UNIVERSITÀ DEGLI STUDI DI L'AQUILA

DISIM
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Autonomous Driving Car

Department of Information Engineering, Computer Science and Mathematics

University Of L'Aquila, Italy

Professor Davide Di Ruscio

Team Members

Alaina Faisal
Sarosh Krishan

Contents

0.1 Introduction

In this report, we detail our project using the Carla simulator to develop an autonomous car prototype. Our car, equipped with a lidar sensor and functioning as an autonomous agent, navigates a simulated environment and avoids obstacles using the MAPE-K loop (Monitoring, Analysis, Planning, Execution, Knowledge). This loop helps the car make decisions in real-time.

We've also incorporated MQTT to facilitate data sharing. This protocol sends data from the car to an InfluxDB database via Node-RED, allowing us to store and analyze large volumes of telemetry data efficiently.

This document will cover the setup and implementation of our system, our project goals, the strategies we developed for adaptation, and a performance analysis. We will conclude with key takeaways and instructions for setting up the system locally.

0.2 Goals of the Project

The goals of our project are:

- **Sensor Integration and Data Processing:** To fully integrate and optimize the use of multiple sensor modalities including LIDAR, cameras, and gps.
- **Autonomous Navigation Algorithm:** To develop and refine autonomous navigation algorithms that can effectively handle complex driving scenarios.
- **Realistic Traffic and Environmental Simulation:** To utilize Carla's dynamic simulation capabilities to create traffic conditions and environmental factors.
- **Performance Metrics and Safety Standards:** To establish a set of comprehensive performance metrics that autonomous vehicles must meet or exceed during simulations.

0.3 Adaptation Goals

GOALS	DESCRIPTION	EVALUATION METRIC
Maintain a safe distance from obstacles using LIDAR	The system must detect obstacles within the defined threshold and adjust navigation accordingly.	Limit: Distance $>$ **1**
Identify vehicles as obstacles	The system must determine if an obstacle is another vehicle based on proximity and respond accordingly.	Limit: Distance $>$ **10**
Ensure vehicle clearance within the planned route	The vehicle should avoid entering zones occupied by detected obstacles.	Limit: Clearance $>$ **0.8**
Prevent collisions with dynamic objects	The system must recognize moving obstacles and calculate a safe path to avoid them.	Danger limit: Obstacle distance $>$ **1** triggers evasive action

Table 1: Obstacle Detection Goals and Thresholds

0.4 Functional Requirements

Identifier	Name	Description
FR1	Autonomous Navigation	The system employs the Carla simulator to enable an autonomous vehicle to navigate through a virtual environment autonomously. This includes starting from an initial point, following a predetermined path, and arriving at the destination safely without human intervention.
FR2	Obstacle Detection and Avoidance	With the integration of a lidar sensor, the autonomous agent is capable of detecting and maneuvering around obstacles, ensuring the vehicle can adapt to changes in the environment such as unexpected obstructions or alterations in the terrain.
FR3	Data Collection and Management	Through the use of MQTT and Node-RED, the system captures, publishes, and stores telemetry data from the autonomous agent into an InfluxDB database, facilitating both real-time and historical data analysis.
FR4	MAPE-K Loop Implementation	The system is structured around the MAPE-K loop (Monitoring, Analysis, Planning, Execution, Knowledge), enabling the autonomous agent to continuously monitor its surroundings, analyze data, plan its route, execute movements, and update its knowledge base based on experiences.
FR5	Real-Time Updates	The system provides real-time updates on the vehicle's status, including its current location, movements, and any incidents such as deviations from the route or crashes.

Table 2: Functional Requirements of the Autonomous Vehicle System

0.5 Non-Functional Requirements

Identifier	Name	Description
NFR1	Performance	The system must process data and react to environmental changes in real-time. Performance metrics include response time for detecting and avoiding obstacles, and data processing speeds.
NFR2	Reliability	Ensuring high reliability is critical for the vehicle to consistently perform its designated functions under different conditions without failure.
NFR3	Scalability	The architecture should be scalable, capable of handling increases in environmental complexity or additional sensors without significant degradation in system performance.
NFR4	Security	Since the system utilizes MQTT for data transmission, securing data transfers to prevent unauthorized access and ensuring data integrity is imperative.
NFR5	Maintainability	The code should be modular and well-documented, making it easier for future updates, maintenance, and troubleshooting.
NFR6	Usability	The system should include a user-friendly interface for monitoring and interacting with the autonomous agent, possibly incorporating tools like Grafana for data visualizations.

Table 3: Non-Functional Requirements of the Autonomous Vehicle System

0.6 Technologies Used

The project integrates the following technologies:

0.6.1 Carla Simulator

The **Carla Simulator** serves as the backbone of our autonomous driving project by providing an open-source simulator for driving. It offers realistic scenarios, enabling detailed testing of autonomous vehicle behaviors in a controlled yet dynamic setting. The simulator allows for the integration of various sensors, providing a comprehensive platform for testing of the autonomous agent's capabilities in navigating complex landscapes.

0.6.2 Node-RED

Node-RED provides a visual programming environment to wire together hardware devices, APIs, and online services. In our project, Node-RED acts as an intermediary to process and route data from the MQTT protocol to our InfluxDB database, enabling flexible and efficient data handling, transformation, and flow control, which are essential for managing complex data streams in real-time.

0.6.3 MQTT (Mosquitto)

The **MQTT Protocol** is a lightweight messaging protocol designed for situations where a small code footprint is required and network bandwidth is limited. In our project, MQTT facilitates reliable, real-time communication between the autonomous agent and the backend systems.

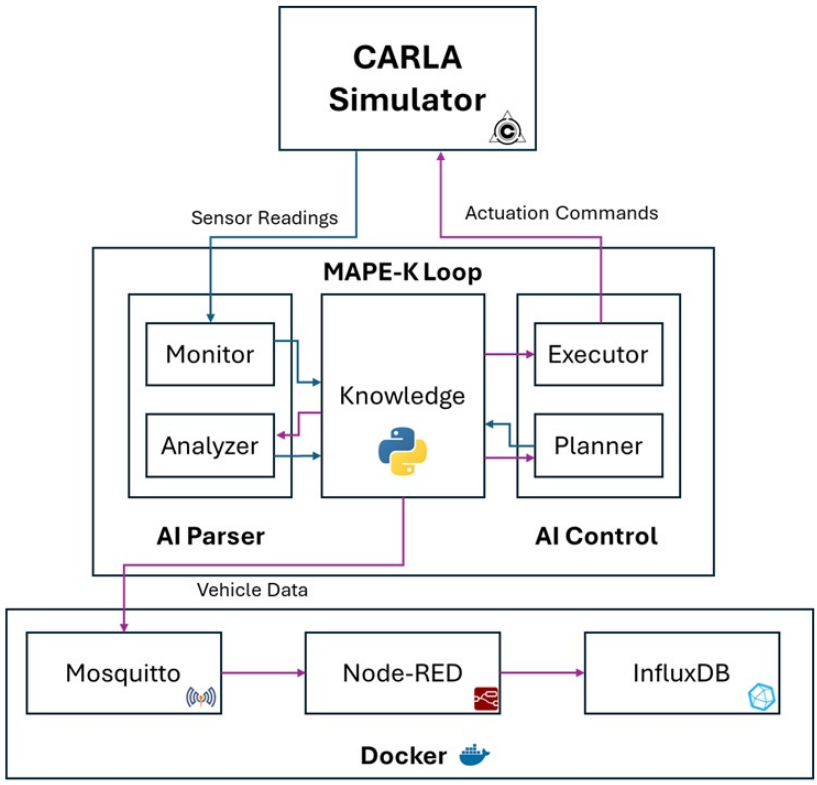
0.6.4 InfluxDB

InfluxDB is a time-series database designed to handle high write and query loads, making it ideal for storing and analyzing large volumes of telemetry data generated by our autonomous agent. It provides powerful querying capabilities and real-time analytics, which are crucial for monitoring the agent's performance and making data-driven decisions to optimize operations.

0.7 System Architecture

The architecture of our autonomous driving simulation system is designed to facilitate development of autonomous vehicle behavior within a virtual environment provided by the Carla Simulator. The system is composed of several interconnected components that work together to mimic real-world autonomous driving scenarios. At the core of this architecture is the **Autonomous Agent**, which acts as the virtual vehicle equipped with advanced

sensors like lidar. This agent is responsible for navigating through the simulator, processing sensor data, and making driving decisions based on the environment and obstacle detection.



The **Lidar Sensor** is pivotal in capturing detailed three-dimensional data about the agent's surroundings, which is processed using the MAPE-K loop for dynamic decision-making. The data flow starts with the MQTT protocol, which facilitates the transmission of sensor data to the backend systems. **Node-RED** acts as the intermediary, efficiently processing and routing this data into the **InfluxDB database** for storage and further analysis.

0.7.1 MAPE-K Loop

The MAPE-K loop (Monitoring, Analysis, Planning, Execution, Knowledge) is the cornerstone of our autonomous agent's operational framework, enabling it to behave adaptively and learn from its interactions with the environment. The loop functions as follows:

Monitoring: The lidar sensors continuously monitor the surrounding environment, capturing spatial and obstacle data in real time.

Analysis: This data is analyzed to assess current conditions and identify potential hazards or navigational obstacles.

Planning: Based on the analysis, the system plans the best course of action, such as adjusting the route to avoid obstacles.

Execution: The planned actions are executed by the autonomous agent, involving steering, accelerating, or braking maneuvers to navigate the environment safely.

Knowledge: Outcomes from executed actions are fed back into the system as new data, updating the knowledge base. This updated knowledge influences future monitoring and analysis, enhancing the agent's performance and adaptability over time.

0.8 Components Implementation

0.8.1 Monitor

The **Monitor** component is used for continuously tracking the state and environment of the autonomous vehicle within the Carla simulation. It interacts directly with the vehicle and various sensors attached to it, including lane detection and collision sensors, and a LIDAR sensor. This component is responsible for monitoring the vehicle's surroundings and its interactions with the environment.

Upon initialization, the Monitor subscribes to data streams from the lane detection sensor and the collision sensor, both attached to the vehicle. These sensors provide real-time data about the vehicle's interaction with lane markers and other objects, respectively. When the lane detection sensor detects that the vehicle has crossed a lane marker, or when the collision sensor detects a collision, the Monitor updates the **Knowledge** database with this information, ensuring that the vehicle's status is continuously updated in response to environmental interactions.

Additionally, the Monitor sets up a LIDAR sensor that provides detailed point cloud data, which is processed to gather comprehensive environmental data around the vehicle. This data is then transformed into a structured format and stored in the Knowledge database for further analysis and decision-making processes. The LIDAR data helps in creating a precise spatial understanding required for navigating safely through the environment.

The Monitor also checks if the vehicle is at a traffic light and updates the traffic light status in the Knowledge database. This functionality aids in compliance with traffic rules and enhances the safety and reliability of the vehicle's autonomous navigation capabilities.

0.8.2 Analyzer

The **Analyzer** component plays a critical role in processing and interpreting the data collected by the Monitor component, particularly focusing on obstacle detection and traffic

light analysis within the Carla simulation environment. It is essential for ensuring that the autonomous vehicle navigates safely by identifying potential hazards in real-time and responding appropriately.

Upon initialization, the Analyzer is configured with thresholds for detecting obstacles and vehicles. It leverages LIDAR data provided by the Knowledge database to perform detailed analyses of the surrounding environment:

- **Obstacle Detection:** The Analyzer processes the LIDAR data to detect physical obstacles within predefined distance thresholds. If an obstacle is detected closer than the set threshold, it is logged and managed accordingly. This component uses the Euclidean distance formula to determine the proximity of detected objects, enhancing the vehicle's ability to react to immediate environmental changes.
- **Vehicle Detection:** In addition to stationary obstacles, the Analyzer is capable of identifying other vehicles in close proximity to the autonomous agent. This is crucial for preventing collisions and managing navigation in traffic-heavy environments.
- **Traffic Light Analysis:** The Analyzer also assesses the traffic light state to ensure compliance with traffic laws. If a red light is detected, the system updates its status to reflect this, pausing vehicle movement as required.

The Analyzer continuously updates the Knowledge database with new findings, including the presence of obstacles, vehicle detection, and traffic light states. This dynamic data flow ensures that the autonomous vehicle's decision-making process is informed by the most current environmental data, allowing for real-time adjustments to driving strategies.

This component also includes a function that periodically updates the vehicle's AI state based on the analyzed data. If a crash is detected, further analysis is halted to prioritize safety responses. Through its comprehensive analysis capabilities, the Analyzer ensures that the autonomous system remains aware and responsive to its surroundings, maintaining operational safety and efficiency.

0.8.3 Planner

The **Planner** component is integral to the autonomous navigation system, primarily responsible for generating and managing navigational paths for the vehicle within the Carla simulation environment. This component interacts closely with the **Knowledge** database to retrieve and update navigation-related information, ensuring that the vehicle follows optimal routes while adapting to dynamic environmental conditions.

The Planner's primary function is to create a comprehensive map of waypoints from a given source to a destination. This is achieved through the following key functions:

- **Path Generation:** The Planner utilizes Carla's map API to generate a sequence of waypoints that the vehicle must follow to reach its destination. It employs a

waypoint-based pathfinding approach, dynamically selecting navigable routes while ensuring obstacle-free movement. The path is continuously updated based on real-time vehicle position and environmental changes.

- **Obstacle Avoidance and Detour Calculation:** If an obstacle is detected within a predefined threshold (3.0 meters), the Planner attempts to compute an alternative route by evaluating detours to the left, right, or around the obstacle. This ensures the vehicle can navigate efficiently even in dynamic environments.
- **Real-Time Route Updates:** The Planner dynamically modifies the route based on real-time sensor inputs. As the vehicle progresses, waypoints are dequeued, and new waypoints are added if environmental changes necessitate path adjustments.
- **Lane Following and Adaptation:** The system ensures the vehicle remains within valid road lanes by utilizing Carla's built-in lane detection API. If the destination is in another lane, the Planner evaluates whether a safe lane change is required.
- **Traffic Light Handling:** The Planner integrates traffic signal detection by querying the Knowledge component for real-time traffic light states. If a red light is detected, the vehicle halts and resumes navigation once the signal allows movement.
- **Obstacle Detour Calculation:** In scenarios where the direct path is obstructed, the Planner calculates detours. This involves dynamically assessing the space around the obstacle and proposing alternative routes to bypass the obstruction, ensuring continuous movement towards the destination.

During each update cycle, the Planner checks the status of the vehicle and the presence of any obstacles along the path. If an obstacle is detected within a critical range, the Planner either adjusts the path to include a detour or stops the vehicle, depending on the situation. This decision-making process is supported by continuous feedback from the Knowledge component, which provides real-time data about the vehicle's environment and status.

0.8.4 Executor

The **Executor** component is the operational core of the autonomous vehicle, directly controlling the vehicle's maneuvers based on the navigational plans and environmental data provided by the Planner and Analyzer components. This component ensures that the vehicle's actions align with the intended navigation paths and responds appropriately to dynamic conditions and obstacles.

The Executor performs several critical functions:

- **Driving Control:** When in the DRIVING or HEALING state, the Executor retrieves the current destination from the Knowledge database and calculates the appropriate vehicle control inputs. This includes steering directions, throttle, and braking commands to navigate towards the destination efficiently and safely.

- **Handling Traffic Lights:** In the event of a REDLIGHT status, the Executor applies brakes to bring the vehicle to a complete stop, ensuring compliance with traffic laws.
- **Crash Response:** Upon detecting a CRASHED status, the Executor immediately halts the vehicle to minimize further damage and ensure safety. This is achieved by applying the brakes and disabling any further throttle input.

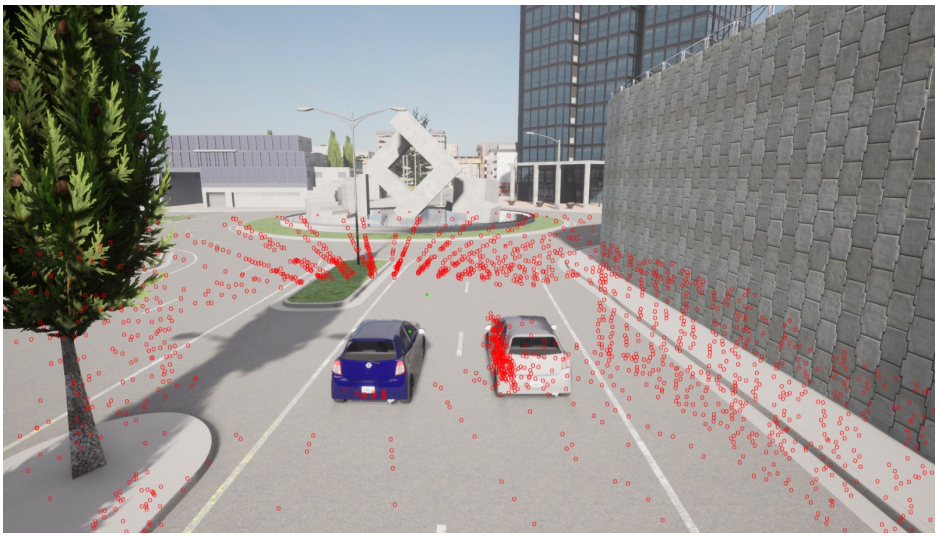
The control logic includes detailed calculations for throttle based on the desired speed versus the current speed, and steering adjustments based on the vehicle's orientation relative to its destination. The vehicle's current position and the destination are converted into vectors, and the steering direction is determined by the cross product of these vectors, providing a measure of how much the vehicle needs to turn to align with the target path.

This dynamic response system allows the Executor to adapt the vehicle's behavior in real-time to various scenarios, ensuring that navigation is not only efficient but also adheres to safety standards. The Executor's ability to adjust controls dynamically is pivotal for maintaining smooth operation within the simulated environment, particularly in responding to unforeseen obstacles or emergency situations.

0.9 Managed Resources

The Managed Resources section of our project encapsulates the vital components that enable the efficient operation of the autonomous vehicle within the Carla simulator. This section highlights how each element contributes to creating a robust and realistic simulation environment, facilitating comprehensive testing and development of autonomous driving technologies.

0.9.1 Simulation Environment



The Carla simulator provides a dynamic environment for testing autonomous vehicles. It features:

- **Weather Variability:** Capabilities to simulate various weather conditions, such as rain, fog, and different lighting conditions, which impact vehicle sensors and driving algorithms.
- **Pedestrian and Vehicle Traffic:** Dynamic simulation of pedestrian and vehicle traffic, which introduces real-world unpredictability into testing scenarios.

0.9.2 Sensor Management

The autonomous vehicle is equipped with a variety of virtual sensors that enable it to perceive and interact with the simulated environment. These sensors play a crucial role in navigation, obstacle detection, and decision-making:

- **LIDAR:** Captures high-resolution 3D spatial data, providing 360-degree environmental awareness. It is essential for detecting obstacles, measuring distances, and mapping the surroundings.
- **Camera:** Provides visual data for scene understanding, lane detection, and object recognition. This data is crucial for interpreting road signs, detecting traffic lights, and identifying other vehicles or pedestrians.
- **GPS:** Supplies real-time position data, allowing the vehicle to track its location within the simulated environment and follow predefined navigation routes.
- **Gyroscope:** Measures angular velocity and orientation changes, helping the vehicle maintain stability and detect changes in movement or direction.

0.9.3 Vehicle Control Systems

The autonomous vehicle's control systems in Carla are managed through direct inputs that simulate real vehicle controls:

- **Autonomous Navigation Algorithms:** Utilize the data from the vehicle's sensors to dynamically navigate through the simulated environment, adjust to traffic conditions, and avoid obstacles.

0.9.4 Data Handling and Communication

- **MQTT Protocol:** Facilitates efficient communication within the simulator environment, enabling real-time data exchange between the vehicle's sensors and its control systems.

- **Data Recording and Analysis:** Essential for refining algorithms and enhancing the vehicle's autonomous capabilities based on test results and performance in diverse scenarios.

0.10 Conclusion

In this project, we developed an autonomous vehicle system using the Carla simulator, integrating multiple sensors and control algorithms to enable autonomous navigation, obstacle avoidance, and real-time data management. The implementation follows the MAPE-K loop, ensuring a structured and adaptive approach to monitoring, analyzing, planning, and executing driving decisions based on real-time environmental feedback.

The system effectively utilizes LIDAR for obstacle detection, MQTT for efficient data transmission, and InfluxDB for data storage and retrieval. Each component, including the Monitor, Analyzer, Planner, and Executor, works in a coordinated manner to facilitate smooth and safe vehicle operations. Through real-time data processing, decision-making, and dynamic path planning, the system successfully demonstrates the core functionalities of autonomous driving.

Overall, this project provides a strong foundation for testing and advancing autonomous driving technologies. The Carla simulator proved to be an effective platform for evaluating different scenarios and refining autonomous vehicle behavior in a controlled virtual environment. Future work can focus on using AI to train deep learning models for better collision detection and finding shortest path.

0.11 Instructions

0.11.1 Prerequisites

- **Docker** must be installed.
- **CARLA Simulator** must be installed.

0.11.2 Steps to Run the Project

1. **Start CARLA Simulator**
    - Navigate to your CARLA installation folder.
    - Double-click on `Carla.exe` (or run `./CarlaUE4.sh -opengl` on Linux/macOS).
    - Wait for CARLA to fully load.
2. **Clone the Repository (If Not Already Cloned)**
    - Open a terminal and run:

```
git clone https://github.com/sarosh2/Autonomous_Project_CARLA.git
```

3. **Build and Run Docker Services**
   - Run the following command to build the necessary Docker images:
     ```
     docker compose build
     ```
   - Start all the services:
     ```
     docker compose up
     ```
4. **Run the AI Test Script**
   - Open a new terminal.
   - Run the following command to execute the AI test for a specific milestone:
     ```
     python MAPE-K_Loop/ai_test.py -m <milestone_number>
     ```
   - Replace `<milestone_number>` with the number of the milestone you want to run.