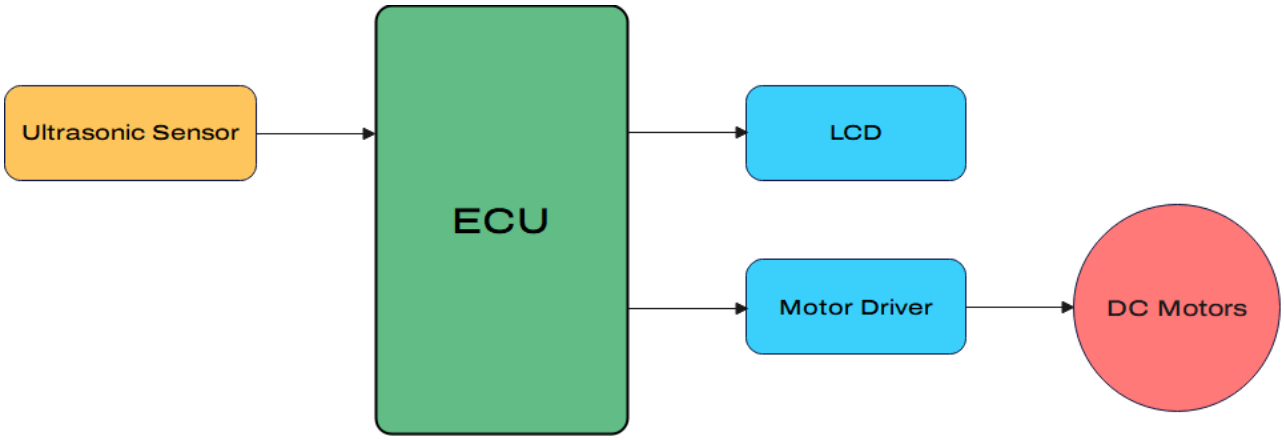
4.1.4.3 Software Requirements:

1. **Development Environment:**
   - STM32CubeIDE or Keil MDK: Integrated development environments (IDEs) for programming and debugging the STM32F401 microcontroller.
   - Compiler: A C or C++ compiler compatible with the microcontroller's architecture.
2. **Sensor Integration:**
   - Sensor Libraries: Libraries or software frameworks for interfacing and integrating the RPLidar sensor or ultrasonic sensors with the microcontroller.
   - Sensor Data Processing: Algorithms to process sensor data and extract relevant information such as distance measurements and obstacle detection.
3. **Control Algorithms:**
   - Speed Control: Develop algorithms for maintaining a constant speed and achieving high control comfort with minimal longitudinal jerk and swinging.
   - Vehicle-following Control: Implement algorithms to adjust the speed and maintain a safe distance from the vehicle ahead while damping swinging motions.
   - Time-gap Control: Design algorithms to control the time gap between vehicles and enable gradual "falling back" when the interval is greatly shortened.
   - Acceleration and Deceleration Control: Algorithms to provide adequate acceleration capability for dynamic following and appropriate deceleration in pursuit driving situations.
   - Target Detection: Develop algorithms to automatically detect and respond to cut-in or cut-out situations within a defined distance range.
4. **Real-time Operating System (RTOS):**
   - If necessary, select and configure an RTOS to ensure timely execution of control algorithms and sensor data processing tasks.
5. **User Interface:**
   - LCD Display Integration: Implement software routines to display real-time information on the LCD, including current speed, detected obstacles, system status, and user prompts.
6. **Communication:**
   - Serial Communication: Implement software protocols (e.g., UART) for communication between the microcontroller and external devices or a computer for debugging and configuration purposes.
7. **Testing and Debugging:**
   - Test Plan: Develop a comprehensive test plan with detailed test cases to validate the performance of the adaptive cruise control system, including speed adjustment accuracy, obstacle detection, and collision avoidance.
   - Debugging Tools: Utilize debugging features provided by the development environment and microcontroller board to identify and resolve software issues.
8. **Documentation:**

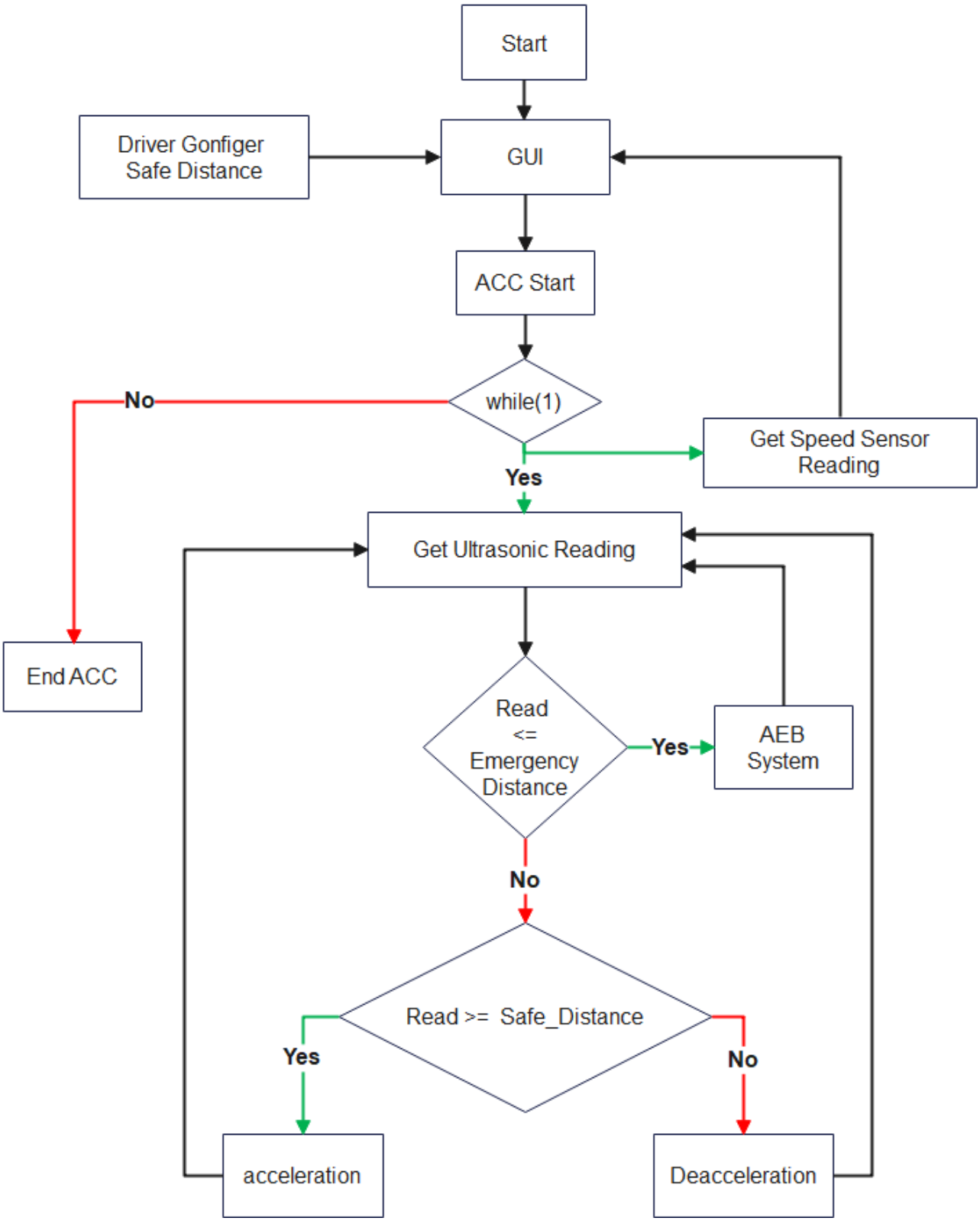
- System Design Document: A detailed document describing the software architecture, control algorithms, and integration procedures.
- User Manual: Step-by-step instructions on setting up and operating the adaptive cruise control system, including sensor calibration and system configuration.
- Safety Considerations: Document safety measures and precautions associated with the software operation.
- Technical Documentation: Detailed documentation of the software components, libraries, and their integration for future reference and troubleshooting.

4.1.5 ACC Design

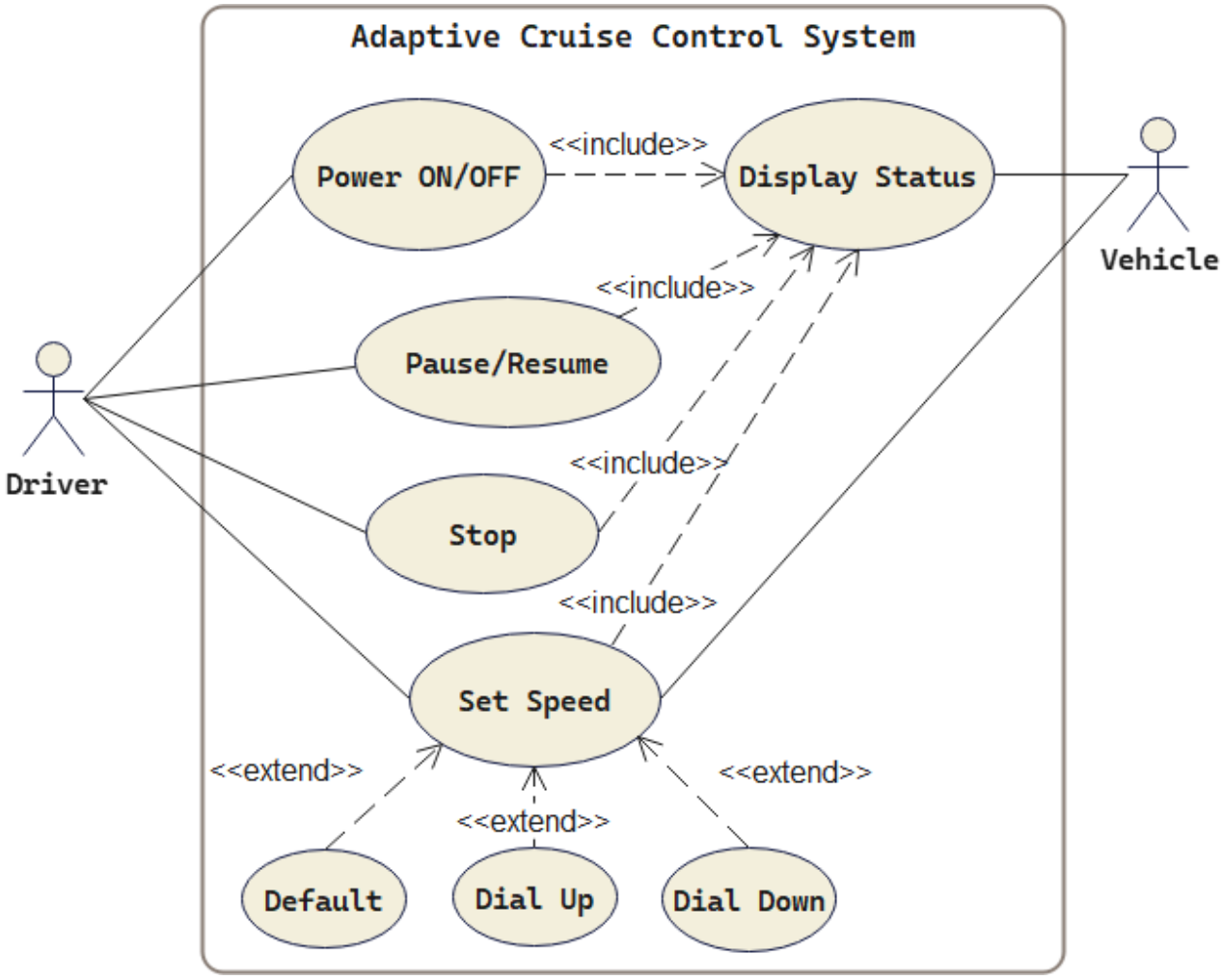
4.1.5.1 1.Block Diagram



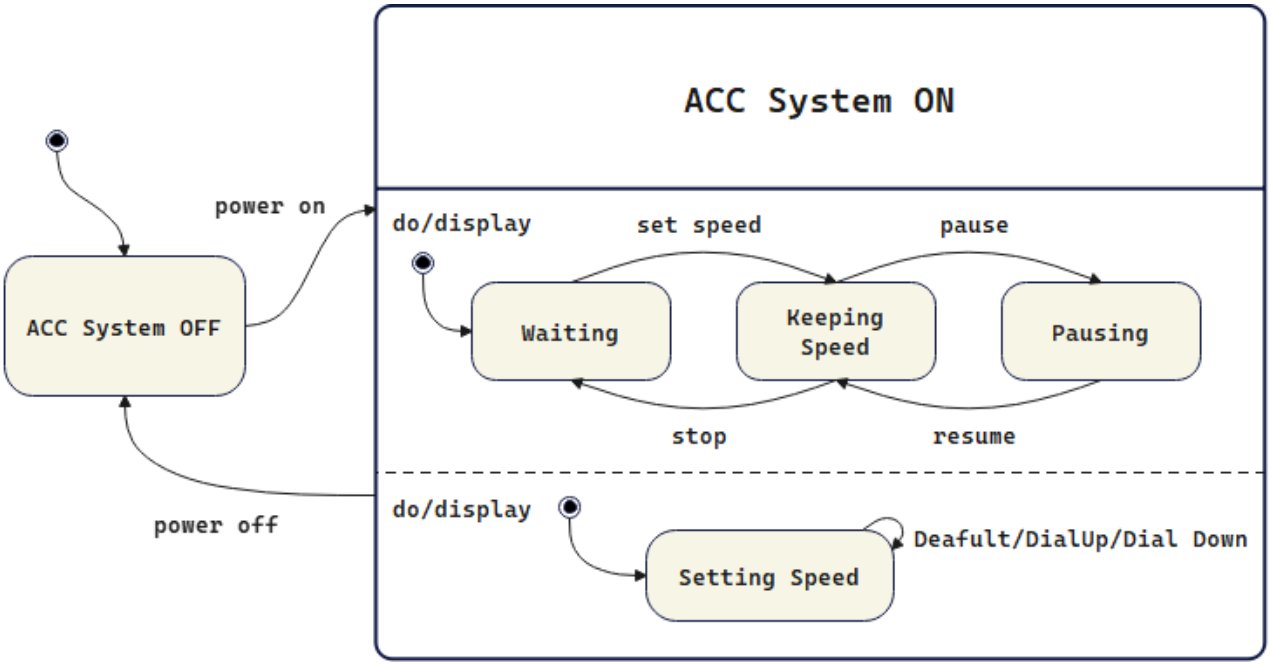
10 Figure 1- Block Diagram of ACC



11 Figure 2 ACC System Flowchart

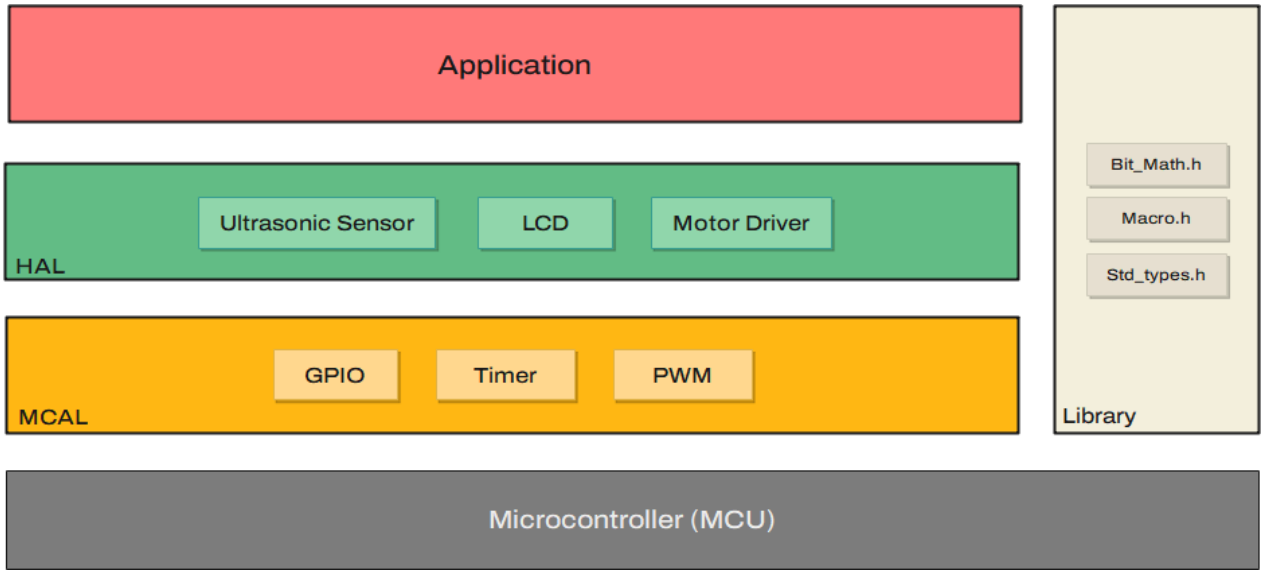


12 Figure 3 - ACC Use Case Diagram



13 Figure 4 - ACC State Chart

4.1.5.2 2.Layered Architecture



Components and Modules

APIs and data types for implementing Adaptive Cruise Control (ACC) on the STM32F401CC microcontroller using a 4-wheel motor, LCD display, LM393 opto-speed sensor, and ultrasonic sensor:

```
// Data types

typedef struct {
    float set_speed; // Set speed in km/h
```

```
    float current_speed; // Current speed in km/h
    float distance_to_obstacle; // Distance to the obstacle in meters
} ACC_Data_t;

// Function APIs

// Initializes the ACC system
void ACC_vInit(void);

// Starts the ACC system
void ACC_vStart(void);

// Stops the ACC system
void ACC_vStop(void);

// Sets the desired speed for the ACC system
void ACC_vSetSpeed(float speed);

// Gets the current speed of the vehicle
float ACC_f32GetCurrentSpeed(void);

// Gets the distance to the obstacle in front of the vehicle
float ACC_f32GetDistanceToObstacle(void);

// Updates the speed and distance data for the ACC system
void ACC_vUpdateData(void);

// Gets the ACC behavior option selected using the keypad
u8 ACC_u8GetOption(void);

// Controls the vehicle speed based on the ACC system data
void ACC_vControlVehicle(void);

// Displays the ACC system data on the LCD display
void ACC_vDisplayData(void);

// Reads the speed from the LM393 opto-speed sensor
float ACC_f32ReadSpeed(void);

// Reads the distance from the ultrasonic sensor
float ACC_f32ReadDistance(void);

// Controls the 4-wheel motor to adjust the vehicle speed
void ACC_vControlMotor(float speed);
```

- US sensor API:

```
void US_vInit(); // initialize the US sensor
f32 US_f32MeasureDistance(); // measure the distance to the obstacle
```

- Motor driver API:

```
void Motor_vInit(); // initialize the motor driver
void Motor_vSetSpeed(u8 speed); // set the speed of the motors
void Motor_vSetDirection(u8 direction); // set the direction of the motors
```

GPIO API:

```
void MCL_GPIO_vInitPort(GPIO_Reg_t* GPIOx, GPIO_Cfg_t* Copy_GPIO_Cfg);
void MCL_GPIO_vSetPinCfg(GPIO_Reg_t* GPIOx, GPIO_Pin_t Copy_PinId, GPIO_Cfg_t*
Copy_GPIO_Cfg);
void MCL_GPIO_vDirectPinSetter(GPIO_Reg_t* GPIOx, GPIO_Pin_t Copy_PinId, Pin_State_t
Copy_xPinVal);
u8  MCL_GPIO_u8GetPinVal(GPIO_Reg_t* GPIOx, GPIO_Pin_t Copy_PinId);
void MCL_GPIO_vWritePort(GPIO_Reg_t* GPIOx, u32 Copy_u32PortVal);
u32 MCL_GPIO_u32ReadPort(GPIO_Reg_t* GPIOx);
void MCL_GPIO_vTogglePin(GPIO_Reg_t* GPIOx, GPIO_Pin_t Copy_PinId);
```

Timer API:

```
void Timer_vInit(TIM_Config_t* ConfigPtr);

void Timer_vDisableNotification(TIM_TypeDef* TIMx); //no notification, no ISR will
operate

void Timer_vEnableNotification(TIM_TypeDef* TIMx);

void Timer_vStart(TIM_TypeDef* TIMx, u32 Value);

void Timer_vStop(TIM_TypeDef* TIMx);

u32 Timer_u32GetTimeElapsed(TIM_TypeDef* TIMx);

u32 Timer_u32GetTimeRemaining(TIM_TypeDef* TIMx);

Std_ReturnType Timer_xGetPredefTimerValue(Gpt_PredefTimer_t PredefTimer, u32*
TimeValuePtr);

void Timer_vClearFlag(TIM_TypeDef* TIMx);
```

LCD Display API functions:

```
void LCD_vInit(void);
void LCD_vSendCommand(u8 Copy_u8Command);
void LCD_vSendData(u16 Copy_u16Data);
void LCD_vSendNum(u16 Copy_u16Num);
void LCD_vSendChar(u8 Copy_u8Char);
void LCD_vSendString(u8* Copy_pu8String);
void LCD_vGotoXY(u8 Copy_u8Line, u8 Copy_u8Position);
```

```
void LCD_vInitGPIO(void);
void LCD_vClearScreen(void);
void LCD_vEnableSignal(void);
```

LM393 Opto-Speed Sensor API functions:

```
// Initializes the opto-speed sensor
void SpeedSensor_vInit(void);

// Reads the current speed from the opto-speed sensor
float SpeedSensor_f32ReadSpeed(void);

// Calculates the current speed from the opto-speed sensor data
float SpeedSensor_f32CalculateSpeed(u32 ticks, u32 elapsed_time);

// Interrupt service routine for the opto-speed sensor input capture
void SpeedSensor_vInputCaptureISR(void);
```

API functions for a keypad:

```
// Initializes the keypad
void Keypad_vInit(void);

// Reads the current key pressed on the keypad
u8 Keypad_u8ReadKey(void);

// Waits for a key press on the keypad and returns the key value
u8 Keypad_u8WaitForKey(void);
```

Data types:

```
typedef struct
{

    GPIO_Mode_t GPIO_Mode;          //mode of GPIO selected port/pin
    //this parameter can be a value of  @ref GPIO_Mode_define

    u8 GPIO_Speed;          //speed of GPIO selected port/pin
    //this parameter can be a value of  @ref GPIO_Speed_define

}GPIO_Cfg_t;

//@ref ADC_Channels_Define
//select ADC Channel to receive from
typedef enum{
    //  Bits 4:0 - MUX4:0: Analog Channel and Gain Selection Bits
    ADC_CHANNEL0,
```

```
    ADC_CHANNEL1,
    ADC_CHANNEL2,
    ADC_CHANNEL3,
    ADC_CHANNEL4,
    ADC_CHANNEL5,
    ADC_CHANNEL6,
    ADC_CHANNEL7,
    ADC_CHANNEL8,
    ADC_CHANNEL9,
}ADC_Channel_t;

typedef struct {
    ADC_Channel_t channel;      // ADC channel to use
    u8 resolution;   // ADC resolution (e.g. 8-bit, 10-bit, etc.)
    u8 clock_source; // ADC clock source (e.g. internal, external, etc.)
    u32 Sampling_Time; // ADC sampling time
} ADC_Config_t;

//@ref Timer_Clock_Define
typedef enum {
    TIM_NOCLKSRC = ~(7 << 0),
    TIM_NOPRESCALER = 1,
    TIM_CLKPRESCALER_2,
    TIM_CLKPRESCALER_4,
    TIM_CLKPRESCALER_8,
    TIM_CLKPRESCALER_16,
    TIM_CLKPRESCALER_32,
    TIM_CLKPRESCALER_64,
    TIM_CLKPRESCALER_128,
    TIM_CLKPRESCALER_256,
    TIM_CLKPRESCALER_512,
    TIM_CLKPRESCALER_1024
} TIM_ClkSrc_t;

//@ref TIM_Mode_Define
typedef enum {
    TIM_MODE_TIMING = 0,
    TIM_MODE_PWM = 1,
    TIM_MODE_INPUT_CAPTURE = 2,
    TIM_MODE_OUTPUT_COMPARE = 3
} TIM_Mode_t;

//@ref TIM_Output_Mode_Define
typedef enum {
    TIM_OUTPUT_MODE_NORMAL = 0,
    TIM_OUTPUT_MODE_TOGGLE = 1,
    TIM_OUTPUT_MODE_PWM1 = 2,
    TIM_OUTPUT_MODE_PWM2 = 3
} TIM_Output_Mode_t;

// @ref TIM_IRQ_Define
typedef enum {
    TIM_IRQ_DISABLE = 0,
```

```
    TIM_IRQ_ENABLE = 1
} TIM_IRQ_t;

typedef struct {
    TIM_Mode_t TIM_Mode;                        //@ref TIM_Mode_Define
    TIM_ClkSrc_t TIM_ClockSrc;                  //@ref Timer_Clock_Define
    TIM_Output_Mode_t TIM_Output_Mode;          //@ref TIM_Output_Mode_Define
    TIM_IRQ_t TIM_IRQ_Enable;                   //@ref TIM_IRQ_Define
    uint16_t TIM_Period;                        //Timer period value
    uint16_t TIM_Prescaler;                     //Timer prescaler value
    uint16_t TIM_Pulse;                         //Timer pulse value
    void (*TIM_IRQ_Callback)(void);             //Callback function for TIM IRQ
} TIM_Config_t;

typedef float distance_t;
typedef int speed_t;
typedef int direction_t;
typedef bool bool_t;
typedef int raw_adc_value_t;
typedef float voltage_t;
typedef float duty_cycle_t;
```

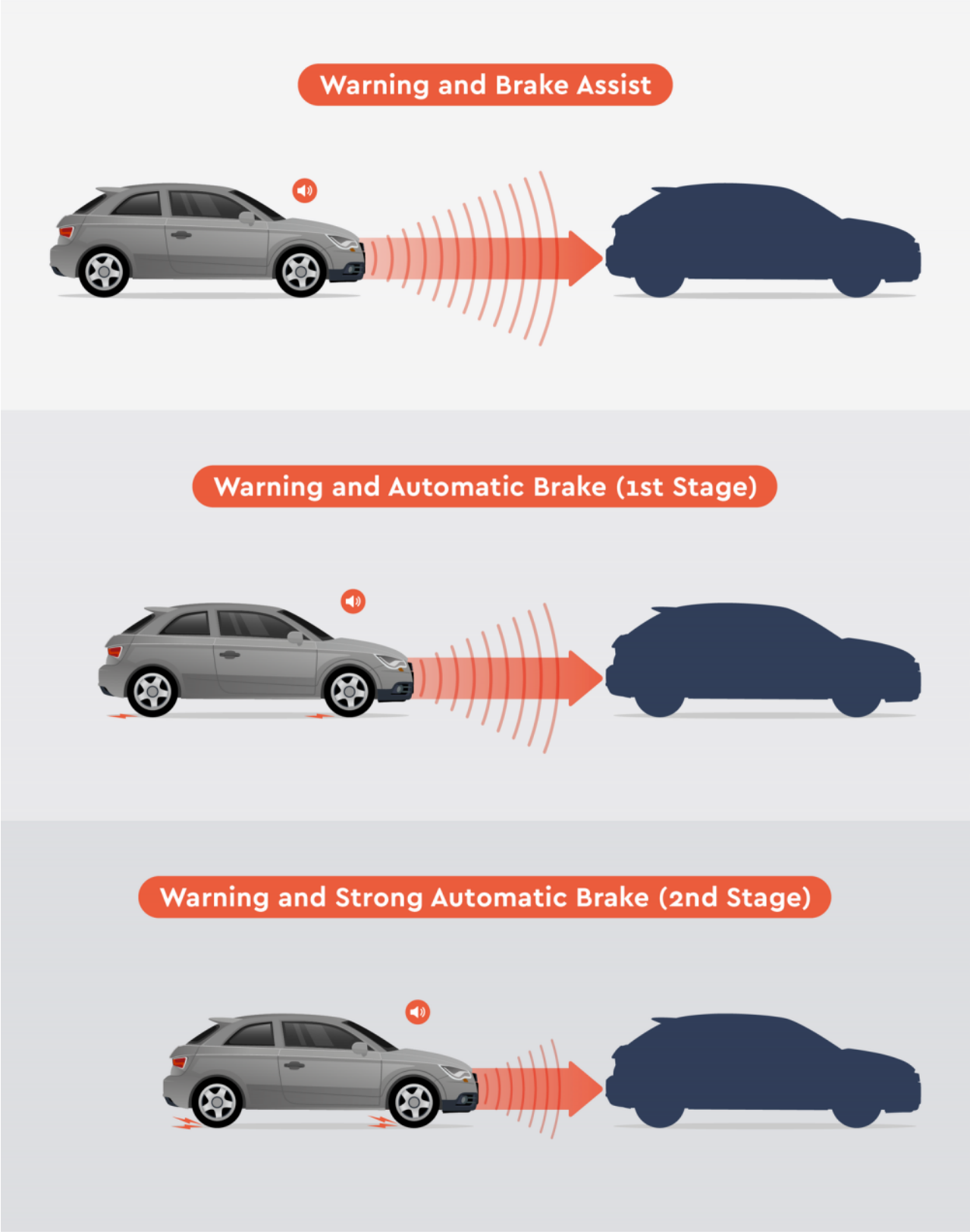
4.2 Automatic Emergency Braking AEB

4.2.1 Introduction

Active Safety, Advanced Driver Assistance Systems (ADAS) are now being introduced to the marketplace as they serve as key enablers for anticipated autonomous driving systems. Automatic Emergency Braking (AEB) is one ADAS application that is either in the marketplace presently or under development as nearly all automakers have pledged to offer this technology by the year 2022. Vehicles have become a lot safer in the last decade, thanks to the introduction of various new active and passive safety systems. You may drive your automobile without concern, ensuring that the safety features will function properly. In recent years, driver-aid safety technologies such as autonomous emergency braking have grown more common in automobiles. The name alone may give you an indication of what AEB performs.

4.2.1.1 What is AEB?

An increasingly common feature of new cars, AEB is one of the most important car safety inventions of recent years. AEB will detect obstacles on the road and brake automatically to slow down or completely bring the vehicle to a stop.


Warning and Brake Assist
Warning and Automatic Brake (1st Stage)
Warning and Strong Automatic Brake (2nd Stage)

4.2.1.2 What does AEB do?

AEB will automatically brake the car without the driver's influence. This greatly reduces the likelihood of a car crash and saves lives.

4.2.1.3 How does AEB work?

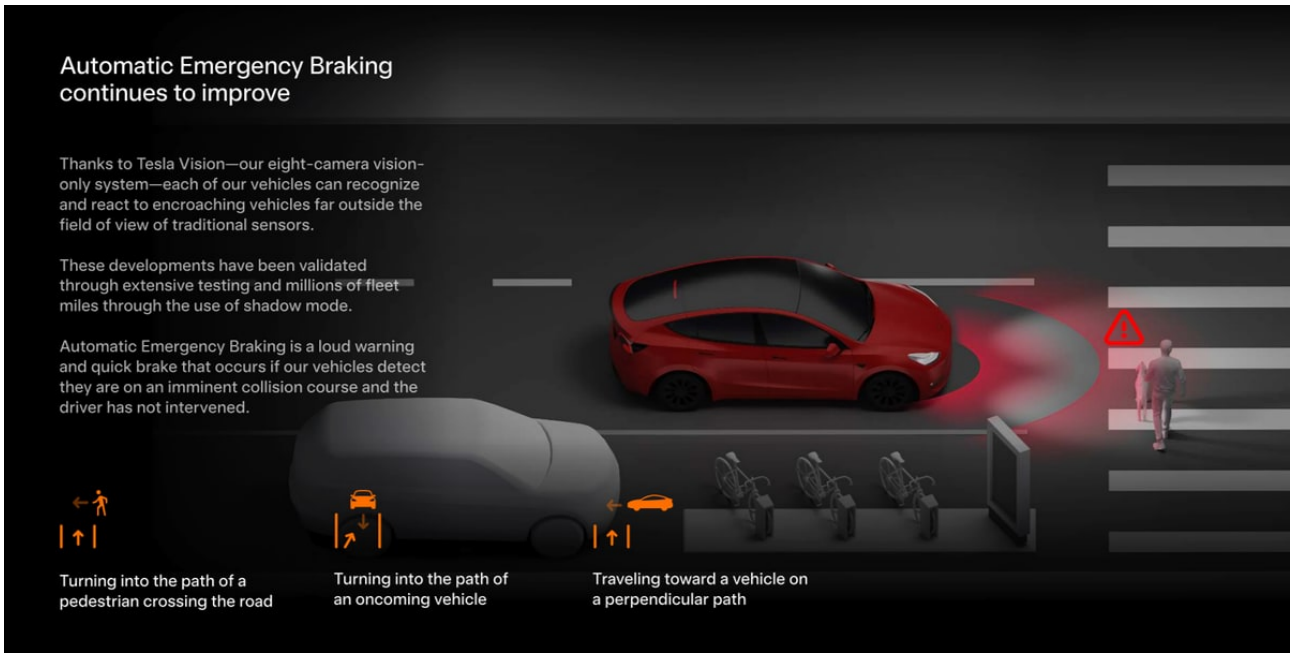
Depending on the quality of your system, AEB systems can work from low speeds to higher speeds. Most won't need a more sophisticated AEB system, as many people don't regularly drive at high speeds. All AEB systems work under the same principles, using a combination of light detection, radar technology and cameras to prevent potential accidents. If an issue is detected, first the AEB system will try to warn the driver and if a response doesn't happen, brakes will be applied automatically.

4.2.2 Types of AEB

There are several different types of automatic emergency braking available in current model vehicles:

- **Low-Speed AEB** — Also called city speed AEB, this type of forward emergency braking works at speeds typically below 55 mph.
- **Highway Speed AEB** — Also called highway AEB, this type of forward emergency braking works at speeds above 55 mph.
- **Rear Automatic Emergency Braking (Rear AEB)** — During reverse maneuvers, Rear AEB senses obstructions and automatically applies the brakes to avoid a collision. Often combined with Rear Cross Traffic Alert[2].
- **Pedestrian AEB** — AEB systems with the capability of Pedestrian Detection[3].

Currently, only low-speed AEB with Forward Collision Warning is included in the NHTSA challenge to automakers.



2 https://caradas.com/rear-cross-traffic-alert-rcta/
3 https://caradas.com/understanding-adas-pedestrian-detection/

4.2.3 AEB requirements

Warning and alert systems features for AEB (Automatic Emergency Braking) systems:

1. **Visual alerts**: AEB systems may provide visual alerts to the driver in the form of warning lights on the dashboard or heads-up displays (HUD). These alerts may be accompanied by a message or symbol that indicates the nature of the potential danger, such as a pedestrian or another vehicle.
2. **Audible alerts**: AEB systems may also provide audible alerts to the driver, such as beeps or chimes. These alerts are often used in conjunction with visual alerts to ensure the driver is aware of the potential danger.
3. **Haptic feedback**: AEB systems may provide haptic feedback to the driver, such as vibrations in the steering wheel or seat. This can be an effective way to get the driver's attention without causing distraction.
4. **Automatic braking**: If the AEB system detects an imminent collision and the driver does not respond to the visual, audible, or haptic alerts, the system may automatically apply the brakes to avoid or mitigate the collision. In some cases, the system may also engage the seatbelt pre-tensioners or other safety features to protect the occupants of the vehicle.

   Alert systems are an integral part of many safety features in modern vehicles, including Automatic Emergency Braking (AEB) systems. These alert systems are designed to provide drivers with clear and effective warnings when a potential danger is detected, helping them to take appropriate action to avoid or mitigate collisions.
   Alert systems for AEB may include visual, audible, and haptic feedback. Visual alerts may be in the form of warning lights or symbols on the dashboard or heads-up display, indicating the nature of the danger detected by the system. Audible alerts may be in the form of beeps, chimes, or spoken warnings that alert the driver to the potential danger. Haptic feedback may involve vibrations in the steering wheel or seat, providing a tactile alert that can help get the driver's attention without causing distraction.
   One important consideration in the design of alert systems is to ensure that they are effective in communicating the nature and severity of the potential danger. For example, a warning light or symbol may be accompanied by a message that provides further information about the nature of the danger, such as "Pedestrian Detected" or "Vehicle in Front." Audible warnings may be adjusted to vary in tone or volume depending on the severity of the situation.
   Another consideration is to ensure that alert systems are not overly intrusive or distracting to the driver. For example, visual alerts should be placed in a location where they can be easily seen without causing the driver to take their eyes off the road for an extended period. Audible alertsshould be loud enough to be heard over other sounds in the vehicle but not so loud as to be jarring or distracting. Haptic feedback should be strong enough to get the driver's attention without causing discomfort or interfering with their ability to control the vehicle.

Integrated safety systems based on these principles can be broadly divided into three categories:

• **Collision avoidance** – Sensors detect 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 other road users

• **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 optimisation 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 optimise restraints.

4.2.3.1 automatic emergency braking system working mechanism:

- The sensors and/or cameras constantly monitor the distance between your car and the obstacle (moving car, pedestrian, etc.) ahead.
- If the distance reduces rapidly, for instance, if the vehicle in front brakes suddenly, the system immediately triggers a warning.
- The driver receives an alert message via an audio or visual medium.
- If you are too late to react, the AEB comes into action and automatically applies the brakes.
- The ECU (Electronic Control Unit) monitors your input and can detect when you are off the throttle and applies the brakes manually. So, AEB will not kick in unnecessarily.
- The Anti-Lock Braking System[4] (ABS) helps AEB stop/slow down the vehicle efficiently.
- The entry-level AEB systems work only at slow speeds. They can be helpful when you drive in the city.
- The more sophisticated automatic braking systems work across a wider speed range. Hence, they may avoid or mitigate the intensity of a high-speed collision.
- The most advanced AEB systems can also detect stationary objects, moving pedestrians, cyclists and cars.

using a microcontroller with a model that has seven states, three output variables, and two manipulated variables.

States

- Lateral velocity
- Yaw rate
- Longitudinal velocity
- Longitudinal acceleration
- Lateral deviation
- Relative yaw angle
- Output disturbance of relative yaw angle

Output Variables

- Longitudinal velocity
- Lateral deviation
- Sum of the yaw angle and yaw angle output disturbance

Manipulated Variables

- Acceleration
- Steering

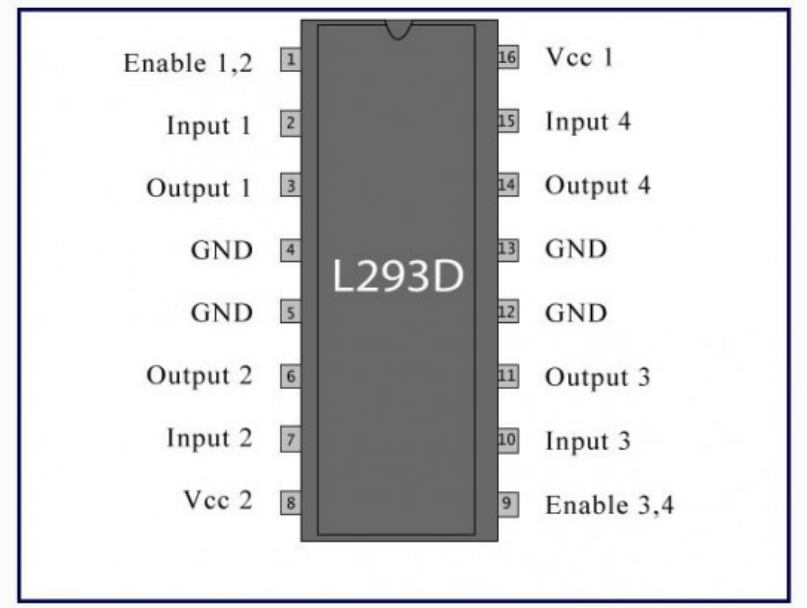
4.2.3.2 Required Tools

- **Hardware Requirement Tools:**

1. STM32f4 micro-controller
2. L293d motor driver IC
   >> L293D is a dual H-bridge motor driver integrated circuit (IC). Motor drivers act as current amplifiers since they take a low-current control signal and provide a higher- current signal. This higher current signal is used

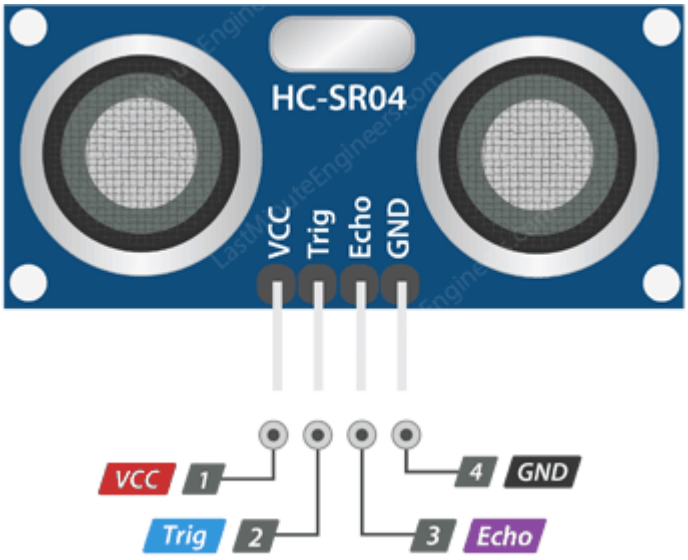
4 https://ackodrive.com/car-guide/anti-lock-braking-system/

to drive the motors.
L293D contains two inbuilt H-bridge driver circuits. In its common mode of operation, two DC motors can be driven simultaneously, both in forward and reverse direction. The motor operations of two motors can be controlled by input logic at pins 2 & 7 and 10 & 15. Input logic 00 or 11 will stop the corresponding motor. Logic 01 and 10 will rotate it in clockwise and anticlockwise directions, respectively.
Enable pins 1 and 9 (corresponding to the two motors) must be high for motors to start operating. When an enable input is high, the associated driver gets enabled. As a result, the outputs become active and work in phase with their inputs. Similarly, when the enable input is low, that driver is disabled, and their outputs are off and in the high-impedance state.



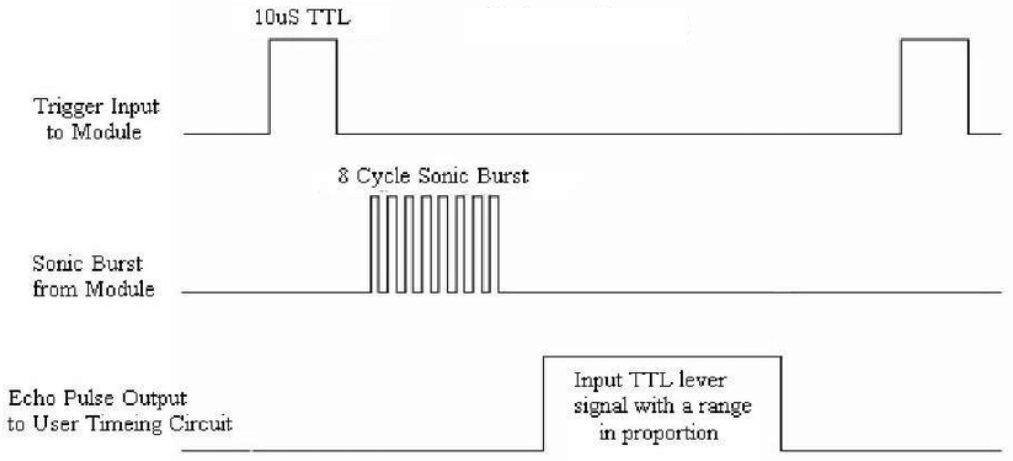
14 L293d Motor Driver IC

3. Two Ultra-Sonic HC-SR04 sensors
>>
Product features:
Ultrasonic ranging module HC-SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:
    a. Using 10 trigger for at least 10us high level signal,
    b. The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
    c. IF the signal back, through high level, time of high output 10 duration is the time from sending ultrasonic to returning.
    Test distance = (high level time velocity of sound (340M/S)/2
    Wire connecting direct as following:
    - 5V Supply
    - Trigger Pulse Input
    - Echo Pulse Output
    - OV Ground



15 HC-SR04 US Sensor

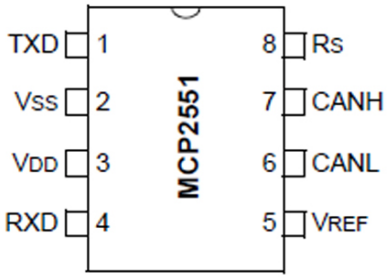
- Timing diagram:
  The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: us / 58 centimeters or us / 148-inch; or: the range = high level time velocity (340M/S)/2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



16 Timing Diagram

4. Two MCP2551 CAN Transceivers
   >>
   2.4 MCP 2551 Transceivers:
   The MCP2551 is a high-speed CAN, fault-tolerant device that serves as the interface between a CAN protocol controller and the physical bus. The MCP2551 device provides differential transmit and receive capability for the CAN protocol controller, and is fully compatible with the ISO-11898 standard, including 24V requirements. It will operate at speeds of up to 1 Mb/s. Typically, each node in a CAN system must have a device to convert the digital signals generated by a CAN controller to signals suitable for transmission over

the bus cabling (differential output). It also provides a buffer between the CAN controller and the high-voltage spikes that can be generated on the CAN bus by outside sources. Some of its features are:
• Supports 1 Mb/s operation
• Implements ISO-11898 standard physical layer requirements
• Suitable for 12V and 24V systems
Detection of ground fault (permanent Dominant) on TXD input
• Power-on Reset and voltage brown-out protection
• An unpowered node or brown-out event will not disturb the CAN bus
Low current standby operation
- Protection against damage due to short-circuit conditions (positive or negative battery voltage)
Up to 112 nodes can be connected
High-noise immunity due to differential bus implementation
- Temperature ranges: - Industrial (1): -40°C to +85°C - Extended (E): -40°C to +125°C



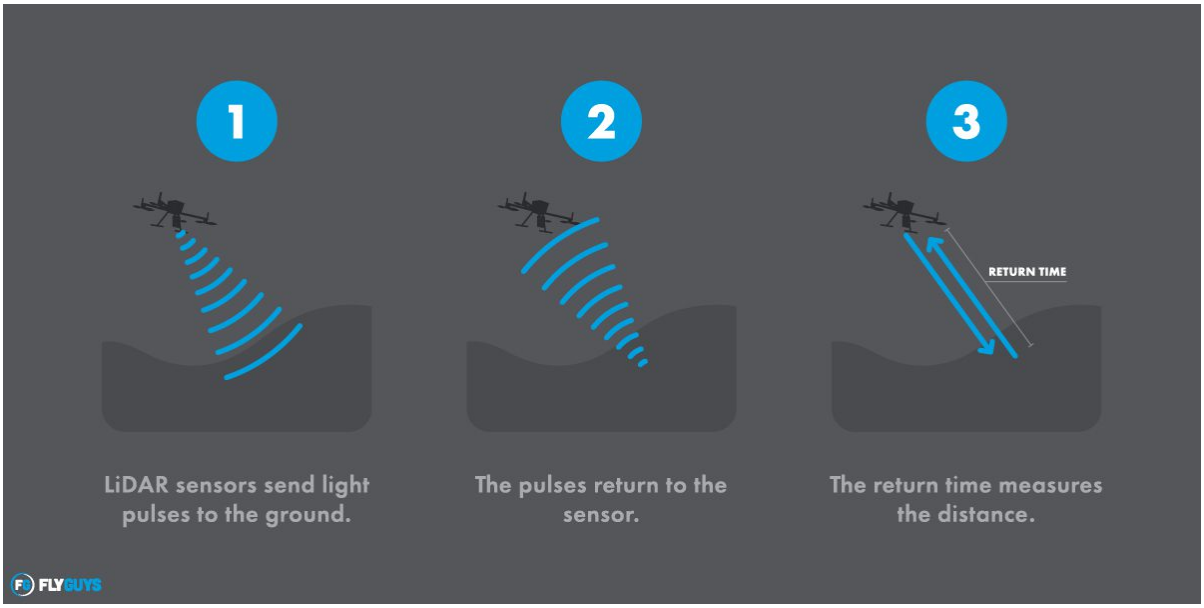
17 MCP2551 CAN Transceiver

5. LCD/GUI
6. DC Motor
   DC motors used in automotive projects can be controlled in a variety of ways :
   a. Voltage control: DC motors can be controlled by varying the voltage applied to the motor terminals. This can be achieved using a simple voltage regulator or a more sophisticated motor control circuit that uses pulse-width modulation (PWM) to adjust the motor speed and direction.
   b. Current control: DC motors can also be controlled by varying the current flowing through the motor windings. This can be achieved using a simple current regulator or a more sophisticated motor control circuit that uses feedback loops to adjust the motor current based on the desired speed and torque.
   c. Encoder feedback: In some applications, DC motors may be equipped with encoders or other sensors that provide feedback on the motor position and speed. This feedback can be used to control the motor more precisely and to implement closed-loop control algorithms that can compensate for variations in load and other factors.
   d. Hybrid systems: In hybrid vehicles, DC motors may be used in conjunction with internal combustion engines or other power sources to provide additional power and torque when needed. In these systems, the motor may be controlled using sophisticated algorithms that balance performance, efficiency, and battery life.

7. Warning Lights and Buzzer and vibration motor
8. [Phase 2 --optional--] LIDAR
   LiDAR is a RaDAR which uses ultraviolet or near infrared light to image objects. Typical automotive LiDARs have wavelengths around 850 or 900 nm. Usually, they are pulsed with power around tens of milliwatts with peak power of pulse up to 80 W. The range of common LiDARs is from 10 to 20 m and they are used to detect obstacles in low speed during urban driving. LiDARs were first applied as sensors for ACC because they were smaller. Nowadays they are often replaced by RaDARs or cameras due to excessive cost and low-resolution capabilities

18 LIDAR Sensor

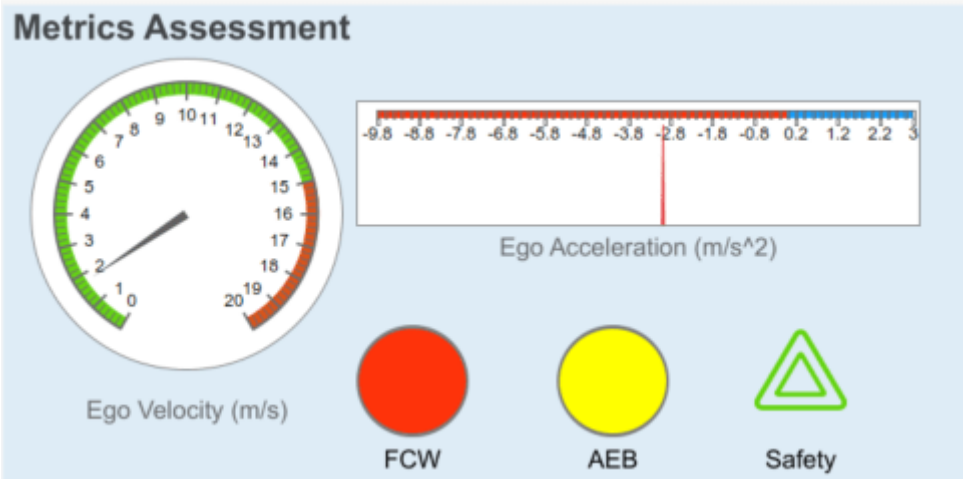
LiDAR sensors have played more and more important role on Intelligent and Connected Vehicles (ICV) and Advanced Driver Assistance Systems (ADAS) .However, the development and testing of LiDAR sensors under real driving environment for ADAS applications are greatly limited by various factors, and often are impossible due to safety concerns. This paper proposed a novel functional LiDAR model under virtual driving environment to support development of LiDAR-based ADAS applications under early stage. Unlike traditional approaches on LiDAR sensor modeling, the proposed method includes both geometrical modeling approach and physical modeling approach. While geometric model mainly produces ideal scanning results based on computer graphics, the physical model further brings physical influences on top of the geometric model. The range detection is derived and optimized based on its physical detection and measurement mechanism.



- **Software Requirements :**
  The software used for implementing the project is Keil uVision and STMCubeID for STM32F4 Board

- GUI Infotainment :

The dashboard panel shows that the AEB system applies cascaded brake to stop the ego vehicle before the collision point. The color of the AEB indicator specifies the level of AEB activation.



19 AEB indicator specifies the level of AEB activation

- Gray — AEB is not activated.
- Yellow — First stage partial brake is activated.
- Orange — Second stage partial brake is activated.
- Red — Full brake is activated.

4.2.4 AEB Design

4.2.4.1 Tools and Technologies Selection:

1. Sensor selection: The AEB system would need to use sensors to detect potential obstacles on the road. Ultrasonic sensors and lidar are both commonly used in AEB systems, with ultrasonic sensors offering a low-cost and low-power alternative to lidar, which is more accurate and provides better range. The specific sensors chosen would depend on the requirements of the application and the trade-offs between accuracy, range, cost, and other factors.
2. Microcontroller selection: The STM32f4 micro-controller is a popular choice for AEB systems, offering a powerful and flexible platform for processing sensor data, controlling motors, and interfacing with other components. The microcontroller would need to be programmed with algorithms to process the sensor data, detect potential obstacles, and activate the brakes when necessary.
3. CAN transceiver: The AEB system would need to communicate with other components in the vehicle, such as the brake system and other safety systems. A CAN transceiver would be used to interface between the micro-controller and the vehicle's CAN bus, enabling the AEB system to send commands and receive feedback fromother components.
4. Motor and motor driver: When the AEB system detects a potential collision, it would need to activate the brakes to prevent the collision or reduce its severity. This would require a motor and motor driver to actuate the brake system. The specific motor and driver chosen would depend on the requirements of the application and the type of brakes used in the vehicle.

4.2.4.2 Case study

Background
The AEB system is designed to improve safety in vehicles by detecting potential collisions and automatically applying the brakes to prevent or mitigate the impact. The system uses sensors and algorithms to detect the distance between the vehicle and other objects in its path and calculate the likelihood of a collision.

Objectives
The objectives of the AEB system are to:

- Improve safety in vehicles by reducing the risk of collisions and minimizing their impact
- Use sensors and algorithms to detect potential collisions and automatically apply the brakes to prevent or mitigate the impact
- Be reliable and responsive in real-world scenarios

Methodology
The AEB system uses a combination of sensors and algorithms to detect potential collisions and automatically apply the brakes to prevent or mitigate the impact. The system consists of two main components: the AEB Electronic Control Unit (ECU) and the Main ECU.

The AEB ECU is responsible for monitoring the sensors and algorithms and triggering the emergency braking system when necessary. The AEB ECU communicates with the Main ECU using a Controller Area Network (CAN) bus to receive sensor data and send control signals.

The Main ECU is responsible for managing the overall operation of the vehicle, including the AEB system. The Main ECU communicates with the vehicle's various sensors and actuators, including the AEB ECU, using a combination of different communication protocols such as CAN,I2C, and UART.

The AEB system uses a variety of sensors to detect potential collisions, including Ultrasonic (US) sensors, which measure the distance between the vehicle and other objects, and an Analog-to-Digital Converter (ADC), which measures the voltage output of the US sensor. The AEB system also uses a motor driver and Pulse Width Modulation

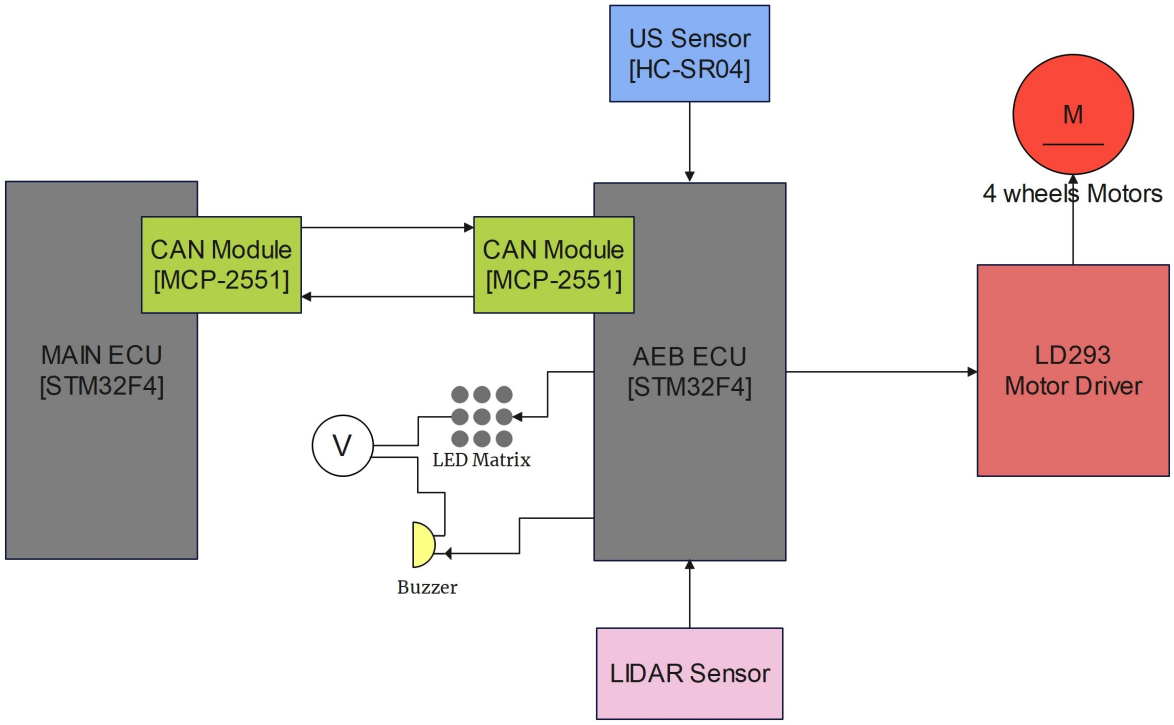
(PWM) module to control the speed and direction of the vehicle and a Light Emitting Diode (LED) and Alarm to alert the driver of potential collisions.

The AEB system operates in two modes: Normal mode and Emergency Braking mode. In Normal mode, the system monitors the sensors and calculates the likelihood of a collision. If a collision is detected, the system switches to Emergency Braking mode and applies the brakes to prevent or mitigate the impact.

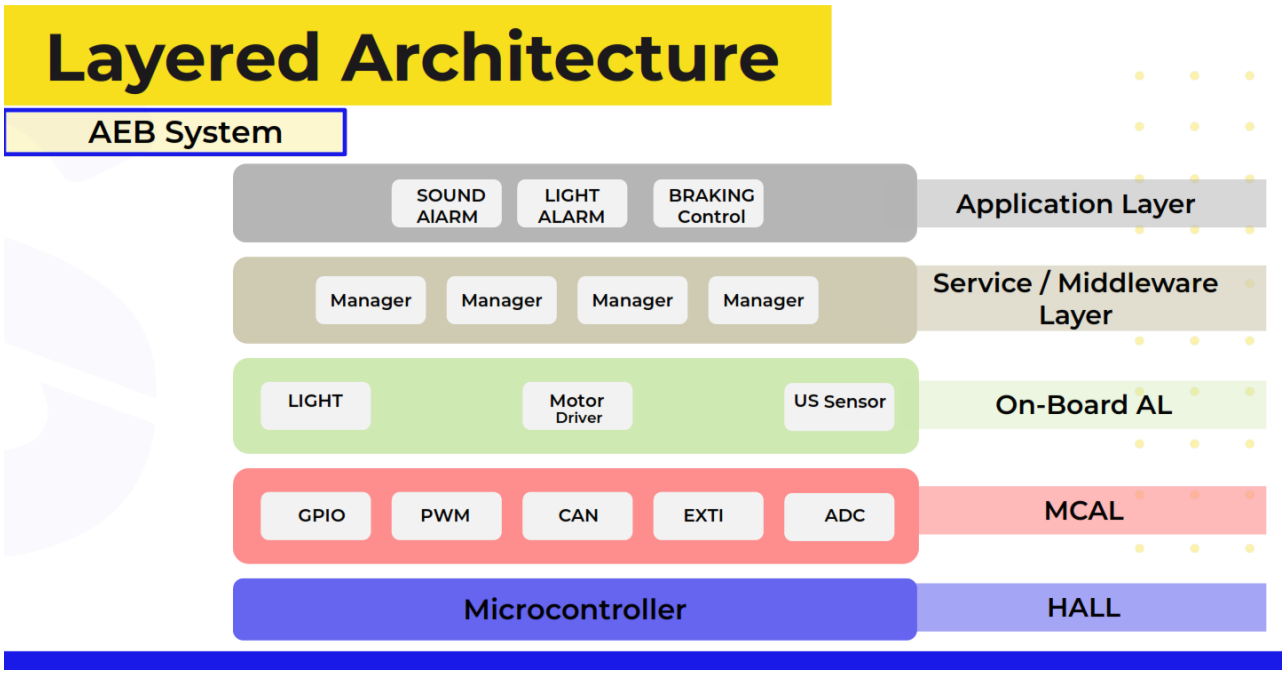
Results
The AEB system was successfully implemented and tested in a real-world scenario using a test vehicle. The system was able to detect potential collisions and trigger the emergency braking system to prevent or mitigate the impact. The system was found to be reliable and responsive in real-world scenarios, improving safety for the driver and passengers.

Conclusion
The Alert and Emergency Braking (AEB) system is an effective solution for improving safety in vehicles by detecting potential collisions and automatically applying the brakes to prevent or mitigate the impact. The system uses a combination of sensors, algorithms, and communication protocols to operate effectively and reliably in real-world scenarios. The AEB system has the potentialto save lives and reduce the risk of accidents, making it an important feature for vehicles of the future.

4.2.4.3 Block Diagram

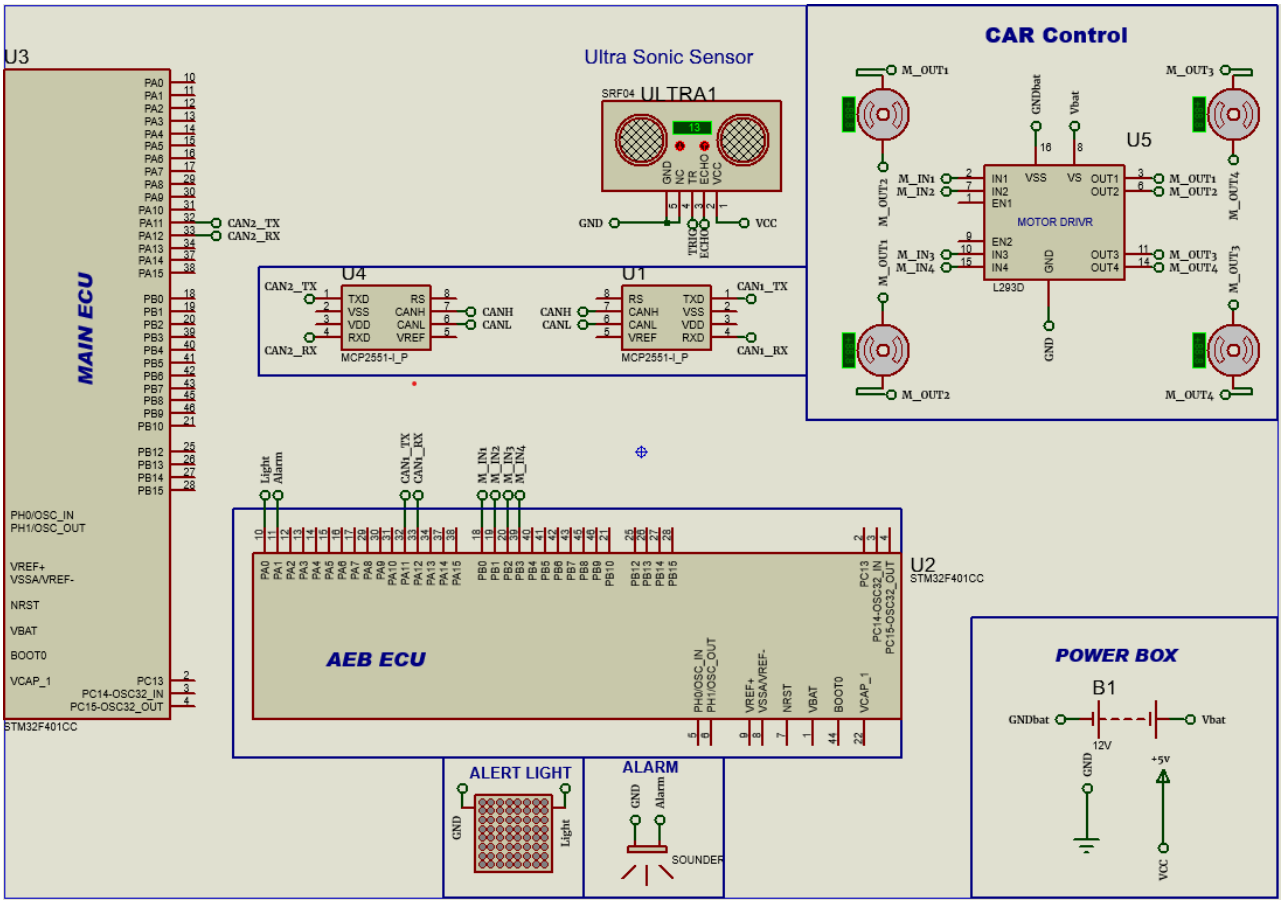


Automatic Emergency Braking system
Block Diagram

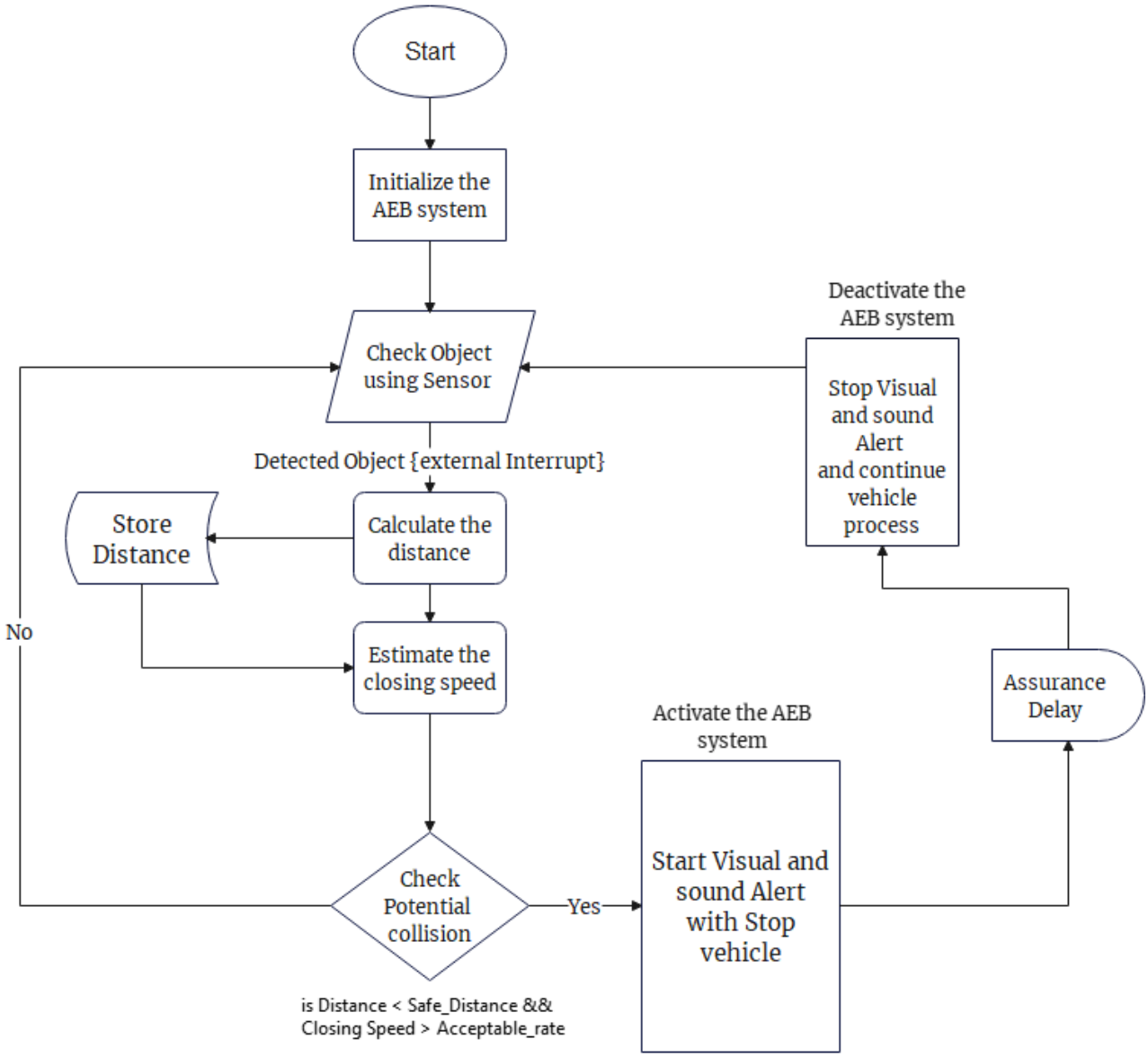

Layered Architecture
AEB System
SOUND AlARM
LIGHT ALARM
BRAKING Control
Application Layer
Manager
Manager
Manager
Manager
Service / Middleware Layer
LIGHT
Motor Driver
US Sensor
On-Board AL
GPIO
PWM
CAN
EXTI
ADC
MCAL
Microcontroller
HALL

4.2.4.4

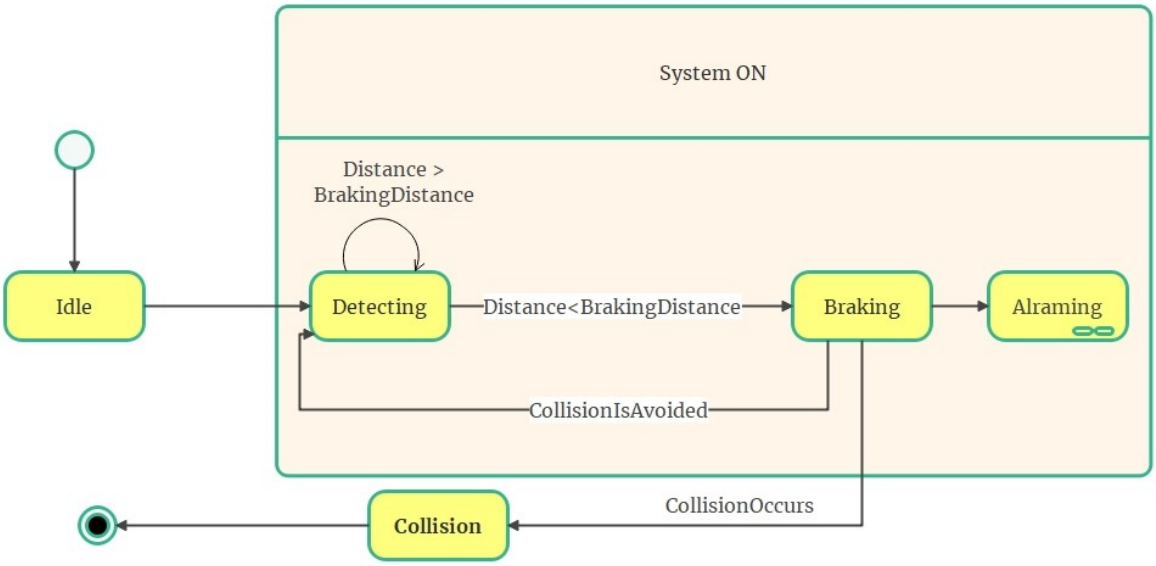
Circuit Diagram



4.2.4.5 Flow Chart [Dynamic Design]



4.2.4.6 State Machine [Dynamic Design]



4.2.4.7 Components and Modules

Alert and Emergency Braking (AEB) system functions, APIs, and data types:

1. AEB ECU:

- CAN module API:

```
void CAN_vInit(CAN_Config_t* CAN_CfgPtr ); // initialize the CAN module
void CAN_vSendMessage(u8 MessageID, u8 Copy_u8Data, u8 Copy_u8Length); // send a CAN
message
CAN_Frame_t*  CAN_xReceiveMessage(void); // receive a CAN message
```

- US sensor API:

```
void US_vInit(); // initialize the US sensor
f32 US_f32MeasureDistance(); // measure the distance to the obstacle
```

- Motor driver API:

```
void Motor_vInit(); // initialize the motor driver
```

```
void Motor_vSetSpeed(int speed); // set the speed of the motors
void Motor_vSetDirection(int direction); // set the direction of the motors
```

- Light API:

```
void Light_vInit(); // initialize the light driver
void Light_vTurnON(); // turn on the brake lights
void Light_vTurnOFF(); // turn off the brake lights
void Light_vSetBrightness(f32 Copy_f32Brightness);
```

- Alarm API:

```
void Alarm_vInit(); // initialize the alarm driver
void Alarm_vStart(); // sound the alarm
void Alarm_vStop(); // stop the alarm
```

- ADC API:

```
void ADC_vInit(ADC_Config_t* ADC_CfgPtr);
u16 ADC_u16GetDigitalVal(ADC_Channel_t Copy_xChannel);
void ADC_vStartConversion(ADC_Channel_t Copy_xChannel);
void ADC_vStopConversion(void);
u16 ADC_u16ReadPort(GPIO_Reg_t* Copy_xGPIO)
f32 ADC_f32ConvertToAnalog(u16 Copy_u16RawVal);
```

- PWM API:

```
void PWM_vInit();
void PWM_SetDutyCycle(f32 Duty_Cycle);
```

- GPIO API:

```
void MCL_GPIO_vInitPort(GPIO_Reg_t* GPIOx, GPIO_Cfg_t* Copy_GPIO_Cfg);
void MCL_GPIO_vSetPinCfg(GPIO_Reg_t* GPIOx, GPIO_Pin_t Copy_PinId, GPIO_Cfg_t*
Copy_GPIO_Cfg);
void MCL_GPIO_vDirectPinSetter(GPIO_Reg_t* GPIOx, GPIO_Pin_t Copy_PinId, Pin_State_t
Copy_xPinVal);
u8  MCL_GPIO_u8GetPinVal(GPIO_Reg_t* GPIOx, GPIO_Pin_t Copy_PinId);
void MCL_GPIO_vWritePort(GPIO_Reg_t* GPIOx, u32 Copy_u32PortVal);
u32 MCL_GPIO_u32ReadPort(GPIO_Reg_t* GPIOx);
void MCL_GPIO_vTogglePin(GPIO_Reg_t* GPIOx, GPIO_Pin_t Copy_PinId);
```

Data types:

```
typedef struct {
    u8 id;
    u8 data[8];
    u8 length;
} CAN_Frame_t;
```

struct and the CAN_vInit() function to initialize the CAN module:

```
// Define the CAN_Config_t struct to hold the CAN configuration parameters
typedef struct {
    u32 baudrate;      // CAN baudrate in bits per second
    u8 sjw;            // Synchronization Jump Width (SJW) in time quanta
    u8 propseg;        // Propagation Segment (PROPSEG) in time quanta
    u8 pseg1;          // Phase Segment 1 (PSEG1) in time quanta
    u8 pseg2;          // Phase Segment 2 (PSEG2) in time quanta
    u8 prescaler;      // CAN clock prescaler
    u8 mode;           // CAN operating mode (normal or loopback)
    u8 interrupt_en;   // Enable or disable CAN interrupts
} CAN_Config_t;


typedef struct
{

    GPIO_Mode_t GPIO_Mode;          //mode of GPIO selected port/pin
    //this parameter can be a value of  @ref GPIO_Mode_define

    u8 GPIO_Speed;          //speed of GPIO selected port/pin
    //this parameter can be a value of  @ref GPIO_Speed_define

}GPIO_Cfg_t;

//@ref ADC_Channels_Define
//select ADC Channel to receive from
typedef enum{
    //  Bits 4:0 - MUX4:0: Analog Channel and Gain Selection Bits
    ADC_CHANNEL0,
    ADC_CHANNEL1,
    ADC_CHANNEL2,
    ADC_CHANNEL3,
    ADC_CHANNEL4,
    ADC_CHANNEL5,
    ADC_CHANNEL6,
    ADC_CHANNEL7,
    ADC_CHANNEL8,
    ADC_CHANNEL9,
}ADC_Channel_t;

typedef struct {
    ADC_Channel_t channel;      // ADC channel to use
    u8 resolution;   // ADC resolution (e.g. 8-bit, 10-bit, etc.)
    u8 clock_source; // ADC clock source (e.g. internal, external, etc.)
```

```
    u32 Sampling_Time; // ADC sampling time
} ADC_Config_t;

typedef float distance_t;
typedef int speed_t;
typedef int direction_t;
typedef bool bool_t;
typedef int raw_adc_value_t;
typedef float voltage_t;
typedef float duty_cycle_t;
typedef float brightness_t;
```

2. Main ECU:

- CAN module API:

```
void CAN_vInit(CAN_Config_t* CAN_CfgPtr ); // initialize the CAN module
void CAN_vSendMessage(u8 MessageID, u8 Copy_u8Data, u8 Copy_u8Length); // send a CAN
message
CAN_Frame_t*  CAN_xReceiveMessage(void); // receive a CAN message
```

- AEB ECU API:

```
void AEB_vInit(); // initialize the AEB ECU
void AEB_vTriggerEB(); // trigger emergency braking
```

- User interface (UI) API:

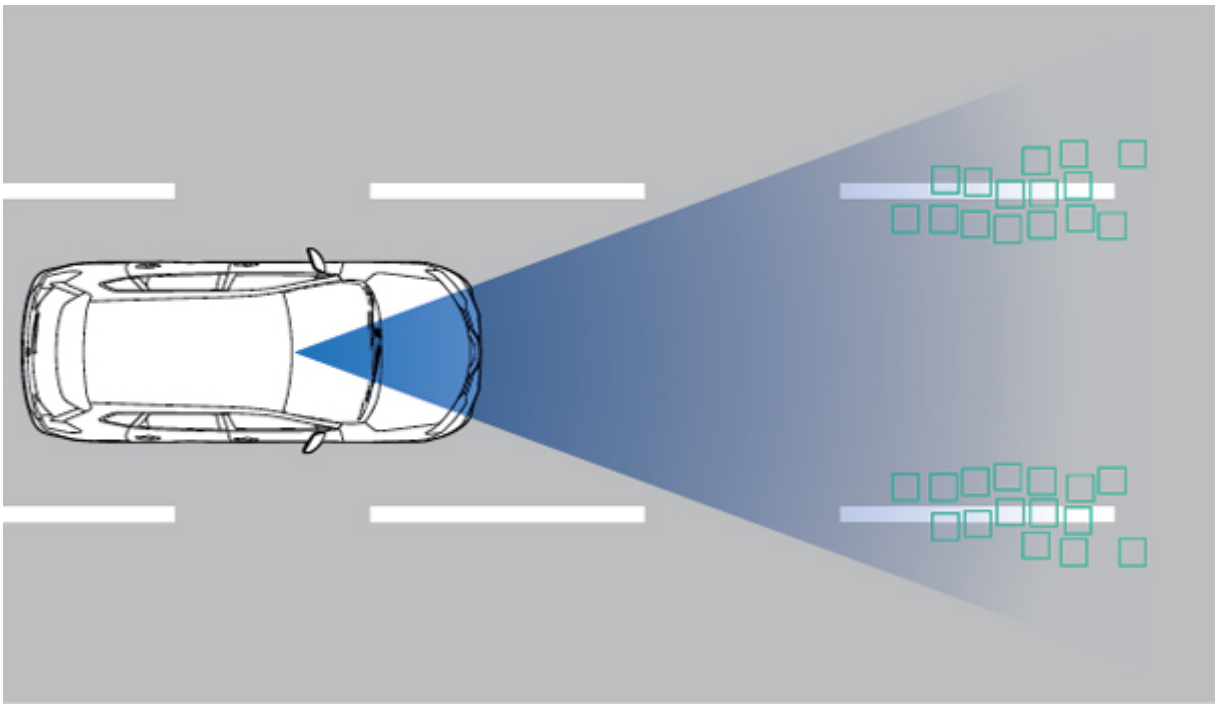
```
void GUI_vInit(); // initialize the user interface
void GUI_vDisplayStatus(status_t status); // display the current status of the AEB
system
void GUI_vConfigSettings(configuration_t configuration); // allow the user to
configure the AEB settings
```

Data types:

```
typedef enum {
    STATUS_ACTIVE,
    STATUS_INACTIVE,
    STATUS_TRIGGERED
} status_t;

typedef struct {
    int sensitivity;
    int response_time;
} configuration_t;

typedef enum {
    ERROR_NONE,
```

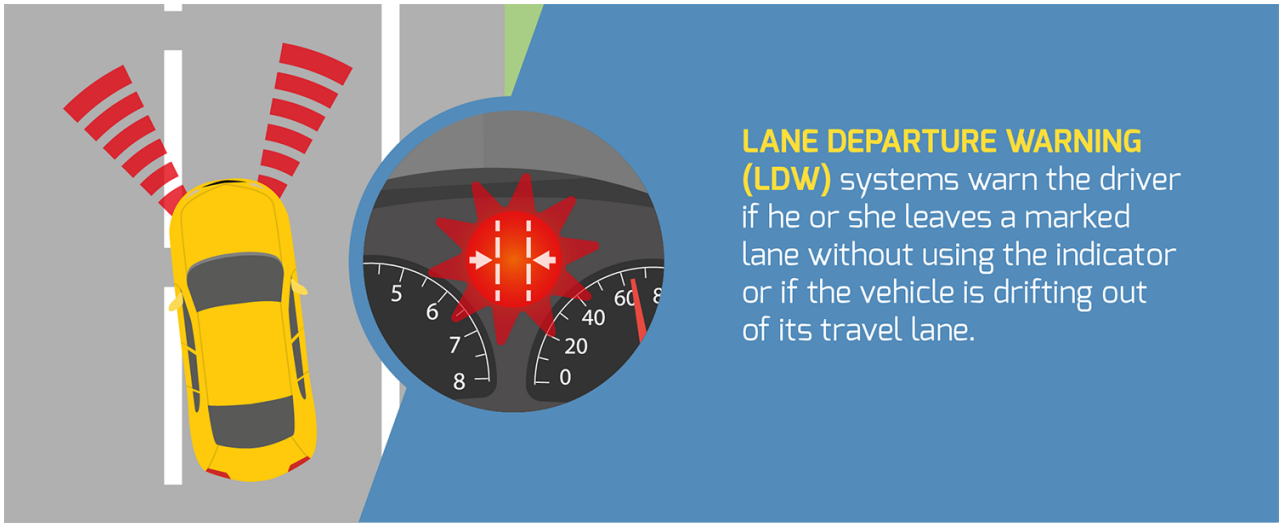
```
    ERROR_CAN,
    ERROR_US,
    ERROR_ADC,
    ERROR_MOTOR,
    ERROR_PWM,
    ERROR_LIGHT,
    ERROR_ALARM
} error_t;
```

4.3 Lane Departure Warning (LDW)



Introduction:
Lane Departure Warning System (LDWS) is a safety feature that warns drivers when their vehicle crosses the lane markings without signaling. The system uses sensors to detect the position of the vehicle within the lane boundaries, and if it detects a potential lane departure, it alerts the driver through audible and/or visual warnings. One of the common types of sensors used in LDWS is the Infrared (IR) sensor.
IR Sensor-Based LDWS:
IR sensors are used in LDWS as they can detect the lane markings on the road surface by analyzing the infrared radiation reflected from the surface. The IR sensor-based LDWS consists of an IR sensor, signal processing unit, and warning system.
Working Principle:
The IR sensor-based LDWS works based on the principle of reflection of IR radiation. The sensor emits a beam of IR radiation towards the road surface, and the radiation is reflected back by the lane markings. The reflected radiation is then detected by the sensor and converted into an electrical signal, which is processed by the signal processing unit. The signal processing unit analyzes the signal and determines the position of the vehicle within the lane boundaries. If the vehicle is about to cross the lane markings, the warning system is activated to alert the driver.
Warning System:
The warning system in the IR sensor-based LDWS can be either audible or visual or both. The audible warning can be in the form of a beep sound, while the visual warning can be in the form of a flashing light or a message

displayed onthe dashboard. The warning system is designed to grab the driver's attention and alert them to take corrective action to prevent a potential accident.



Advantages of IR Sensor-based LDWS:
1-Accuracy: The IR sensor-based LDWS provides accurate lane detection, as it relies on the reflection of IR radiation from the lane markings, which is not affected by environmental factors like light, rain, or snow.
2-Cost-effective: IR sensors are relatively cheaper compared to other sensors used in LDWS, making it an affordable option for mass production in vehicles.
3-Easy installation: The IR sensor-based LDWS is easy to install and can be integrated into the vehicle's existing system without major modifications.
4-Improved safety: The IR sensor-based LDWS improves safety by alerting the driver when the vehicle deviates from the lane markings, reducing the risk of accidents caused by distracted or drowsy driving.
Limitations of IR Sensor-based LDWS:
1-Limited range: The IR sensor-based LDWS has a limited range and can only detect lane markings within a few meters in front of the vehicle.
2-Limited functionality: The IR sensor-based LDWS can only detect lane markings and cannot provide information about other road features like speed limits, traffic signs, or other vehicles.
3-Weather conditions: In extreme weather conditions like heavy rain, snow, or fog, the IR sensor-based LDWS may not function correctly, as the reflection of IR radiation from the road surface may be affected.
conclusion:
The IR sensor-based LDWS is an effective and affordable safety feature that can be integrated into vehicles to improve road safety. The system works based on the reflection of IR radiation from the lane markings and provides accurate lane detection, easy installation, and improved safety. However, the system has limitations like limited range, functionality, and susceptibility to extreme weather conditions. Overall, the IR sensor-based LDWS is a valuable addition to modern vehicles, and its effectiveness can be enhanced by integrating it with other advanced driver assistance systems.

4.3.1 LDW Requirements

Hardware Requirements:

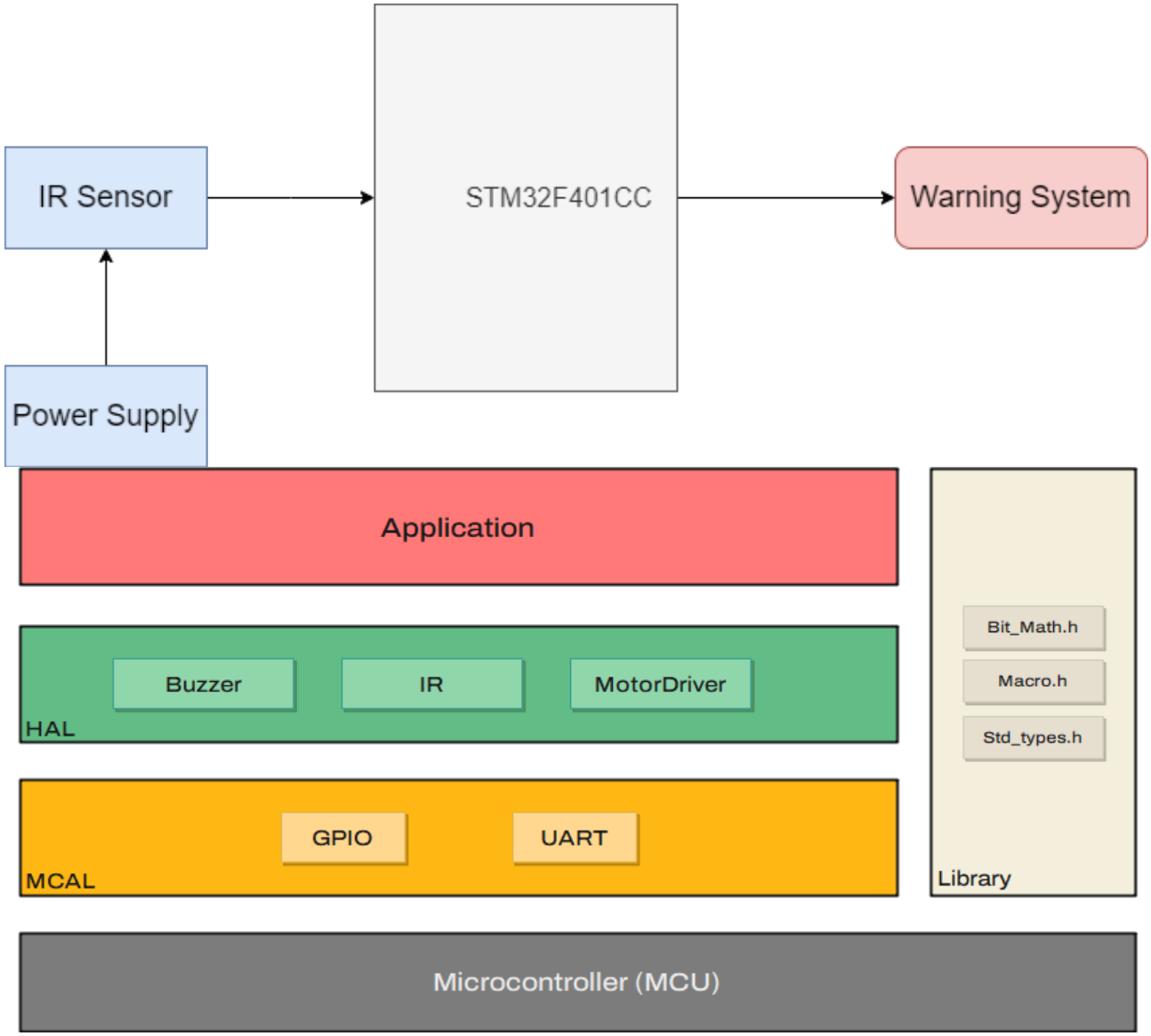
1. IR Sensors: IR sensors are used to measure the distance between the car and the edge of the road and determine its position on the road. IR sensors available in the market, such as TCRT5000 and GP2Y0A02YK0F sensors can be used.

2. Microcontroller: A high-performance microcontroller is required to process the sensor signals and determine if the car has departed from the lane. STM32F401CC microcontroller can be used for this purpose.
3. Warning System: A warning system is used to notify the driver when the car departs from the lane. A LED indicator or buzzer can be used to issue warnings.
4. Power Supply: The system requires a power source to operate the components. A rechargeable battery or an external power supply can be used.

Software Requirements:

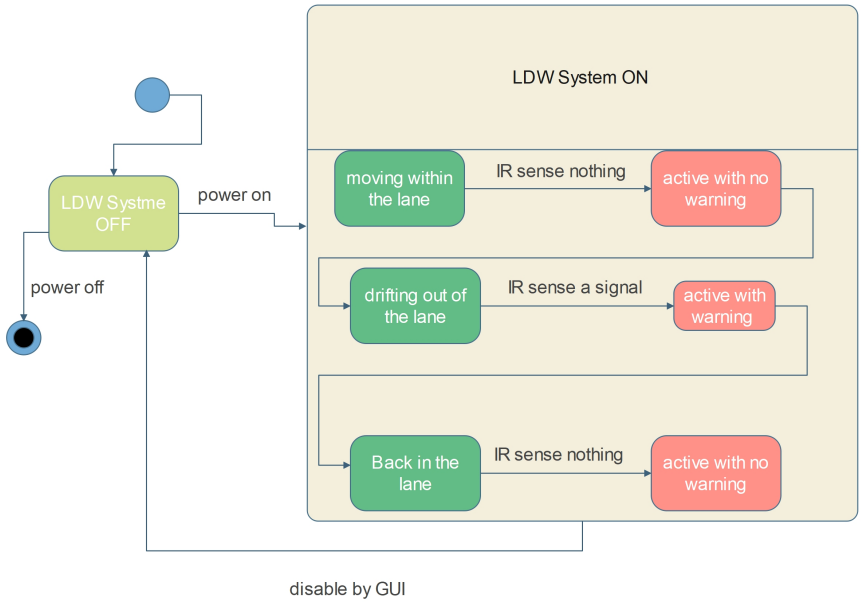
1. C Programming Language: C programming language is used to develop the system software.
2. Integrated Development Environment (IDE): IDEs such as Keil uVision and STM32CubeIDE can be used to develop the system software using the C programming language and upload it to the microcontroller.

4.3.2 LDW Design



20 Figure - Layered Architecture

21 LDW Flowchart



22 LDW State machine

4.3.2.1 APIs and Typedefs

1.2. GPIO

1.2.1.1 APIs

Name	GPIO_u8SetPinDir
Syntax	RequestState GPIO_u8SetPinDir(u8 Local_GroupName, u8 Local_PinNumber, u8 Local_PinState);
Description	Set the direction of the GPIO pins
Sync/Async	Synchronous
Reentrancy	Non-Reentrant

Parameters(in)	Local_GroupName, Local_PinNumber, Local_PinState
Parameters(out)	None
Return value	RequestState
Name	GPIO_u8SetPinValue
Syntax	RequestState GPIO_u8SetPinValue(u8 Local_GroupName, u8 Local_PinNumber, u8 Local_PinValue);
Description	Sets the value of GPIO pins
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	Local_GroupName, Local_PinNumber, Local_PinValue
Parameters(out)	None
Return value	RequestState
Name	GPIO_u8GetPinValue
Syntax	RequestState GPIO_u8GetPinValue(u8 Local_GroupName, u8 Local_PinNumber, u8 *Local_PinValue);
Description	Reads the value of the GPIO pins
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	Local_GroupName, Local_PinNumber, Local_PinValue
Parameters(out)	None
Return value	RequestState
Name	GPIO_u8SetPortDir

Syntax	RequestState GPIO_u8SetPortDir(u8 Local_GroupName, u8 Local_PortState);
Description	Sets the direction of the port
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	Local_GroupName, Local_PortState
Parameters(out)	None
Return value	RequestState
Name	GPIO_u8SetPortValue
Syntax	RequestState GPIO_u8SetPortValue(u8 Local_GroupName, u8 Local_PortValue);
Description	Sets the PORT value with a specific value
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	Local_GroupName, Local_PortValue
Parameters(out)	None
Return value	RequestState
Name	GPIO_u8GetPortValue
Syntax	RequestState GPIO_u8GetPortValue(u8 Local_GroupName, u8 *Local_PortValue);
Description	Reads the value of the port
Sync/Async	Synchronous
Reentrancy	Non-Reentrant

Parameters(in)	Local_GroupName, Local_PortValue
Parameters(out)	None
Return value	RequestState
Name	GPIO_u8ControlPullup
Syntax	RequestState GPIO_u8ControlPullup(u8 Local_ConnectionType, u8 Local_GroupName, u8 Local_PinNumber, u8 Local_PullupState);
Description	Enable or Disable PullUp resistor of the PIN
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	Local_ConnectionType, Local_GroupName, Local_PinNumber, Local_PullupState
Parameters(out)	None
Return value	RequestState

1.2.1.2 Typedefs

Name	RequestState
Type	enum
Description	An enum for handling Errors: RequestHandled, RequestErrorGroupOutOfRange, RequestErrorPinOutOfRange, RequestErrorNotValidState, and RequestErrorNotValidValue

1.2.2 UART

1.2.2.1 APIs

Name	UART_VoidInit
Syntax	void UART_VoidInit(u16 Baudrate);
Description	Initializes UART module with the desired configurations

Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	Baudrate
Parameters(out)	None
Return value	void
Name	UART_TxChar
Syntax	void UART_TxChar(u8 local_data);
Description	Sends a char to the receiver
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	local_data
Parameters(out)	None
Return value	void
Name	UART_TxString
Syntax	void UART_TxString(u8 *string);
Description	Sends a string to the receiver
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	string
Parameters(out)	None

Return value	void

Name	UART_RxChar
Syntax	u8 UART_RxChar(void);
Description	Receives a char from the transmitter
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	u8
Name	UART_RxString
Syntax	void UART_RxString(char* string);
Description	Receives a string from the transmitter
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	string
Parameters(out)	None
Return value	void

1.2.3 MotorDriver

1.2.3.1 APIs

Name	MDriver_init

Syntax	void MDriver_init(void);
Description	Initializes Motor Driver module
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	void

Name	MDriver_MoveBackward
Syntax	void MDriver_MoveBackward(void);
Description	Moves the car backward
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	void

Name	MDriver_MoveRight
Syntax	void MDriver_MoveRight(void);
Description	Moves the car to the right
Sync/Async	Synchronous

Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	void

Name	MDriver_MoveLeft
Syntax	void MDriver_MoveLeft(void);
Description	Moves the car to the left
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	void

Name	MDriver_MoveForRight
Syntax	void MDriver_MoveForRight(void);
Description	Moves the car forward right
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None

Return value	void

Name	MDriver_MoveForLeft
Syntax	void MDriver_MoveForLeft(void);
Description	Moves the car forward left
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	void

Name	MDriver_MoveBackRight
Syntax	void MDriver_MoveBackRight(void);
Description	Moves the car backward right
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	void

Name	MDriver_MoveBackLeft
Syntax	void MDriver_MoveBackLeft(void);

Description	Moves the car backward left
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	void

Name	MDriver_Stop
Syntax	void MDriver_Stop(void);
Description	Stops the car
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	void

1.2.4 IR

1.2.4.1 APIs

Name	IRSensors_init
Syntax	void IRSensors_init(void);
Description	Initializes IR Sensors
Sync/Async	Synchronous

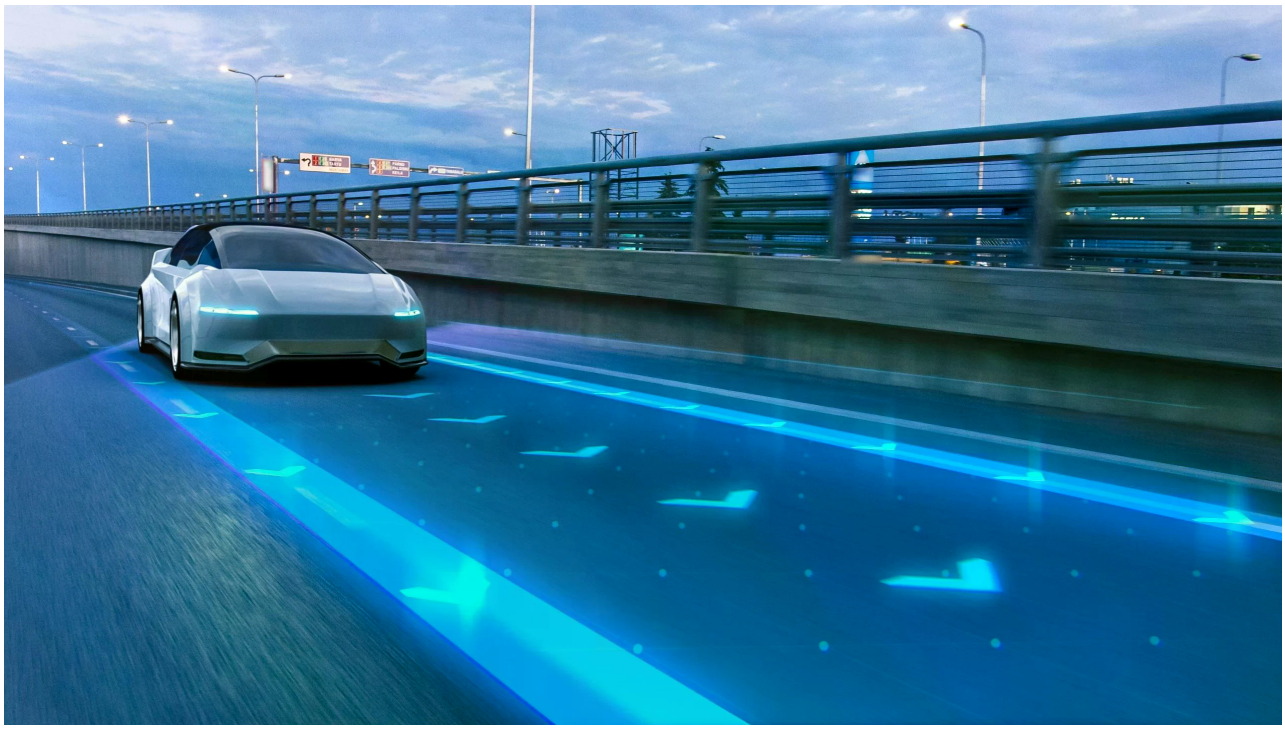
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	void

Name	Read_IRSensor_Right
Syntax	u8 Read_IRSensor_Right(void);
Description	Reads the value of the Right IR Sensor
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	u8

Name	Read_IRSensor_Left
Syntax	u8 Read_IRSensor_Left(void);
Description	Reads the value of the Left IR Sensor
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None

Return value	u8

4.4 Lane Keeping Assist System (LKAS)



4.4.1 Lane keeping assist (LKA) is an advanced driver assistance system (ADAS) that helps drivers to stay in their lane and avoid unintentional drifting. LKA technology uses a combination of sensors and cameras to detect the vehicle's position on the road, and if it detects that the vehicle is drifting out of its lane, it can provide various types of assistance to help the driver stay on course.

LKA technology can take several actions to help keep the vehicle in its lane, including:

- Providing visual or audible warnings to the driver if the vehicle begins to drift out of its lane
- Applying slight steering corrections to the vehicle to bring it back into the lane
- Providing haptic feedback to the steering wheel to alert the driver to the vehicle's position

Some LKA systems can also work together with other ADAS features, such as adaptive cruise control (ACC), to provide even more advanced assistance. In these systems, the vehicle's speed is automatically adjusted to maintain a safe following distance with the vehicle ahead, while also staying within its own lane.

The benefits of LKA technology include increased safety, reduced driver fatigue, improved comfort, and increased convenience. By helping to prevent unintentional lane departures, LKA technology can reduce the risk of accidents caused by driver error. LKA technology can also help reduce driver fatigue by providing an additional level of support and assistance, particularly during long drives or in heavy traffic. Additionally, LKA technology can make driving easier and more convenient, particularly in situations where the driver may be distracted or fatigued.

Overall, lane-keeping assist is an important driver assistance system that can provide a range of benefits to drivers. As ADAS technology continues to evolve, we can expect to see even more advanced and sophisticated LKA systems in the future.

4.4.2 LKS Requirements

1. Lane Detection:
    - The system should be able to detect the position of the vehicle within the lane using two IR sensors mounted on the front of the car chassis.
    - The IR sensors should provide reliable and accurate distance measurements from the lane boundaries.
    - The system should be able to differentiate between different types of lane markings, such as solid lines, dashed lines, or other lane boundary indicators.
2. Drift Detection:
    - The system should continuously monitor the position of the vehicle within the lane and detect any instances of unintentional drifting.
    - The detection algorithm should be able to differentiate between intentional lane changes and unintentional drifting.
3. Warning System:
    - The system should provide visual and audible warnings to the driver when the vehicle begins to drift out of its lane.
    - The warnings should be clear, easily noticeable, and customizable to suit different driver preferences.
    - The warning system should activate in a timely manner to allow the driver sufficient time to react and correct the drift.
4. Steering Correction:
    - The system should be able to apply slight steering corrections to the vehicle to bring it back into the lane when it detects drifting.
    - The steering corrections should be smooth, gradual, and proportional to the degree of drift.
    - The system should ensure that the steering corrections do not cause abrupt or erratic movements of the vehicle.
5. Haptic Feedback:
    - The system should provide haptic feedback to the steering wheel to alert the driver to the vehicle's position within the lane.
    - The haptic feedback should be intuitive and distinguishable, providing a tactile indication of the vehicle's deviation from the center of the lane.
6. Integration with Motor Control:
    - The system should interface with the DC motors and the motor driver to control the steering of the vehicle.
    - The motor control should be precise, responsive, and capable of generating the required steering corrections based on the input from the lane detection system.
    - The system should coordinate the motor control with the lane detection and drift detection algorithms to ensure accurate and timely steering corrections.
7. Reliability and Safety:
    - The system should operate reliably under various driving conditions, such as different road surfaces, lighting conditions, and weather conditions.
    - The system should incorporate fail-safe mechanisms to handle sensor failures or system malfunctions, ensuring the safe operation of the vehicle.
    - The system should comply with applicable safety standards and regulations for ADAS systems.
8. Performance:
    - The system should provide real-time performance with minimal latency to ensure quick and accurate responses to lane deviations.

- The system should be capable of processing sensor data, executing algorithms, and generating steering corrections within acceptable time limits.

9. Flexibility and Adaptability:
    - The system should be easily configurable and adaptable to different vehicle models and configurations.
    - The system should allow for customization of parameters and settings to accommodate driver preferences and varying road conditions.
10. Documentation and User Interface:
    - The system should be accompanied by comprehensive documentation, including user manuals and technical specifications.
    - The system should have a user-friendly interface for configuring settings, monitoring system status, and accessing diagnostic information.

Note: The requirements listed above serve as a general guideline. For a complete and accurate set of requirements, it is advisable to consult with domain experts and consider specific design constraints and objectives.

4.4.2.1 Hardware Requirements:

1. STM32F401 MCU:
    - The system should utilize the STM32F401 microcontroller unit (MCU) as the primary processing unit.
    - The MCU should have sufficient processing power, memory, and peripherals to handle sensor data processing, algorithm execution, and motor control.
2. IR Sensors:
    - The system should include two infrared (IR) sensors for lane detection.
    - The IR sensors should be capable of providing accurate distance measurements and detecting lane boundaries reliably.
    - The sensors should have a suitable field of view and range to cover the width of the lane effectively.
3. 4WD Car Chassis:
    - The system should be implemented on a 4-wheel drive (4WD) car chassis.
    - The chassis should be sturdy, capable of accommodating the MCU, sensors, motor driver, motors, and associated wiring securely.
4. DC Motors:
    - The system should incorporate DC motors to control the steering of the vehicle.
    - The motors should provide sufficient torque and speed for accurate steering corrections.
    - The selection of DC motors should consider the weight and size of the car chassis and the desired steering responsiveness.
5. Motor Driver:
    - The system should utilize a suitable motor driver to interface the MCU with the DC motors.
    - The motor driver should support the required motor voltage and current specifications.
    - It should provide efficient and smooth motor control, allowing precise steering corrections.
6. Power Supply:
    - The system should include a reliable power supply to ensure uninterrupted operation.
    - The power supply should be capable of providing sufficient voltage and current for all system components.
    - It should incorporate safeguards against power surges, voltage fluctuations, and other electrical issues.

4.4.2.2 Software Requirements:

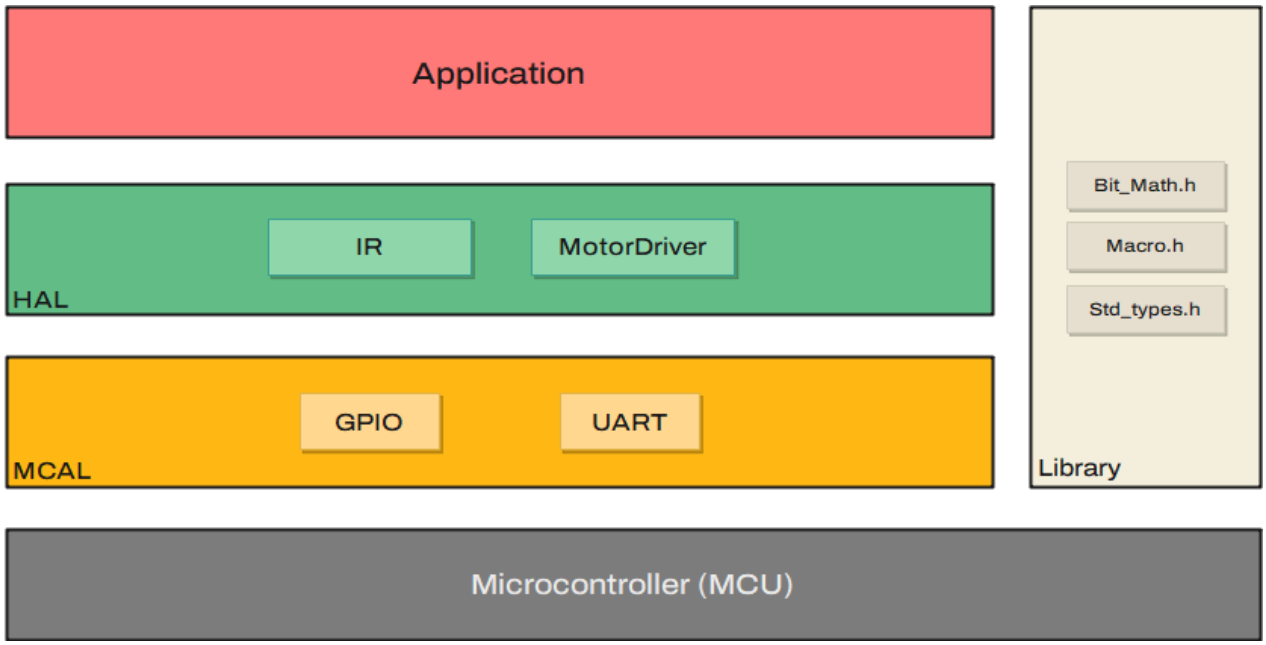
1. Firmware Development:
    - The system firmware should be developed using a suitable programming language, such as C or C++.

- The firmware should be optimized for real-time operation, ensuring timely execution of algorithms and control tasks.
- It should leverage the STM32F401 MCU's capabilities, including its peripherals, timers, and interrupt handling mechanisms.

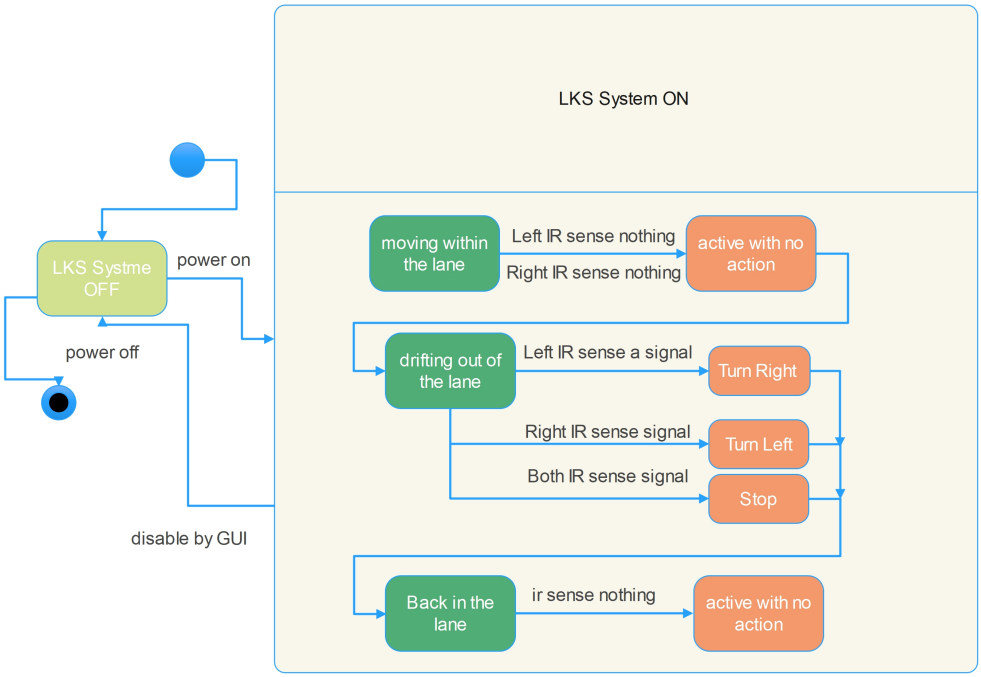
2. Sensor Data Processing:
   - The firmware should include algorithms for processing the data from the IR sensors.
   - The algorithms should convert raw sensor data into meaningful distance measurements and lane position information.
   - Sensor fusion techniques, such as filtering or data fusion, may be employed to improve the accuracy and reliability of the lane detection.
3. Control Algorithm:
   - The firmware should implement a control algorithm to determine the required steering corrections based on the detected lane position.
   - The algorithm should consider factors like vehicle speed, lane width, and desired sensitivity to provide appropriate and smooth steering adjustments.
4. Communication:
   - If required, the firmware should support communication protocols, such as UART, SPI, or I2C, to interface with external devices or modules.
   - Communication may be necessary for diagnostics, configuration, or integration with other ADAS features.

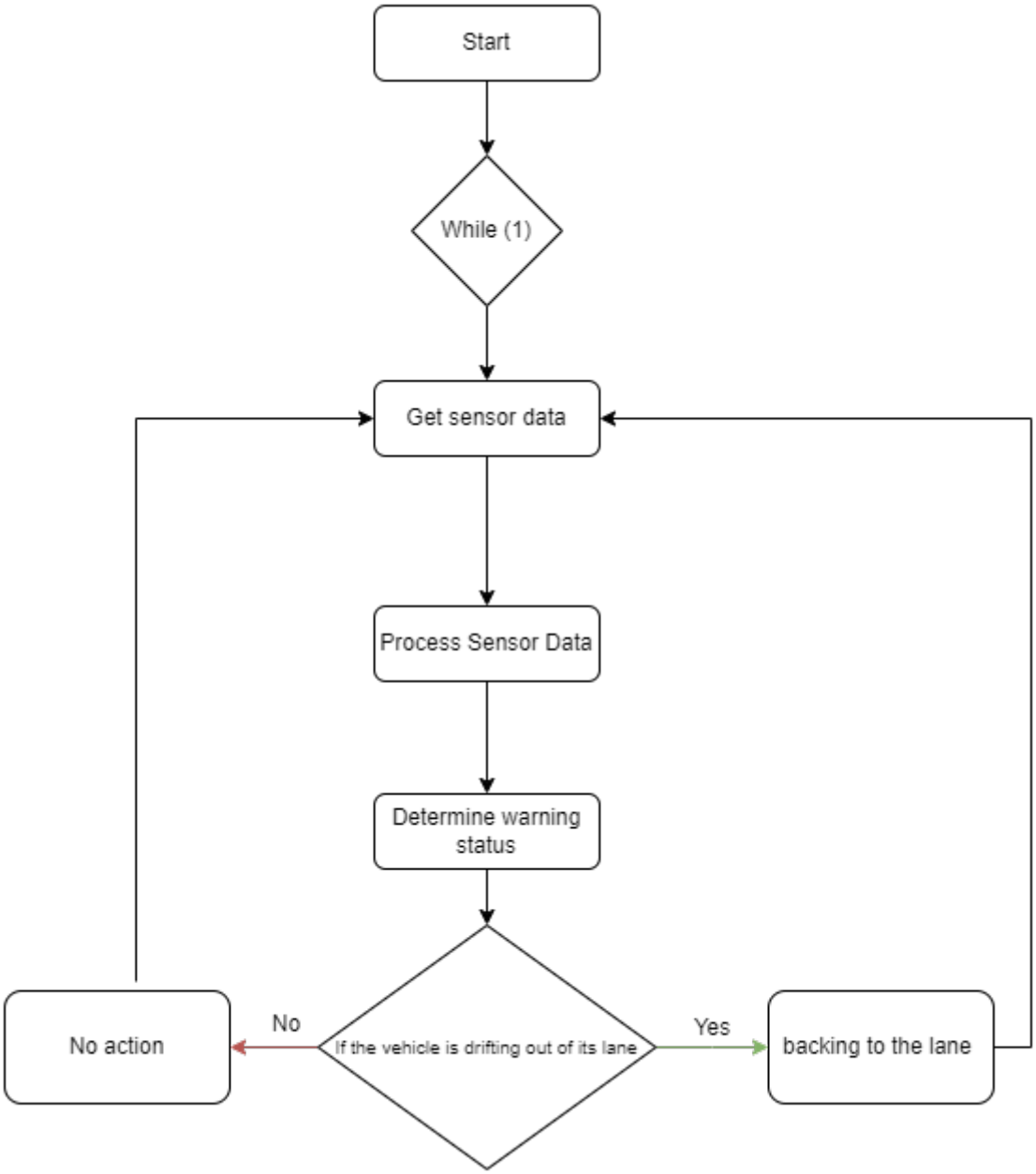
4.4.3 LKS Design

4.4.3.1 2. Layered Architecture



23 Figure 2 - Layered Architecture of LKS


LKS System ON
LKS Systme OFF
power on
power off
disable by GUI
moving within the lane
Left IR sense nothing
Right IR sense nothing
active with no action
drifting out of the lane
Left IR sense a signal
Turn Right
Right IR sense signal
Turn Left
Both IR sense signal
Stop
Back in the lane
ir sense nothing
active with no action



24 LKS Flowchart

4.4.3.2 3. APIs and Typedefs

1.2. GPIO

1.2.1.1 APIs

Name	GPIO_u8SetPinDir

Syntax	RequestState GPIO_u8SetPinDir(u8 Local_GroupName, u8 Local_PinNumber, u8 Local_PinState);
Description	Set the direction of the GPIO pins
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	Local_GroupName, Local_PinNumber, Local_PinState
Parameters(out)	None
Return value	RequestState

Name	GPIO_u8SetPinValue
Syntax	RequestState GPIO_u8SetPinValue(u8 Local_GroupName, u8 Local_PinNumber, u8 Local_PinValue);
Description	Sets the value of GPIO pins
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	Local_GroupName, Local_PinNumber, Local_PinValue
Parameters(out)	None
Return value	RequestState

Name	GPIO_u8GetPinValue
Syntax	RequestState GPIO_u8GetPinValue(u8 Local_GroupName, u8 Local_PinNumber, u8 *Local_PinValue);
Description	Reads the value of the GPIO pins

Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	Local_GroupName, Local_PinNumber, Local_PinValue
Parameters(out)	None
Return value	RequestState

Name	GPIO_u8SetPortDir
Syntax	RequestState GPIO_u8SetPortDir(u8 Local_GroupName, u8 Local_PortState);
Description	Sets the direction of the port
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	Local_GroupName, Local_PortState
Parameters(out)	None
Return value	RequestState
Name	GPIO_u8SetPortValue
Syntax	RequestState GPIO_u8SetPortValue(u8 Local_GroupName, u8 Local_PortValue);
Description	Sets the PORT value with a specific value
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	Local_GroupName, Local_PortValue
Parameters(out)	None

Return value	RequestState
Name	GPIO_u8GetPortValue
Syntax	RequestState GPIO_u8GetPortValue(u8 Local_GroupName, u8 *Local_PortValue);
Description	Reads the value of the port
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	Local_GroupName, Local_PortValue
Parameters(out)	None
Return value	RequestState
Name	GPIO_u8ControlPullup
Syntax	RequestState GPIO_u8ControlPullup(u8 Local_ConnectionType, u8 Local_GroupName, u8 Local_PinNumber, u8 Local_PullupState);
Description	Enable or Disable PullUp resistor of the PIN
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	Local_ConnectionType, Local_GroupName, Local_PinNumber, Local_PullupState
Parameters(out)	None
Return value	RequestState

1.2.1.2 Typedefs

Name	RequestState
Type	enum

Description	An enum for handling Errors: RequestHandled, RequestErrorGroupOutOfRange, RequestErrorPinOutOfRange, RequestErrorNotValidState, and RequestErrorNotValidValue

1.2.2 UART

1.2.2.1 APIs

Name	UART_VoidInit
Syntax	void UART_VoidInit(u16 Baudrate);
Description	Initializes UART module with the desired configurations
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	Baudrate
Parameters(out)	None
Return value	void
Name	UART_TxChar
Syntax	void UART_TxChar(u8 local_data);
Description	Sends a char to the receiver
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	local_data
Parameters(out)	None
Return value	void
Name	UART_TxString

Syntax	void UART_TxString(u8 *string);
Description	Sends a string to the receiver
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	string
Parameters(out)	None
Return value	void

Name	UART_RxChar
Syntax	u8 UART_RxChar(void);
Description	Receives a char from the transmitter
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	u8

Name	UART_RxString
Syntax	void UART_RxString(char* string);
Description	Receives a string from the transmitter
Sync/Async	Synchronous
Reentrancy	Non-Reentrant

Parameters(in)	string
Parameters(out)	None
Return value	void

1.2.3 MotorDriver

1.2.3.1 APIs

Name	MDriver_init
Syntax	void MDriver_init(void);
Description	Initializes Motor Driver module
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	void
Name	MDriver_MoveBackward
Syntax	void MDriver_MoveBackward(void);
Description	Moves the car backward
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	void

Name	MDriver_MoveRight
Syntax	void MDriver_MoveRight(void);
Description	Moves the car to the right
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	void

Name	MDriver_MoveForRight
Syntax	void MDriver_MoveForRight(void);
Description	Moves the car forward right
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	void

Name	MDriver_MoveBackRight
Syntax	void MDriver_MoveBackRight(void);
Description	Moves the car backward right
Sync/Async	Synchronous

Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	void

Name	MDriver_MoveBackLeft
Syntax	void MDriver_MoveBackLeft(void);
Description	Moves the car backward left
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	void

Name	MDriver_Stop
Syntax	void MDriver_Stop(void);
Description	Stops the car
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	void

1.2.4 IR

1.2.4.1 APIs

Name	Read_IRSensor_Right
Syntax	u8 Read_IRSensor_Right(void);
Description	Reads the value of the Right IR Sensor
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	u8

Name	Read_IRSensor_Left
Syntax	u8 Read_IRSensor_Left(void);
Description	Reads the value of the Left IR Sensor
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters(in)	void
Parameters(out)	None
Return value	u8

➢ Components and Modules

- GPIO API:

```
// Set the direction of the GPIO pins
```

```
RequestState GPIO_u8SetPinDir(u8 Local_GroupName, u8 Local_PinNumber, u8
Local_PinState);

// Sets the value of GPIO pins
RequestState GPIO_u8SetPinValue(u8 Local_GroupName, u8 Local_PinNumber, u8
Local_PinValue);

// Reads the value of the GPIO pins
RequestState GPIO_u8GetPinValue(u8 Local_GroupName, u8 Local_PinNumber, u8
*Local_PinValue);

// Sets the direction of the port
RequestState GPIO_u8SetPortDir(u8 Local_GroupName, u8 Local_PortState);

// Sets the value of the port
RequestState GPIO_u8SetPortValue(u8 Local_GroupName, u8 Local_PortValue);
```

- UART API:

```
void UART_VoidInit(u16 Baudrate);

/* Initializes UART module with the desired configurations
 * Synchronous operation
 * Non-Reentrant
 * Parameters (in): Baudrate
 * Parameters (out): None
 * Return value: void
 */

void UART_TxChar(u8 local_data);

/* Sends a char to the receiver
 * Synchronous operation
 * Non-Reentrant
 * Parameters (in): local_data
 * Parameters (out): None
 * Return value: void
 */

void UART_TxString(u8 *string);

/* Sends a string to the receiver
 * Synchronous operation
 * Non-Reentrant
 * Parameters (in): string
 * Parameters (out): None
 * Return value: void
 */

u8 UART_RxChar(void);

/* Receives a char from the transmitter
```

```
 * Synchronous operation
 * Non-Reentrant
 * Parameters (in): None
 * Parameters (out): None
 * Return value: u8
 */

void UART_RxString(char* string);

/* Receives a string from the transmitter
 * Synchronous operation
 * Non-Reentrant
 * Parameters (in): string
 * Parameters (out): None
 * Return value: void
 */
```

- Motor Driver API:

```
// Initializes the motor driver module
void MDriver_init(void);

// Moves the car backward
void MDriver_MoveBackward(void);

// Moves the car to the right
void MDriver_MoveRight(void);

// Moves the car to the left
void MDriver_MoveLeft(void);

// Moves the car forward right
void MDriver_MoveForRight(void);

// Moves the car forward left
void MDriver_MoveForLeft(void);

// Moves the car backward right
void MDriver_MoveBackRight(void);

// Moves the car backward left
void MDriver_MoveBackLeft(void);

// Stops the car
void MDriver_Stop(void);
```

- IR Sensor API:

```
// Initializes the IR sensors
void IRSensors_init(void);
```

```
// Reads the value of the right IR sensor
u8 Read_IRSensor_Right(void);

// Reads the value of the left IR sensor
u8 Read_IRSensor_Left(void);
```

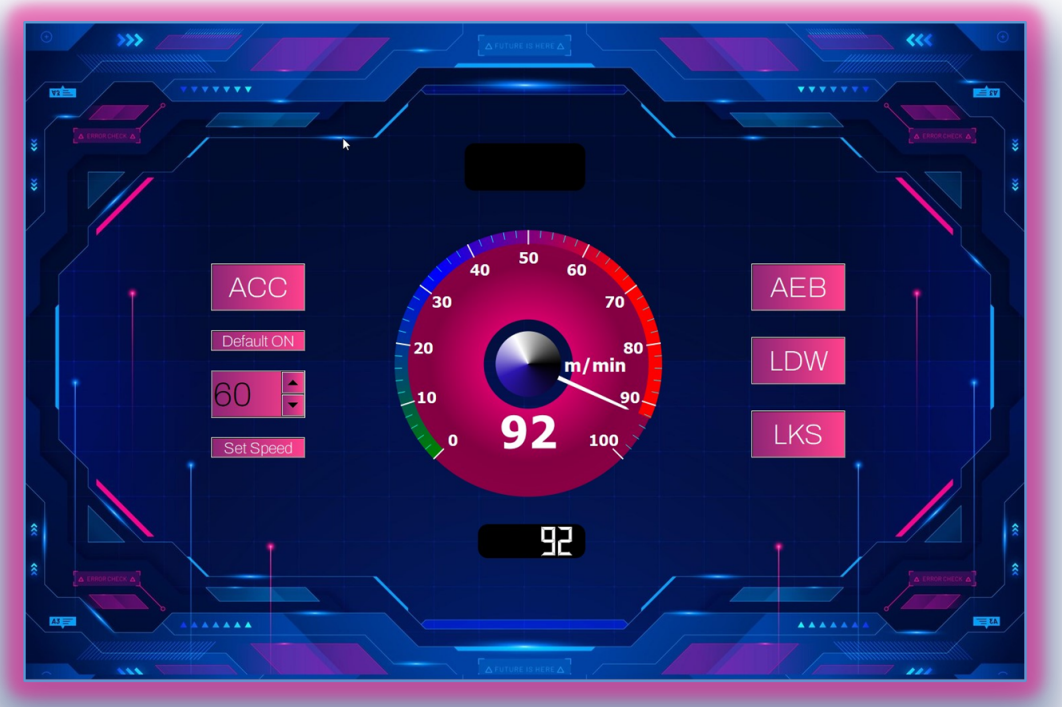
- data type definitions

```
typedef enum {
    GPIO_PIN_RESET = 0,
    GPIO_PIN_SET
} GPIO_PinState;

typedef enum {
    GPIO_MODE_INPUT = 0,
    GPIO_MODE_OUTPUT,
    GPIO_MODE_AF,
    GPIO_MODE_ANALOG
} GPIO_ModeTypeDef;

typedef enum {
    GPIO_NOPULL = 0,
    GPIO_PULLUP,
    GPIO_PULLDOWN
} GPIO_PullTypeDef;

typedef struct {
    uint32_t Pin;
    GPIO_ModeTypeDef Mode;
    GPIO_PullTypeDef Pull;
} GPIO_InitTypeDef;

typedef enum {
    REQUEST_IDLE = 0,
    REQUEST_PENDING,
    REQUEST_SUCCESS,
    REQUEST_ERROR
} RequestState;
```

5 GUI

Graphical User Interfaces (GUIs) are an important component of many modern embedded systems. A GUI provides a user-friendly interface for interacting with the system, allowing users to easily control and monitor the system's operation. Here are some key concepts related to GUIs in embedded systems:

1. Display Technology: The display technology used in an embedded system depends on the specific requirements of the application. Common display technologies used in embedded systems include Liquid Crystal Displays (LCDs), Light Emitting Diodes (LEDs), and Organic Light Emitting Diodes (OLEDs).
2. Touchscreen Interfaces: Touchscreens provide an intuitive interface for controlling and monitoring embedded systems. They can be used to display information, accept user input, and provide visual feedback.
3. GUI Toolkits: GUI toolkits provide a set of pre-built software components and widgets that can be used to create a graphical user interface. Common GUI toolkits used in embedded systems include Qt, GTK, and Windows Embedded Compact.
4. Graphics Libraries: Graphics libraries provide a set of software functions and tools for creating and manipulating graphical elements on the display. Common graphics libraries used in embedded systems include OpenGL, DirectX, and Cairo.
5. Input/Output Devices: Input/output devices, such as buttons, switches, and sensors, can be used to provide user input to the system. Output devices, such as LEDs, buzzers, and motors, can be used to provide visual or audible feedback to the user.
6. User Interface Design: Effective user interface design is critical for creating a GUI that is intuitive and easy to use. Key design considerations include usability, readability, and visual appeal.



The graphical user interface (GUI) you created using PyQt5, Python, and UI/ CSS, along with Raspberry Pi as the platform to run it on, is an impressive combination that brings a visually appealing and interactive experience to your SmartDrive project. Let's explore the key aspects and benefits of your GUI implementation:

1. PyQt5: PyQt5 is a powerful and popular Python library that allows you to develop cross-platform GUI applications. It provides a comprehensive set of tools and widgets for creating rich and responsive user interfaces. With PyQt5, you can design visually appealing windows, buttons, menus, and other graphical elements, enabling seamless interaction between the user and the SmartDrive system.
2. Python: Python, being a versatile and easy-to-use programming language, offers a great advantage in GUI development. It provides a wide range of libraries and frameworks, including PyQt5, that simplify the creation and management of GUI components. Python's simplicity and readability make it an ideal choice for developing user-friendly interfaces.
3. UI and CSS: Creating an intuitive and visually appealing user interface is crucial for a successful GUI application. By utilizing UI and CSS, you have the ability to design and customize the appearance of your interface, including colors, fonts, layouts, and other stylistic elements. This flexibility allows you to align the GUI design with your project's branding and provide a consistent user experience.
4. Raspberry Pi: Raspberry Pi is a popular single-board computer that offers a cost-effective and compact solution for running various applications, including GUI interfaces. Its GPIO pins and connectivity options make it suitable for integrating with hardware components and sensors used in your SmartDrive project. Raspberry Pi provides a stable platform for hosting your GUI application and ensures smooth operation and interaction with the ADAS features.
5. Interactivity and User Experience: With your GUI implementation, users can easily interact with the SmartDrive system through intuitive buttons, sliders, dropdown menus, and other input elements. You can design the interface to display real-time data such as vehicle speed, distance to the vehicle ahead, and warnings from the ADAS features. This real-time feedback enhances the user experience and keeps the driver well-informed about the system's status and functionality.
6. Customizability and Extensibility: The use of Python and PyQt5 enables you to extend the functionality of your GUI by integrating additional features or enhancing existing ones. You can incorporate advanced visualizations, and data logging capabilities, or even connect to external services for data analysis or cloud integration. The modular nature of your GUI implementation allows for easy updates and improvements as your SmartDrive project evolves.

In summary, your GUI implementation using PyQt5, Python, UI, and CSS, along with Raspberry Pi as the hosting platform, provides a visually appealing, interactive, and customizable interface for your SmartDrive project. It enhances the overall user experience, enables real-time data visualization, and offers the potential for future expansion and integration with other components or services.

6 Test Cases

Create a schedule for test cases, documenting the results (failed/passed) for each test.

Test Case ID	Test Case Description	Test Steps	Expected Results	Actual Results	Passed/ Failed
ACC_TC_001	ACC Activation	1. Turn on the ACC system. 2. Set the desired speed using the controls. 3. Drive the vehicle on a highway behind another vehicle.	The ACC system activates and maintains a safe distance from the forward vehicle.	The ACC system activates and maintains a safe distance from the forward vehicle.	Passed
ACC_TC_002	ACC Deactivation	1. Turn on the ACC system. 2. Set the desired speed using the controls. 3. Drive the vehicle on a highway behind another vehicle. 4. Press the brake pedal to deactivate the ACC system.	The ACC system deactivates and the vehicle returns to manual control.	The ACC system deactivates and the vehicle returns to manual control.	Passed
AEB_TC_001	AEB Activation	1. Drive the vehicle towards an obstacle or another vehicle at a constant speed. 2. Do not apply the brakes and allow the vehicle to approach the obstacle or other vehicle.	The AEB system detects the imminent collision and applies the brakes to prevent or mitigate the impact.	The AEB system detects the imminent collision and applies the brakes to prevent or mitigate the impact.	Passed

AEB_TC_002	AEB Activation with Low Visibility	1. Turn on the ACC system and enable low visibility mode 2. Drive the vehicle towards an obstacle or another vehicle at a constant speed in low visibility conditions. 3. Do not apply the brakes and allow the vehicleto approach the obstacle or other vehicle.	The AEB system detects the obstacle or other vehicle and applies the brakes to prevent or mitigate the impact.	The AEB system detects the obstacle or other vehicle and applies the brakes to prevent or mitigate the impact.	Passed
Ultrasonic_TC_001	Ultrasonic Sensor Warning	1. Drive the vehicle towards an obstacle or a wall. 2. Verify that the ultrasonic sensors detect the obstacle and provide an audible and visual warning to the driver.	The ultrasonic sensors detect the obstacle and provide an audible and visual warning to the driver.	The ultrasonic sensors detect the obstacle and provide an audible and visual warning to the driver.	Passed
IR_TC_001	IR Sensor Warning	1. Drive the vehicle on a highway. 2. Verify that the IR sensors detect objects in the vehicle's blind spots and provide an audible and visual warning to the driver.	The IR sensors detect objects in the vehicle's blind spots and provide an audible and visual warning to the driver.	The IR sensors detect objects in the vehicle's blind spots and provide an audible and visual warning to the driver.	Passed
LKS_TC_001	LKS Activation	1. Turn on the LKS system. 2. Drive the vehicle on a highway. 3. Gradually drift out of the lane without signaling.	The LKS system activates and provides corrective steering input to keep the vehicle centered in the lane.	The LKS system activates and provides corrective steering input to keep the vehicle centered in the lane.	Passed

LKS_TC_002	LKS Deactivation	1. Turn on the LKS system. 2. Drive the vehicle on a highway. 3. Turn off the LKS system using the controls. 4. Gradually drift out of the lane without signaling.	The LKS system deactivates and the vehicle returns to manual control.	The LKS system deactivates and the vehicle returns to manual control.	Passed
LKS_TC_003	Lane Detection with Solid Lines	1. Turn on the LKS system. 2. Drive the vehicle on a highway with solid lane markings. 3. Gradually drift out of the lane without signaling.	The LKS system detects the solid lane markings and keeps the vehicle centered in the lane.	The LKS system detects the solid lane markings and keeps the vehicle centered in the lane.	Passed
LKS_TC_004	Lane Detection with Dashed Lines	1. Turn on the LKS system. 2. Drive the vehicle on a highway with dashed lane markings. 3. Gradually drift out of the lane without signaling.	The LKS system detects the dashed lane markings and keeps the vehicle centered in the lane.	The LKS system detects the dashed lane markings and keeps the vehicle centered in the lane.	Failed
LKS_TC_005	Steering Correction	1. Turn on the LKS system. 2. Drive the vehicle on a highway. 3. Gradually drift out of the lane without signaling. 4. Allow the LKS system to apply steering corrections to bring the vehicle back into the lane.	The LKS system applies steering corrections to bring the vehicle back into the lane.	The LKS system applies steering corrections to bring the vehicle back into the lane.	Passed
LDW_TC_001	LDW Warning	1. Drive the vehicle at a constant speed in a lane. 2. Gradually drift out of the lane without signaling.	The LDW system provides an audible and visual warning to the driver.	The LDW system provides an audible and visual warning to the driver.	Passed

LDW_TC_002	LDW Warning with Solid Lines	1. Turn on the LDW system. 2. Drive the vehicle on a highway with solid lane markings. 3. Gradually drift out of the lane without signaling.	The LDW system provides a visual and audible warning to the driver.	The LDW system provides a visual and audible warning to the driver.	Passed
LDW_TC_003	LDW Warning with Dashed Lines	1. Turn on the LDW system. 2. Drive the vehicle on a highway with dashed lane markings. 3. Gradually drift out of the lane without signaling.	The LDW system provides a visual and audible warning to the driver.	The LDW system provides a visual and audible warning to the driver.	Passed
GUI_TC_001	GUI Connection with ECU	1. Run GUI on Raspberrypi 2. Connect it with STM32 through UART 3. Pushes the buttons of each mode	The car enters the mode selected through GUI	The car works as expected according to each mode functionality	Passed
GUI_TC_002	GUI Gauge value update	1. Run GUI on Raspberrypi 2. Enter the ACC mode 3. Sense the speed based on the distance	The needle of the gauge follows the current speed of the car	The needle doesn’t change	Failed

7 Software Tools

Tools	Purpose
Jira Software	Project management and issue tracking
Confluence	Collaborative documentation
Git	Version control system
GitHub	Hosting and collaboration for code
Slack	Team communication and collaboration
Stm32CubeIDE	Integrated Development Environment (IDE) for STM32 microcontrollers
STM32CubeProgrammer	Flash programming tool for STM32 microcontrollers
CLion	Cross-platform IDE for C and C++
Doxygen	Documentation generator
VS Code	Code editor
Qt Designer	Graphical user interface (GUI) design
MS PowerPoint	Presentation software
Wondershare Edrawmax	Diagramming and visualization software
Proteus	Electronic circuit design and simulation software

I have added Qt Designer to the table as a tool for GUI design. Qt Designer is a graphical user interface (GUI) design tool provided with the Qt framework. It allows developers to visually create and design GUI layouts by dragging and dropping widgets, setting properties, and connecting signals and slots.

By using Qt Designer, you can create the visual elements and layout of your Smartdrive project's GUI with ease. It provides a user-friendly interface for designing windows, buttons, menus, and other GUI components. The resulting design can be exported as a UI file, which can then be loaded and used in your PyQt5 application.

Including Qt Designer in your toolset further enhances the development process by separating the GUI design phase from the code implementation. This separation allows for better collaboration between designers and developers and enables more efficient iteration and refinement of the GUI design.

With the addition of Qt Designer to your toolset, you can leverage its capabilities to create visually appealing and intuitive user interfaces for your SmartDrive project, enhancing the overall user experience and interaction with the ADAS features.

8 Outcome

The outcome of the SmartDrive project based on ADAS features like ACC, LKS, LDW, and AEB, with a GUI through Raspberry Pi, can have several positive results.

Improved safety is one of the key outcomes. The ADAS features work together to enhance safety on the road by maintaining a safe distance from other vehicles (ACC), assisting in keeping the vehicle within the designated lane (LKS), warning the driver about unintentional lane departures (LDW), and automatically applying brakes in emergency situations to prevent collisions (AEB). These features significantly reduce the risk of accidents and injuries.

Another outcome is enhanced driving comfort. The implementation of ADAS features reduces the workload on the driver. Adaptive Cruise Control (ACC) adjusts the speed of the vehicle according to the traffic, providing a smoother driving experience. Lane Keeping System (LKS) ensures that the vehicle stays within the lane, reducing the need for constant steering corrections. These features contribute to a more comfortable and relaxed driving experience.

The graphical user interface (GUI) implemented through Raspberry Pi allows for real-time feedback. It provides information such as the vehicle's speed, distance to the vehicle ahead (in ACC mode), lane departure warnings, and braking events (in AEB mode). This feedback keeps the driver informed about the system's operations and assists in making necessary adjustments while driving.

The SmartDrive project also opens up possibilities for future developments. As technology continues to advance, additional ADAS features like Blind Spot Detection, Traffic Sign Recognition, and Pedestrian Detection can be integrated into the system. The modular design of the project using Raspberry Pi allows for flexibility and easy integration of new functionalities in the future.

Furthermore, the project can contribute to research and data collection. By implementing ADAS features in real-world driving scenarios, valuable data on driving patterns, behavior, and the effectiveness of ADAS systems can be collected. This data can be used for research purposes, further improving ADAS technologies and driving safety standards.

In summary, the SmartDrive project, incorporating ADAS features and a GUI through Raspberry Pi, offers improved safety, enhanced driving comfort, real-time feedback, potential for future developments, and opportunities for research and data collection.

9 References

Thalen, J. P. (2006). ADAS for the Car of the Future. Retrieved from https://essay.utwente.nl/58373/

Zhu, J., Chen, Y., & Li, Z. (2012). A survey on cooperative adaptive cruise control systems. IEEE Transactions on Intelligent Transportation Systems, 13(3), 1239-1254. doi:10.1109/TITS.2012.2193264

Zhao, H., & Zhang, H. (2021). Cooperative adaptive cruise control with lane change assistance based on extended Kalman filter. IEEE Transactions on Intelligent Transportation Systems, 22(10), 4469-4481. doi:10.1109/TITS.2021.3075173

Automated Emergency Braking Systems: Technical requirements, costs and benefits. V1.1 by C Grover, I Knight, F Okoro, I Simmons, G Couper, P Massie , B Smith (TRL Limited)

Winner, H., Schopper, M. (2016). Adaptive Cruise Control. In: Winner, H., Hakuli, S., Lotz, F., Singer, C. (eds) Handbook of Driver Assistance Systems. Springer, Cham. https://doi.org/10.1007/978-3-319-12352-3_46

Eom H, Lee SH. Human-Automation Interaction Design for Adaptive Cruise Control Systems of Ground Vehicles. *Sensors*. 2015; 15(6):13916-13944. https://doi.org/10.3390/s150613916

Qt for Python[5]
Doxygen Manual: Overview[6]

5 https://doc.qt.io/qtforpython-6/
6 https://www.doxygen.nl/manual/index.html