SYNOPSIS

Report on

Automated Vehicle Control using Trampoline

by

Anchal Tyagi 2300290140023
Akshat Gautam 2300290140005
Anuj Singh 2300290140029
Aakrti Sharma 2300290140001

Session:2024-2025 (IV Semester)

Under the supervision of

Prof. Mr.Amit Kumar Gupta

KIET Group of Institutions, Delhi-NCR, Ghaziabad

DEPARTMENT OF COMPUTER APPLICATIONS
KIET GROUP OF INSTITUTIONS, DELHI-NCR, GHAZIABAD-201206

(2023)

ABSTRACT

In the rapidly advancing field of automotive technology, integrating smart decision-making systems is crucial for ensuring safer and more efficient transportation. This project focuses on designing and implementing a robust decision-making architecture for autonomous vehicles using an OSEK/VDX-based Real-Time Operating System (RTOS) called Trampoline. The system processes inputs from multiple sensors, including proximity sensors, gyroscopic sensors, and pressure sensors, to assess the vehicle's surroundings and generate essential movement parameters: **Acceleration Intensity, Brake Intensity, and Steering Intensity**.

To enable seamless communication between different vehicle subsystems, we utilize the Controller Area Network (CAN) **protocol**. A CAN driver has been implemented for POSIX systems, using Linux Virtual CAN (vCAN) for hardware communication simulation. The proposed framework ensures real-time data processing, allowing vehicles to navigate autonomously while responding dynamically to road conditions. This project serves as a foundation for intelligent vehicular systems, contributing to the broader vision of autonomous and connected transportation

TABLE OF CONTENTS

Introduction

Autonomous driving technology is revolutionizing the transportation industry by improving vehicle safety, efficiency, and decision-making capabilities. A critical aspect of autonomous vehicle systems is the ability to **perceive the environment, process sensor data, and make intelligent driving decisions**. This project focuses on developing a **real-time decision-making system** for automotive vehicles using an **OSEK/VDX-based Real-Time Operating System (RTOS) called Trampoline**.

The system integrates **Proximity Sensors** to gather critical environmental data. The gathered data is processed within the RTOS, generating **three key outputs—Acceleration Intensity, Brake Intensity, and Steering Intensity**—that determine vehicle movement and direction. To facilitate seamless communication between sensors and processing units, we employ the **Controller Area Network (CAN)** protocol, utilizing a **custom CAN driver implemented using Linux Virtual CAN (vCAN)** for hardware simulation.

By implementing this system, we aim to create a **low-cost, reliable, and real-time decision-making framework** that can serve as a foundation for autonomous driving applications.

Literature Review

2.1 Autonomous Vehicle Control Systems

Autonomous vehicles rely on **real-time sensor data processing, advanced control algorithms, and robust communication frameworks** to navigate safely. Previous research in this field has focused on:

- **Sensor Fusion Techniques:** Taking array of sensor's input to improve accuracy in obstacle detection and navigation.
- **Decision-Making Algorithms:** Machine learning and rule-based logic systems for determining vehicle actions.
- **RTOS in Automotive Systems:** Real-time operating systems (RTOS) provide **deterministic task execution** essential for safety-critical applications.

2.2 OSEK/VDX-Based RTOS (Trampoline-Github Repo)

OSEK/VDX (Open Systems and the Corresponding Interfaces for Automotive Electronics) is an **automotive RTOS standard** used for real-time task scheduling and multi-threaded execution. Trampoline is an open-source implementation of **OSEK/VDX** that enables **real-time task execution, resource management, and inter-task communication** in automotive applications.

2.3 Controller Area Network (CAN) in Automotive Applications

The **CAN protocol (ISO 11898)** is widely used in automotive systems to facilitate communication between different vehicle components, such as sensors, actuators, and control units. CAN offers **high-speed, reliable, and collision-free** data exchange, making it ideal for real-time automotive applications.

Project / Research Objective

This app, focus on:

1. Target Audience

The **primary target audience** for this project includes:

- **Automobile Manufacturers & Engineers:** To enhance autonomous vehicle systems.
- **Research & Development Teams:** To explore real-time decision-making frameworks.
- **Automotive Enthusiasts & Developers:** To experiment with smart vehicle navigation.
- **Safety Regulators & Transportation Authorities:** To ensure road safety compliance.

The system aims to support **Level 2 & 3 autonomy**, assisting vehicles in controlled environments while allowing for further enhancements toward full autonomy.

2. Core Features

Our project focuses on **real-time decision-making** in vehicles by integrating multiple sensors and an RTOS-based control system.

Key Features:

🔷 **Obstacle Detection:** Using **Proximity Sensors** to identify objects in front, sides, and back of the vehicle.

🔷 **Lane Monitoring:** Sensors determine lane boundaries and prevent unintended lane departure.

🔷 **Real-Time Motion Control: Acceleration, Brake, and Steering Intensity** are dynamically

calculated based on sensor data.

🔷 **Gyro-Based Slope Detection:** Adjusts vehicle control when going **uphill or downhill**.

🔷 **CAN-Based Communication:** Uses **Controller Area Network (CAN)** for seamless data exchange between vehicle components.

🔷 **RTOS-Driven Task Execution:** Ensures efficient, **low-latency decision-making** with **Trampoline RTOS**.

3. Tech Stack

The project is built using a **combination of hardware and software technologies** designed for efficiency, real-time execution, and automotive-grade reliability.

Category	Technology/Tool	Purpose
RTOS	**Trampoline (OSEK/VDX)**	Real-time task scheduling
Programming	**C/C++**	Sensor drivers & control logic
Sensors	**Proximity, Gyro, Pressure**	Data collection
Communication	**CAN (Controller Area Network)**	Vehicle component communication
Simulation	**Linux Virtual CAN (vCAN)**	CAN bus simulation

Embedded Development	**Microcontroller (ARM-based)**	Real-time processing

4. Security Considerations

Since the project involves vehicle control, **security is critical** to prevent unauthorized access or data manipulation.

🔐 **Secure Communication:**

- **CAN Message Authentication**: Ensuring only valid data is accepted.
- **Data Encryption**: Protecting sensor data from unauthorized access.
- **Error Detection in CAN Bus**: Identifying and preventing corrupt data packets.

🔐 **System Protection:**

- **Task Isolation in RTOS**: Ensuring one faulty task doesn't crash the system.
- **Fail-Safe Mode**: If an error is detected, the system shifts to **manual control or emergency stop**.

5. Scalability

Our architecture is designed to be **modular and scalable**, allowing for **future enhancements and new sensor integrations**.

🚀 **Modular Architecture:**

- Additional **sensors** (e.g., LIDAR, Radar) can be integrated easily.
- The **decision-making algorithm** can be upgraded with **AI-based models**.
- The system can be adapted for **different vehicle types** (cars, trucks, autonomous shuttles).

🚀 **Cloud Integration Possibilities:**

- The system can be extended to send real-time vehicle data to a **cloud dashboard** for monitoring.
- OTA (Over-the-Air) **updates** can improve software functionality without manual intervention.

6. User Experience (UX)

Since the project is focused on **autonomous decision-making**, user experience is determined by **safety, reliability, and responsiveness**.

✔️ **Smooth & Predictable Vehicle Movements** – Ensures comfortable driving behavior.

✔️ **Minimal Delays in Response Time** – Optimized real-time execution with RTOS.

✔️ **Clear Communication & Alerts** – CAN-based messaging ensures vehicle status updates.

✔️ **Failsafe Mechanism** – If sensors fail, the system **notifies the driver or applies emergency controls**.

Hardware and Software Requirements

For developing and running the **Smart Decision-Making System for Automotive Vehicles**, the following hardware and software requirements are essential:

Hardware Requirements:

Development Machine:

- **Processor:** Intel i5 or higher.
- **RAM:** Minimum 8 GB.
- **Storage:** SSD with at least 256 GB for faster operations.
- **Internet:** Reliable internet connection for testing, version control (Git), and deployment.

Software Requirements:

Development Tools:

- **Operating System:** Linux (Ubuntu 20.04 or later) or Windows 10/11.
- **Code Editor:** Visual Studio Code, Eclipse
- **Version Control:** Git with GitHub/GitLab for collaboration and source control.
- **Compiler & Debugging Tools:** GCC toolchain for compiling C code.

Embedded Development:

- **RTOS:** Trampoline RTOS (OSEK/VDX-based) for real-time scheduling.
- **Programming Language:** C/C++ for writing sensor drivers, control logic, and decision-making algorithms.

Communication & Simulation:

- **CAN Communication:**
  - SocketCAN (Linux-based CAN framework) for real-time communication.
  - CAN utilities (cansend, candump, etc.) for debugging CAN messages.
- **Testing & Debugging:**
  - Linux Virtual CAN (vCAN) for testing CAN communication without physical hardware.

Project Flow for Automated Vehicle Control using Trampoline

1. **Requirement Gathering**:

a. Identify key features: obstacle detection, lane monitoring, real-time vehicle control.
b. Determine essential sensors: proximity, gyro, pressure sensors for environment sensing.
c. Establish communication protocol: **CAN Bus** for inter-device communication.
d. Define the real-time processing needs: using **Trampoline RTOS** for efficient task management.

2. **Planning**:

a. Define project scope: **sensor-based vehicle navigation and autonomous decision-making system**.
b. Break the development into phases:
    - **Phase 1:** Sensor data collection and CAN communication.
    - **Phase 2:** Real-time decision-making algorithm implementation.
    - **Phase 3:** Integration of acceleration, braking, and steering control logic.
c. Choose the technology stack:
    - **Hardware:** ARM-based microcontroller, proximity sensors, gyro sensors.
    - **Software: C/C++, Trampoline RTOS, CAN communication framework**.
d. Set up timelines and development milestones.

3. **Design**:

a. **System Architecture Design:** Define how sensors interact with the **RTOS and CAN bus**.
b. **Data Flow Design:** Outline how sensor inputs are processed to generate acceleration, braking, and steering outputs.
c. **Task Scheduling Design:**
    - Define tasks for **sensor data acquisition, decision processing, and control signal generation**.

    - Optimize RTOS scheduling for minimal latency.
d. **Hardware Design:** Identify **sensor placements and microcontroller configuration**.

4. **Development**:

a. **Sensor Integration:** Write drivers for **proximity, gyro, and pressure sensors**.
b. **CAN Communication Implementation:** Develop and test **CAN drivers for real-time data exchange**.
c. **Decision-Making Algorithm:** Implement logic to calculate:
    - **Acceleration Intensity (1 to 10)**.
    - **Brake Intensity (1 to 10, based on obstacle proximity)**.
    - **Steering Intensity (1 to 10 for left, -1 to -10 for right turns)**.
d. **RTOS Task Management:** Schedule tasks for:
    - **Sensor data collection**.
    - **Decision processing**.
    - **Control signal execution**.
e. **Testing in Virtual Environment:**
    - Use **Linux Virtual CAN (vCAN)** to simulate hardware communication.

5. **Testing**:

a. **Unit Testing:** Validate **sensor readings, CAN communication, and control outputs** separately.
b. **Integration Testing:** Verify interaction between **RTOS, sensors, and decision-making system**.
c. **Fault Handling & Safety Checks:** Ensure proper **error detection in CAN communication and sensor failures & check some edge cases.**

6. **Integration**:

a. **Deploy firmware with the integration on UNITY for better visualization or can have the decisions on framework.**

Project Outcome

The successful implementation of this project will result in:

1. **A real-time decision-making system** capable of autonomous vehicle control.

2. **Seamless CAN-based communication** between sensors and control units.

3. **A scalable RTOS framework** that can be extended for further autonomous driving applications.

4. **A simulated environment (Linux vCAN)** to test and refine the system before real-world deployment.

5. **A research foundation** for advanced intelligent vehicle navigation and control.

Proposed Time Duration

The project is planned to be completed in approximately 3 months, divided into different phases to ensure systematic development, testing, and Integration.

Month 1: Requirement Analysis & Planning

- Define project scope, objectives, and core features.
- Identify required hardware components (sensors, microcontroller, communication modules).
- Research on **Trampoline RTOS** and **CAN Bus communication** for implementation.

Month 2: System Design & Architecture

- Design system flow, sensor data file, and decision-making logic.
- Develop initial integration plan for **sensor connectivity and data processing**.
- Set up **RTOS environment**, create necessary **OIL files**, and configure task scheduling.

Month 2: Development Phase II (Decision-Making & Control Algorithms)

- Implement algorithms for **Acceleration Intensity, Brake Intensity, and Steering Intensity**.
- Integrate **RTOS task scheduling** for real-time decision execution.
- Conduct **unit testing** of individual system components

Month 3 : Integration & Final Optimization

- Integrate the software on UNITY platform for better visualisation.
- Improve system efficiency based on real-world performance data.
- Prepare project documentation and final reports.

References/ Bibliography

1. Trampoline RTOS Documentation –Github Repository
2. OSEK/VDX Standard for Automotive RTOS – OSEK Specification
3. CAN Bus Communication Protocol – ISO 11898 Standard
4. Linux Virtual CAN (vCAN) – Kernel Documentation
5. Research Papers on Autonomous Vehicle Decision Making