Project Report

On

ADAPTIVE CRUISE CONTROL

Submitted
In partial fulfilment

For the award of the Degree of

PG-Diploma in Embedded Systems and Design (PG-DESD)

C-DAC, ACTS (Pune)

Guided By:	**Submitted By:**	
Mr. Arafat Khan	Akshit Bangarwa	240840130003
	Shraddha Ankush Mane	240840130019
	Swajal Kumar Jha	240840130043
	Vaibhavi Pramod Shahare	240840130048
	Velugu Vekata Koonal	240840130003

Centre for Development of Advanced Computing (C-DAC), ACTS

(Pune- 411008)

ABSTRACT

The project uses an **Adaptive Cruise Control System (ACC)** based on an STM32 Discovery board, which is uniquely designed to simulate the Electronic Control Unit (ECU) of a car. The system utilizes different components such as an ultrasonic sensor to calculate the gap between cars, a variable RPM motor to simulate car wheels, and an accelerometer to simulate the movement of the steering wheel. The ESP32 module is used for data transfer to the cloud, thus providing a black box feature to monitor car data in real-time. The system also utilizes a Real-Time Operating System (RTOS) to provide effective task scheduling and synchronization, in addition to using the Controller Area Network (CAN) for smooth data transfer among different modules.

Table of Contents

Chapter 1
Introduction

1.1 Introduction

Adaptive Cruise Control (ACC) is a modern driver-assistance technology designed to enhance road safety and driving comfort by automatically adjusting a vehicle’s speed based on traffic conditions. Unlike traditional cruise control systems that maintain a constant speed, ACC continuously monitors the distance to the preceding vehicle and dynamically modifies the speed to ensure sage following distances. This system reduces driver Fatigue, improves fuel efficiency, and minimizes the risk of read-end collisions.

This project aims to implement and **Adaptive Cruise Control System** using an **STM32 Discovery Board** as primary Microcontroller. The system employs an **Ultrasonic Sensor** to measure the distance between vehicles and performs real-time calculations to regulate motor speed (acting as wheels of the vehicles). By adjusting the **RPM (Revolution Per Minute)** of the motor, the system simulates vehicle acceleration and deceleration to maintain a safe gap. The data such as Speed of Vehicle, Distance between of vehicles, etc. will be send to the cloud with the help of **ESP32** acting as **Blackbox feature**.

1.2 Objective

The objectives of the project work are as –

- Develop a real-time Adaptive Cruise Control system that dynamically adjusts vehicle speed based on proximity to surrounding objects.

- Implement an ultrasonic sensor to accurately measure the distance between vehicles and provide input for speed regulation.

- Utilize a variable RPM motor to simulate vehicle movement and adjust speed based on the computed distance data.

- Integrate an ESP32 module to enable real-time data transmission to the cloud, acting as a black box for event logging and remote monitoring.

- Implement RTOS for effective task scheduling and synchronization, ensuring seamless operation of all system components.

- Use CAN communication to facilitate data exchange between different embedded components, emulating real-world automotive communication protocols.

- Leverage the inbuilt accelerometer of the STM32 Discovery board to simulate lane-changing functionality, allowing user interaction with the system.

1.3 Specifications

The specifications of the project work are as –

- Microcontroller: STM32 Discovery Board (serving as the ECU).
- Sensors: Ultrasonic sensor for distance measurement.
- Actuators: Variable RPM motor for speed adjustment.
- Connectivity: ESP32 module for cloud-based data transmission.
- Operating System: Real-Time Operating System (RTOS) for managing concurrent tasks.
- Communication Protocol: CAN bus for inter-device communication.
- User Input: STM32's built-in accelerometer for lane control.

Chapter 2
Literature Review

IEEE Transactions on Intelligent Vehicles - "Adaptive Cruise Control: A Literature Review" [1], Adaptive Cruise Control which has been an area of research in automotive engineering, mainly because of its ability to improve traffic safety and efficiency. Early implementations were radar and LIDAR-based for the detection of near vehicles and obstacles. These were, however, expensive and computationally intensive. In recent years, ultrasonic sensors have been explored as a less expensive option for short-range distance measurement and will be helpful for smaller scale prototypes models.

Embedded Systems Design Handbook - "Real-Time Operating Systems for Embedded Systems" [2], The use of Real-Time Operating Systems (RTOS) in embedded systems is a standard practice that allows several tasks to be executed within a time-critical environment. Research has proven that RTOS significantly enhances the reliability of automotive control systems through deterministic task scheduling and latency reduction. Additionally, the Controller Area Network (CAN) has become the de facto standard protocol for vehicle communication, providing robust and efficient data exchange between Electronic Control Units (ECUs), sensors, and actuators.

Through the use of these technologies, this project is in line with ongoing research in automotive safety and embedded system design, providing a practical and scalable implementation of Adaptive Cruise Control (ACC).

Chapter 3
Methodology and Techniques

The development of the Adaptive Cruise Control (ACC) System followed a structured, step-by-step approach to ensure proper integration and synchronization of all components. The methodology was divided into several key phases:

1. System Design and Component Selection
2. Hardware Integration
3. Software Development and RTOS Implementation
4. Communication and Data Exchange
5. Testing and Validation

Each of these phases played a crucial role in achieving a reliable and functional Adaptive Cruise Control system.

3.1. System Design and Component Selection

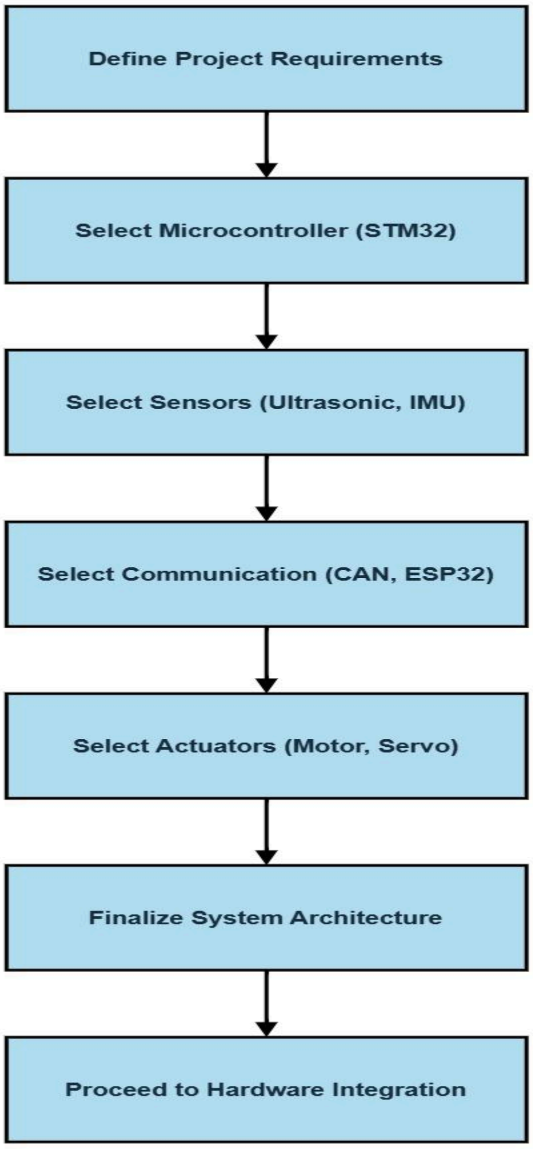
The first step was to define the functional requirements and select the appropriate hardware components to meet project objectives. Since Adaptive Cruise Control requires real-time decision-making, the STM32 Discovery Board was chosen due to its high processing speed, embedded peripherals, and RTOS support.

Key Considerations for Hardware Selection:

- Microcontroller (STM32 Discovery Board) – Acts as the ECU, processes sensor data, and controls motor speed.
- Ultrasonic Sensor – Measures the distance to the preceding vehicle and provides real-time input for speed adjustments.
- Variable RPM Motor – Simulates vehicle wheels and adjusts speed dynamically

based on sensor data.

- ESP32 Module – Transmits operational data to the cloud via CAN communication.
- Accelerometer (Inbuilt in STM32) – Detects driver input for lane-changing functionality.
- Controller Area Network (CAN) – Enables efficient and reliable communication between STM32 and ESP32.
- Real-Time Operating System (RTOS) – Ensures task synchronization and time-critical operations.


Define Project Requirements
Select Microcontroller (STM32)
Select Sensors (Ultrasonic, IMU)
Select Communication (CAN, ESP32)
Select Actuators (Motor, Servo)
Finalize System Architecture
Proceed to Hardware Integration

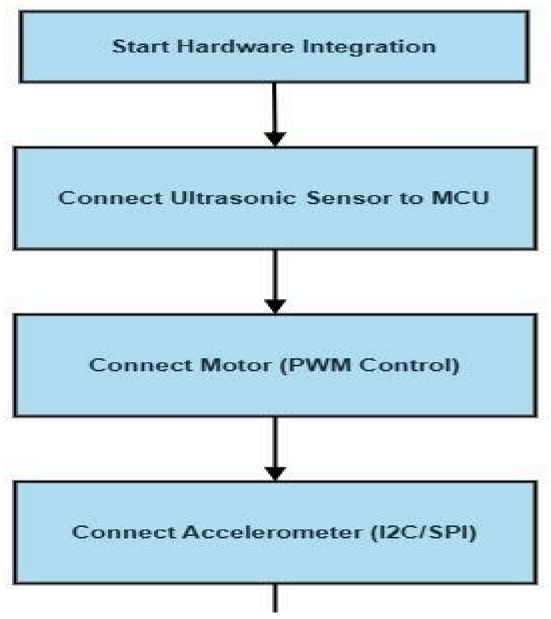
3.2. Hardware Integration

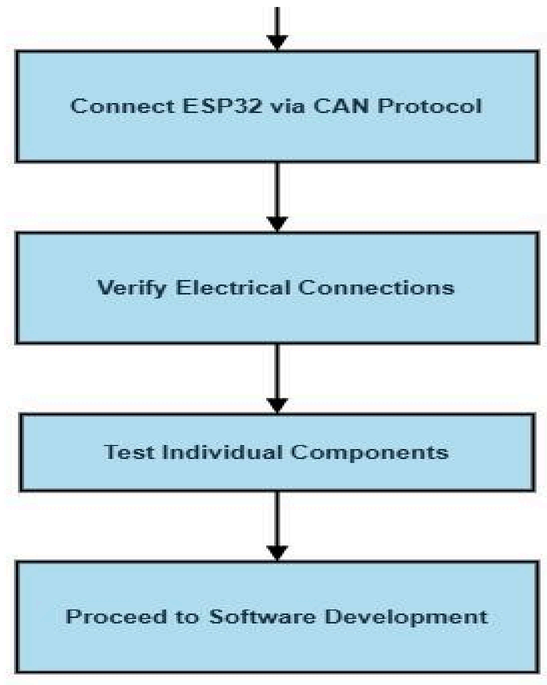
The next phase was to physically connect and integrate all components to enable seamless data flow. The connections were established as follows:

- STM32 → Ultrasonic Sensor: Connected via GPIO pins to measure distance and provide input for speed control.
- STM32 → Variable RPM Motor: Connected via PWM output, allowing dynamic speed adjustments based on proximity data.
- STM32 → ESP32 (via CAN Bus): Enables data exchange and cloud connectivity.
- STM32 → Servo Motor (Lane Change): Controlled via PWM signals based on accelerometer input.

Key Hardware Integration Steps:

1. Configuring GPIOs for sensor inputs and motor control outputs.
2. Setting up PWM signals for motor speed regulation and servo movement.
3. Interfacing CAN bus for efficient data transfer between STM32 and ESP32.
4. Calibrating the accelerometer for precise detection of lane-changing gestures.


Start Hardware Integration
Connect Ultrasonic Sensor to MCU
Connect Motor (PWM Control)
Connect Accelerometer (I2C/SPI)


Connect ESP32 via CAN Protocol
Verify Electrical Connections
Test Individual Components
Proceed to Software Development

3.3. Software Development and RTOS Implementation

Once the hardware was set up, the next step was developing the firmware for real-time decision-making. This involved sensor data processing, motor control, CAN communication, and RTOS task scheduling.

RTOS Task Scheduling and Priorities

To ensure efficient operation, RTOS (Real-Time Operating System) was implemented, allowing multiple tasks to run concurrently. The following priority levels were assigned:

Task	**Priority**	**Description**
Distance Measurement (Ultrasonic Sensor)	High	Ensures real-time obstacle detection and braking
Motor Speed Control	High	Adjusts vehicle speed dynamically
Steering Control (Accelerometer to	Medium	Allows lane changes based on user input

Task	Priority	Description
Servo)		
Data Logging & Transmission (CAN to ESP32)	Low	Logs vehicle data to the cloud

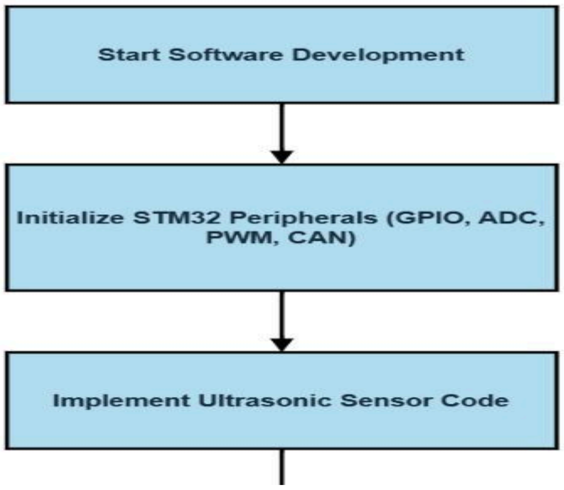
Key RTOS Features Used:

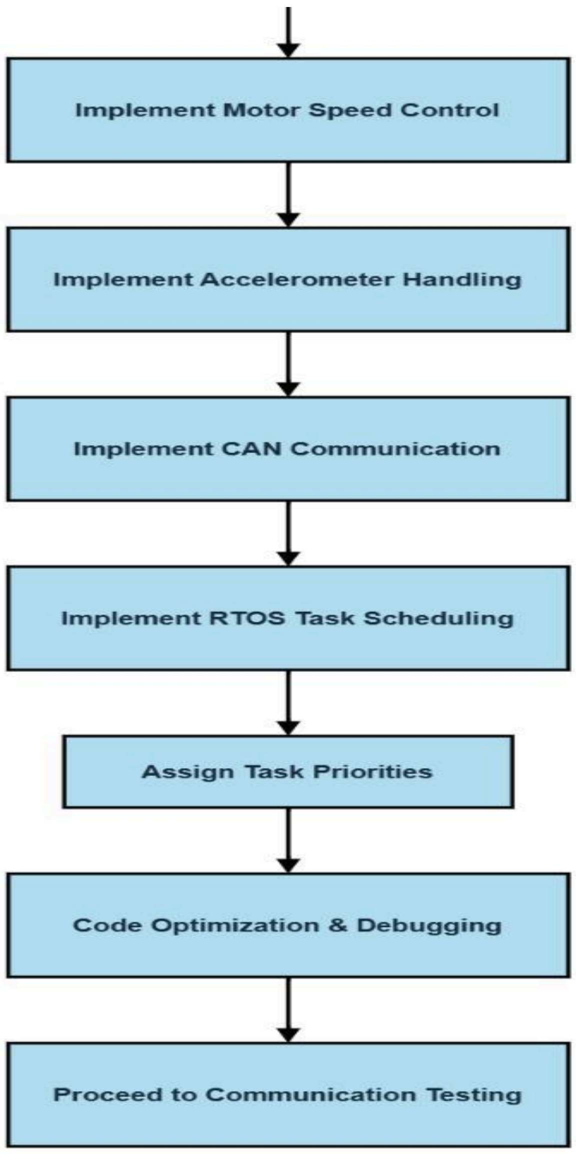
- Task Preemption: Ensured immediate execution of critical tasks.
- Semaphore Synchronization: Prevented data conflicts in shared resources.
- Message Queues: Facilitated inter-task communication.

Key Software Development Steps:

1. Initializing STM32 Peripherals: Set up GPIOs, ADC, PWM, and CAN interfaces.
2. Implementing Distance Measurement Algorithm: Captured ultrasonic sensor readings and processed them using a distance threshold function.
3. Motor Control Logic: Adjusted PWM duty cycles based on distance calculations.

4. Steering Control Logic: Processed accelerometer inputs to control the servo motor for lane changing.
5. Cloud Communication: Implemented CAN-based messaging to send data from STM32 to ESP32 for cloud logging.


Start Software Development
Initialize STM32 Peripherals (GPIO, ADC, PWM, CAN)
Implement Ultrasonic Sensor Code


Implement Motor Speed Control
Implement Accelerometer Handling
Implement CAN Communication
Implement RTOS Task Scheduling
Assign Task Priorities
Code Optimization & Debugging
Proceed to Communication Testing

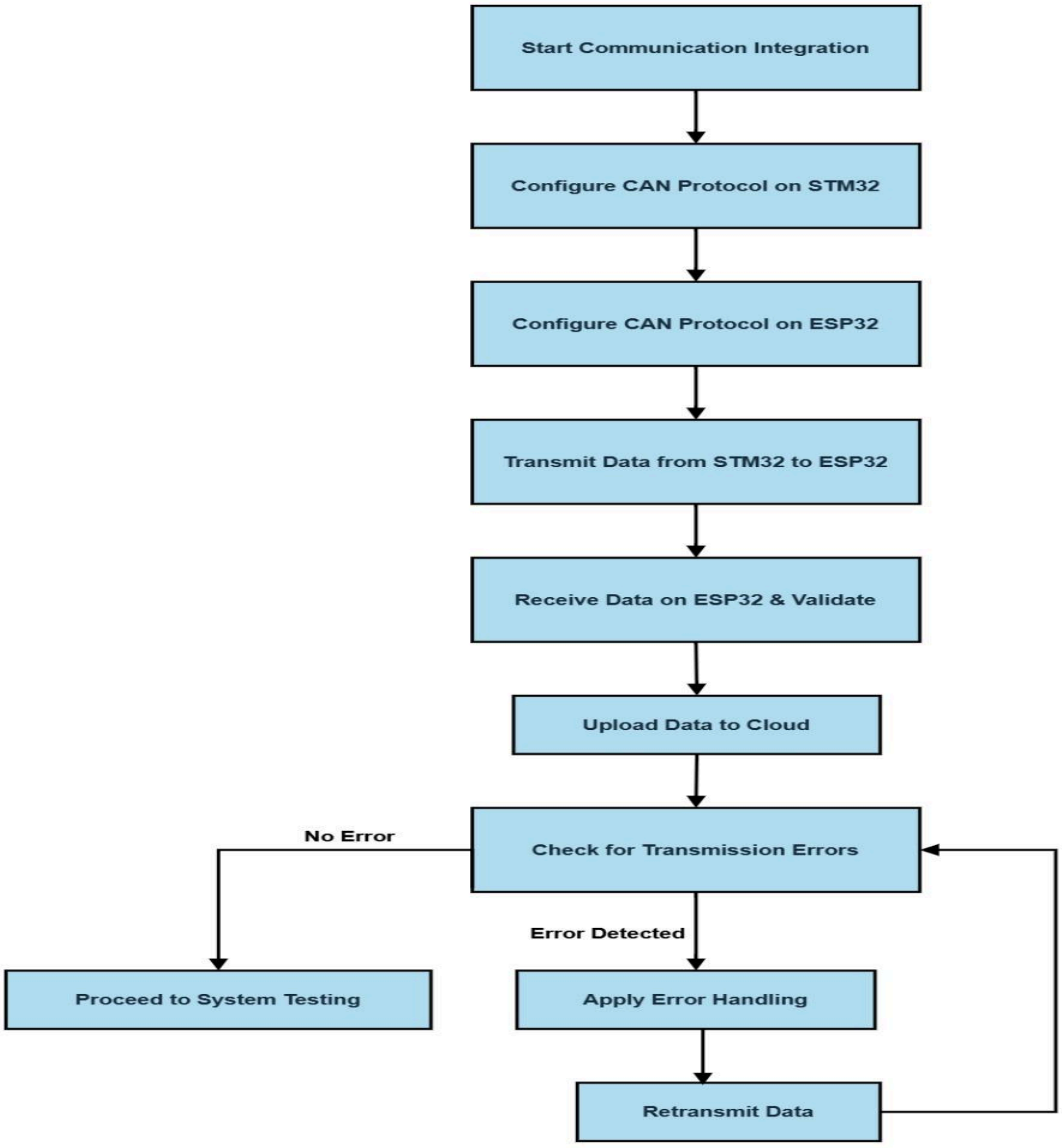
3.4. Communication and Data Exchange

Efficient data transfer between different modules was crucial for real-time operation. This was achieved using the CAN (Controller Area Network) protocol, ensuring reliable message passing between the STM32 microcontroller and the ESP32 cloud module.

CAN Communication Implementation:

1. Configuring CAN Bus:
   - STM32 sends sensor data as CAN frames to ESP32.
   - ESP32 reads the frames, processes the data, and uploads it to the cloud.

2. Message Structure:
    - ID: Unique identifier for each data type (Distance, Speed, Steering Angle).
    - Data: Sensor readings and control commands (e.g., Speed Adjustment, Braking Status).
3. Error Handling in CAN:
    - Implemented CRC checks to detect and correct transmission errors.
    - Used ACK messages to confirm data reception.

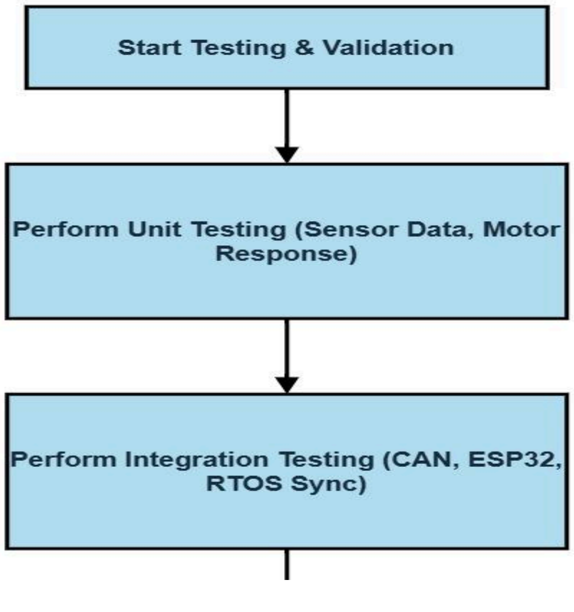

Start Communication Integration
Configure CAN Protocol on STM32
Configure CAN Protocol on ESP32
Transmit Data from STM32 to ESP32
Receive Data on ESP32 & Validate
Upload Data to Cloud
Check for Transmission Errors
No Error
Error Detected
Proceed to System Testing
Apply Error Handling
Retransmit Data

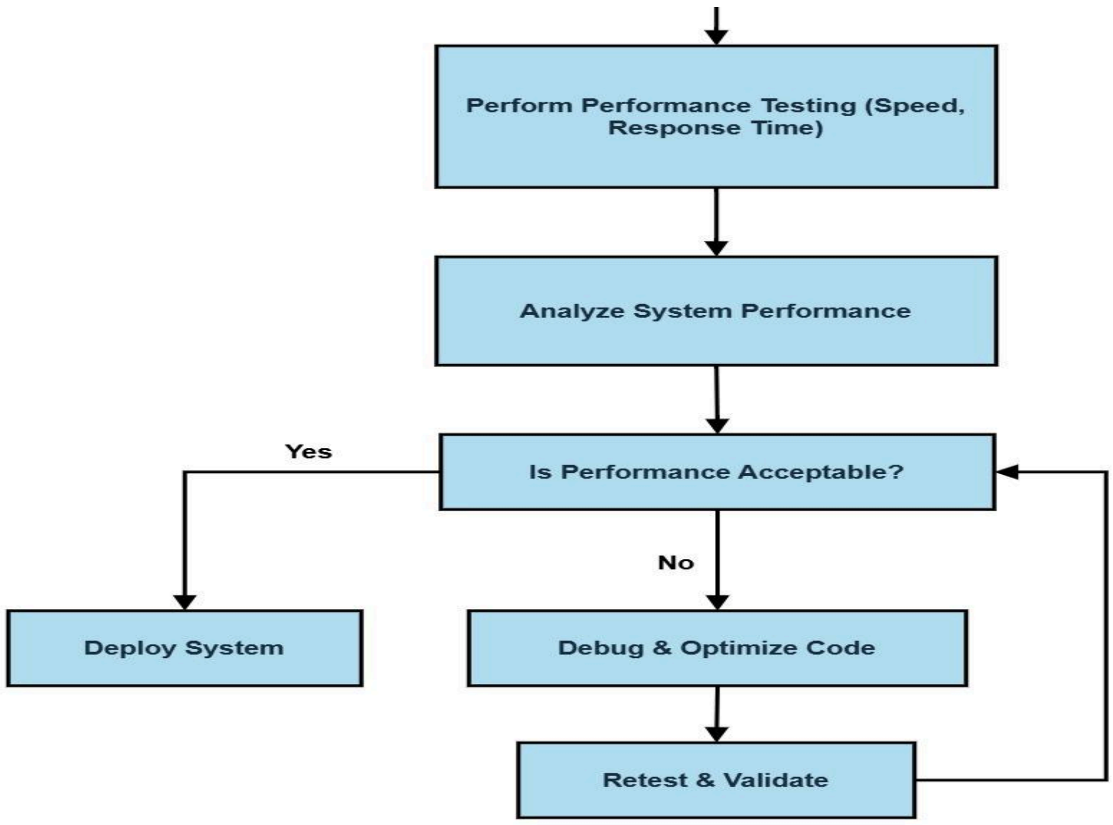
3.5. Testing and Validation

After hardware integration and software development, extensive testing and validation were conducted to ensure smooth operation.

Testing Procedures:

1. Unit Testing:
   - Verified sensor readings against expected values.
   - Checked motor response at different distance thresholds.
   - Ensured servo motor steering control responded correctly to accelerometer input.
2. Integration Testing:
   - Tested CAN communication between STM32 and ESP32.
   - Verified real-time synchronization using RTOS task scheduling.
3. Performance Testing:
   - Measured response time for speed adjustments.
   - Evaluated RTOS scheduling efficiency under different loads.


Start Testing & Validation
Perform Unit Testing (Sensor Data, Motor Response)
Perform Integration Testing (CAN, ESP32, RTOS Sync)


Perform Performance Testing (Speed, Response Time)
Analyze System Performance
Is Performance Acceptable?
Yes
No
Deploy System
Debug & Optimize Code
Retest & Validate

3.6. Summarization

Phase	Key Activities
System Design & Component Selection	Defined system requirements and selected suitable hardware components.
Hardware Integration	Connected STM32 to sensors, motors, and communication modules.
Software Development	Developed embedded C firmware for data processing, control logic, and RTOS scheduling.
Communication & Data Exchange	Implemented CAN protocol for STM32-to-ESP32 data transfer.
Testing & Validation	Conducted unit, integration, and performance testing to ensure system reliability.

Chapter 4
Implementation

The implementation of the Adaptive Cruise Control (ACC) system involved integrating multiple hardware components and developing firmware for seamless operation. The STM32 Discovery Board acted as the central controller (ECU), interfacing with various sensors and actuators. The implementation was divided into four major parts:

1. STM32 to Ultrasonic Sensor

One of the core functionalities of the ACC system is distance measurement using an ultrasonic sensor, which determines the proximity of the leading vehicle. The sensor transmits ultrasonic waves, and based on the time taken for the waves to return after hitting an obstacle, the STM32 calculates the distance.

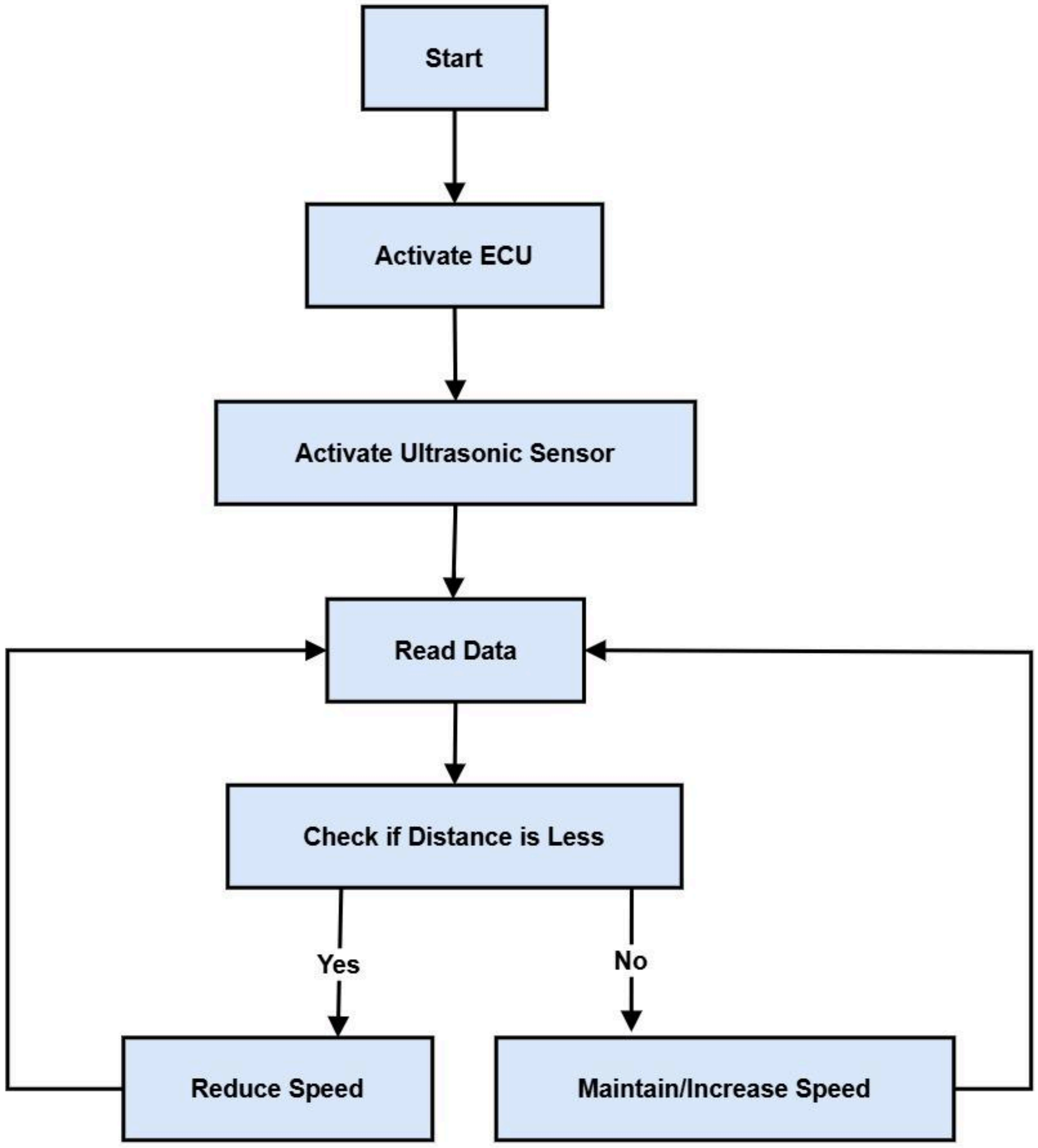
Steps Involved:

- The ultrasonic sensor is connected to the STM32, which triggers the sensor to emit sound waves.
- The echo pin receives the reflected signal and calculates the distance based on the formula:

  Distance = (Time × Speed of Sound) / 2
- The STM32 continuously monitors this data and adjusts the RPM of the motor accordingly:
  - If the distance is greater than the safe threshold, the RPM motor maintains or increases speed.
  - If the distance is less than the threshold, the STM32 decreases the motor speed to prevent collision.
  - If the distance is too close, the STM32 stops the motor completely to simulate braking.

Challenges and Solutions:

- Real-time response: Implemented priority scheduling in RTOS to ensure immediate response to proximity changes.
- Noise reduction: Used filtering techniques to eliminate fluctuations in sensor readings.



2. **STM32 to ESP32**

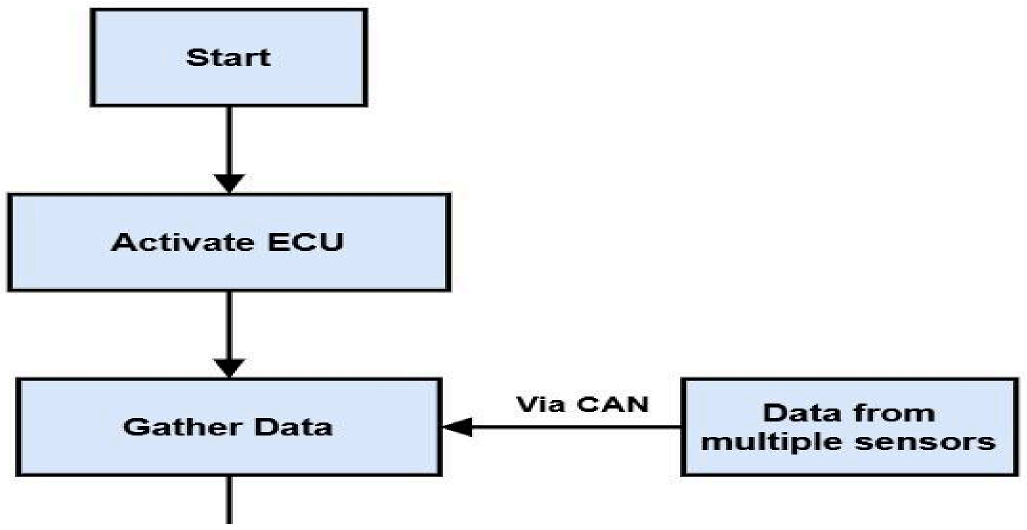
For real-time data logging and monitoring, the ESP32 module was integrated to send vehicle parameters (speed, distance, lane status) to the cloud. Communication between STM32 and ESP32 was established using Controller Area Network (CAN), a protocol widely used in automotive applications.

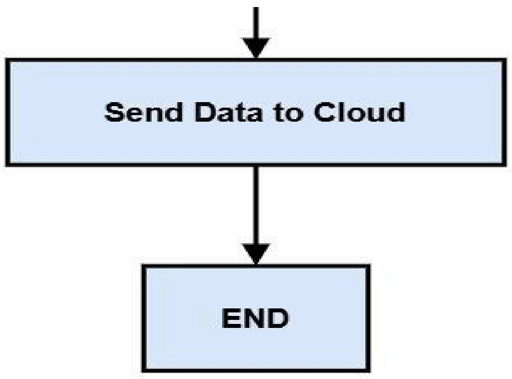
Steps Involved:

- STM32 collects distance, speed, and steering angle data.
- Using CAN protocol, data is sent to the ESP32 module.
- ESP32 receives the CAN message, processes it, and transmits it over Wi-Fi to the cloud.
- The cloud receives and logs the data, allowing remote monitoring of vehicle parameters.

Challenges and Solutions:

- Synchronization issues: Implemented priority-based task scheduling in RTOS to manage CAN messages efficiently.
- Data loss mitigation: Implemented ACK-based error handling to ensure reliable transmission between STM32 and ESP32.





3. **STM32 to Servo Motor**

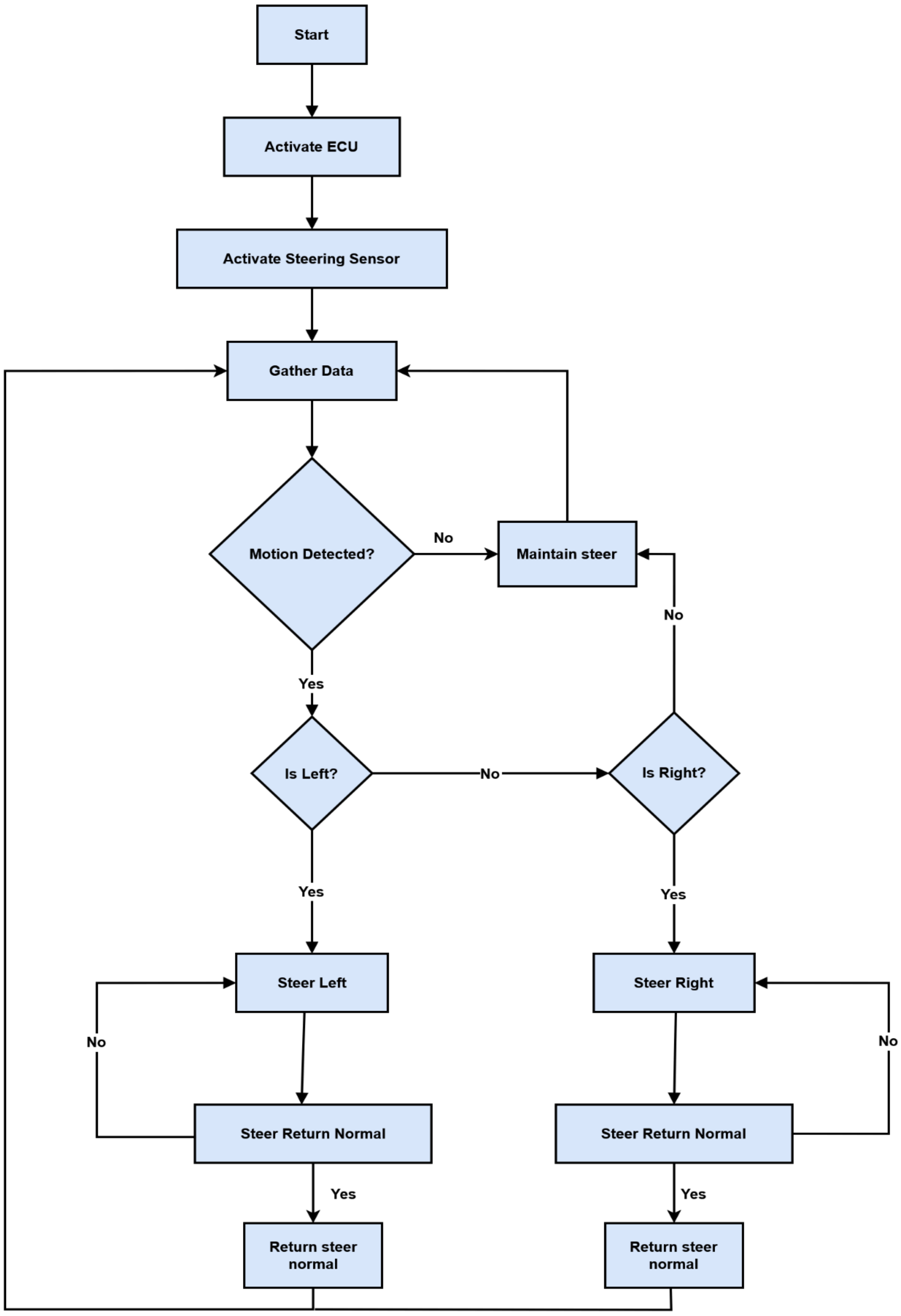
To simulate lane-changing functionality, the STM32's inbuilt accelerometer was utilized as a steering input, which controlled a servo motor representing the vehicle’s wheels.

Steps Involved:

- The STM32 continuously monitors accelerometer data to detect tilt along the X-axis.
- Based on the tilt values, the STM32 sends PWM signals to the servo motor to adjust the steering angle:
  - Right Tilt → Positive PWM → Right Turn
  - Left Tilt → Negative PWM → Left Turn
  - Neutral Position → No Steering Change
- The servo motor responds to these commands and adjusts its position accordingly.

Challenges and Solutions:

- Noise in accelerometer readings: Applied low-pass filtering to smoothen sensor data.
- Latency in response: Used RTOS for real-time sensor polling and immediate action execution.


Start
Activate ECU
Activate Steering Sensor
Gather Data
Motion Detected?
No
Maintain steer
Yes
No
Is Left?
No
Is Right?
Yes
Yes
Steer Left
Steer Right
No
No
Steer Return Normal
Steer Return Normal
Yes
Yes
Return steer normal
Return steer normal

4. Implementation of RTOS for Task Management

A Real-Time Operating System (RTOS) was implemented to handle multiple tasks efficiently. Each task was assigned a priority to ensure critical operations (braking, acceleration) had higher precedence over non-critical operations (data logging).

RTOS Implementation Techniques:

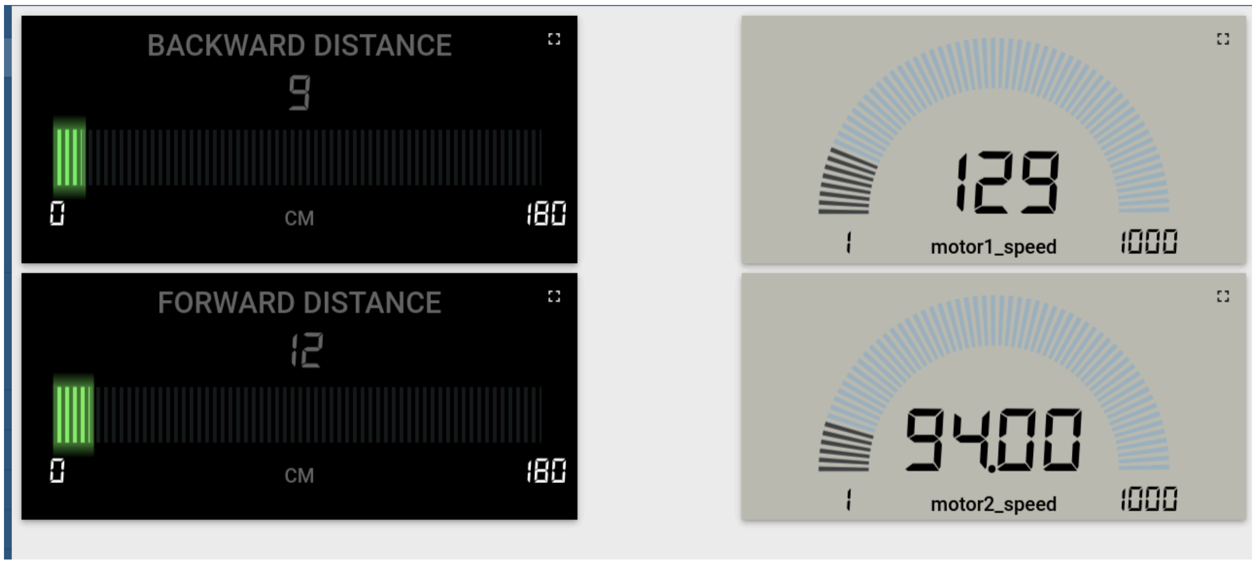
- Task Synchronization: Used semaphores to avoid resource conflicts.
- Task Preemption: Ensured critical tasks interrupt lower-priority tasks if needed.
- Inter-Task Communication: Used message queues to pass data between tasks.

Task Priority Setup:

Task	Priority Level	Description
Ultrasonic Sensor Processing	High	Continuously measure distance and prevent collisions
Motor Speed Control	High	Adjust speed based on proximity data
Steering Control (Accelerometer Input to Servo Motor)	Medium	Change lane based on driver input
Data Transmission to Cloud (ESP32 via CAN)	Low	Log data without affecting real-time decision-making

Chapter 5
Results

The system has successfully proven its ability to maintain a safe following distance by dynamically adjusting motor speed based on information obtained from ultrasonic sensors. The lane-changing feature showed high accuracy in reaction to input from accelerometers, thus allowing users to simulate real driving conditions. The cloud-based data logging feature provided a reliable means of tracking vehicle performance, hence ensuring transparency and safety. The use of a Real-Time Operating System (RTOS) significantly improved system efficiency by reducing processing latency and enabling real-time synchronization of tasks.



Chapter 6
Conclusion

6.1 Conclusion

This project is a prime example of the deployment of Adaptive Cruise Control using embedded systems. With the inclusion of STM32, RTOS, CAN, and cloud monitoring, the system successfully proves the viability of intelligent driver assistance systems in modern vehicles. The use of real-time computing ensures accurate control, and the inclusion of a black box feature provides safety through data logging and remote analysis. Future developments can include the incorporation of AI-based decision-making algorithms to further improve adaptive response capabilities.

The prototype can be adapted for scalability, better convergence, and better accuracy.

6.2 Future Enhancement

- Better code optimizations.
- More Accurate reading from sensor, and features of pothole detection.
- AI Based driver’s driving pattern.

Chapter 7
References

[1] IEEE Transactions on Intelligent Vehicles - "Adaptive Cruise Control: A Literature Review"

[2] Embedded Systems Design Handbook - "Real-Time Operating Systems for Embedded Systems"

[3] Journal of Automotive Engineering - "Controller Area Network (CAN) Protocol in Automotive Systems"

[4] STM32 Documentation and User Manual