Automated Vehicle Control using Trampoline RTOS

-Anchal Tyagi

Introduction

Cars are becoming smarter, and technology is helping them make better driving decisions. In this project, we are building a software-based real-time system for automated vehicle control using Trampoline RTOS. Instead of using actual sensors, we simulate sensor data (like obstacle detection, road conditions, and vehicle movement) to test how the system responds.

Our system:

✅ Processes simulated sensor inputs (distance to obstacle, acceleration)

✅ Determines the best driving actions (acceleration, braking, steering)

We tested critical edge cases in a Linux-based virtual environment, ensuring real-time decision-making without the need for actual hardware.

Literature review

What is a Real-Time Operating System (RTOS)?

An RTOS (Real-Time Operating System) is a special type of operating system designed to handle tasks quickly and efficiently. It ensures that important tasks are completed on time, which is very important for systems like car control, where delays can be dangerous.

Why we used OSEK/VDX RTOS ?

- OSEK/VDX is a standard RTOS used in the automotive industry.
- It is able to handle multiple tasks at the same time (like braking, steering, and acceleration).
- It ensures fast and reliable responses, making driving safer.

Literature review

Comparison with Other RTOS Solutions

- Many cars use other RTOS like FreeRTOS, VxWorks, or QNX, but they may not be automotive-specific like OSEK/VDX.
- Trampoline RTOS follows OSEK/VDX standards, making it ideal for car control systems.

Why Choose Trampoline RTOS?

✅ Lightweight and fast – Works efficiently with limited system resources.
✅ Open-source – Can be modified and tested freely.
✅ Supports real-time scheduling – Ensures car responses are quick and accurate.
✅ Compatible with CAN Bus – Makes communication between different parts of the system easy. (can integrate with CAN bus as edge cases passed)

By using Trampoline RTOS, we can simulate smart vehicle control in a safe and efficient way without needing actual hardware. 🚗💨

Objective of the project

Main Goal:

- Develop a smart decision-making system for vehicle control using Trampoline RTOS.

Key Functions:

- ✅ Simulate sensor data to detect obstacles and road conditions.
- ✅ Regulating acceleration, braking and steering based on input data.
- ✅ Use real-time task scheduling to ensure quick and accurate responses.
- This project helps us understand how an RTOS can manage multiple tasks efficiently to improve vehicle safety and automation. 🚗💡

Technology Used

Our project is developed using the following technologies:

✅ **Operating System:** Ubuntu 24.04 (Linux-based) – Used as the development environment.

✅ **RTOS:** Trampoline (OSEK/VDX Compliant) – A real-time operating system designed for automotive applications.

✅ **Programming Language:** C – Used to write the system logic and GOIL for task scheduling.

Technology Used

✅ **Development Tools:**

- GCC Compiler – Compiles the C code for real-time execution.
- Make & CMake – Helps in building and managing the project efficiently.
- Git – Version control to track changes and manage project development.
- Python – Used for OIL file processing and build automation (goil and make.py).
- Linux Virtual CAN (vCAN) – Simulates CAN bus communication for real-time testing. (integrate as edge cases passed)
- OSEK/VDX Framework – Provides the real-time operating system architecture for automotive applications. (trampoline)

Hardware Requirements

For running our project, we need the following hardware:

✅ **Processor (CPU):** Intel or AMD with virtualization support (for running virtual machines).

✅ **Memory (RAM):** At least 4GB (8GB recommended for better performance).

✅ **Storage:** At least 20GB of free space to install and run the system.

Since our project is software-based, we simulate sensor data instead of using real sensors. However, in a real-world scenario, it would use:

- Proximity Sensors – To detect nearby obstacles.
- Gyro Sensor – To check if the vehicle is on a slope.
- Pressure Sensors – To determine braking and acceleration force.

Software Requirements

To develop and test our project, we need the following software:

✅ Ubuntu 24.04 (Virtual Machine) – A Linux-based system where we develop and run the project.

✅ Trampoline RTOS (Source Code from GitHub) – The real-time operating system we use for task scheduling and vehicle control logic.

✅ GCC Compiler & Make – Tools used to write, build, and compile the C code for our system.

✅ Git (Version Control) – Helps track changes and manage the project code efficiently.

Note: We are not linking CAN communication right now priority is to setup trampoline on VM and pass all manually created possible edge case, so we do not need CAN utilities (can-utils) at this stage.

System Architecture

- Input Devices:
  - Front, Left, and Right Proximity Sensors.
  - Gyro Sensor or proximity to detect car rotation (uphill/downhill).
- Processing:
  - Trampoline RTOS schedules tasks for data processing.
  - Uses preemptive priority-based scheduling.
- Generated Output: (to be acutate by actuators in real world)
  - Acceleration Intensity: Controls the vehicle’s speed based on sensor data.
  - Brake Intensity: Regulates braking force based on obstacles distance.
  - Steer Intensity: Adjusts left/right turns.

Modules

5 elements of sensors, 2 of left and right ,2 for offroading

1.Steering Module 🚗🔄: (−10)−(+10)

2. Braking Module 🛑: On the scale of 0−10

3.Acceleration Module ⚡🚀: On the scale of 0−10

Some Considerations:

OBSTACLE_THRESHOLD 50
Car_width 2.5
Road_width 9
min_lr_road 0.5 (for offroad conditions)
lr_sensor_threshold=2.5; (for obstacle in left/right)

Sample input:

sensor_data[9]={50,30,45,15,60,3,1,3,3}; //{5 elements of 5 sensors, 2 of left and right ,2 for offroading}

To_Steer=0; (inital state of steering)
max_left_steer=-10; (Left)
max_Right_Steer=10; (right)

braking=0;
max_braking=10;
acceleration=10;
max_aceleration=10;

Workflow

Our system follows a step-by-step process to simulate vehicle control using Trampoline RTOS.

Workflow Steps:

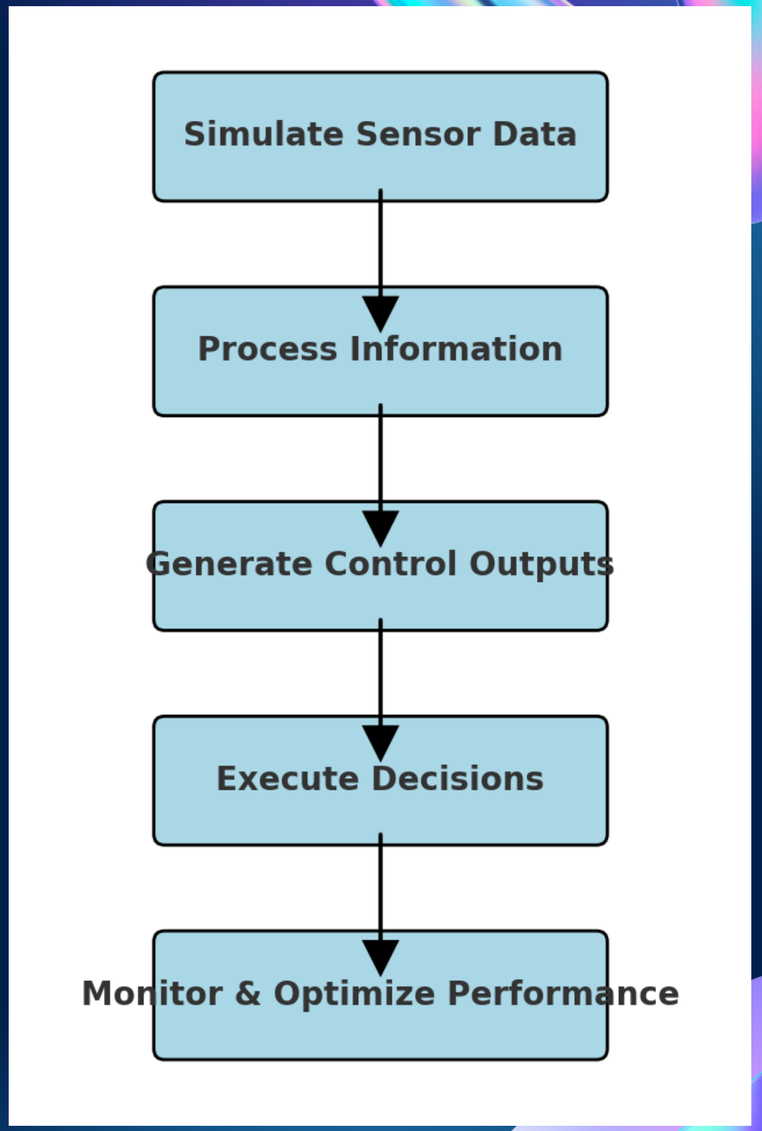
Simulate Sensor Data – Instead of using real sensors, we generate virtual sensor data to represent obstacles and road conditions.

Process Information – The RTOS analyzes the data and decides what action to take.

Generate Control Outputs – Determines if the car should accelerate, brake, or steer.

Execute Decisions – The system applies the calculated actions in real time.

Monitor Performance – Logs data and optimizes future decisions for better results.



References

📌 Sources Used in the Project:

✅ Trampoline RTOS Documentation – Guide for implementing the RTOS.

✅ OSEK/VDX Standards – Framework for real-time automotive systems.

✅ Research Papers on RTOS – Insights into real-time scheduling and vehicle control.

✅ Open-source Contributions – Community support and development resources.

References

- **Reference Papers:**
- https://ieeexplore.ieee.org/document/7295845
- https://github.com/TrampolineRTOS/trampoline/blob/master/documentation/manual/main.pdf
- https://github.com/TrampolineRTOS/trampoline/blob/master/README.md

- **Reference links:**
- https://github.com/TrampolineRTOS/trampoline.git
- https://www.researchgate.net/publication/221593418_Effect_Analysis_of_the_Introduction_of_AUTOSAR_A_Systematic_Literature_Review

Thank You