1 Introduction

This report describes how the team initiated the project work, what was implemented during the reporting period, and what is proposed for completion this month. The team was highly motivated and enthusiastic from the first day of the project. During the work, several technical and organizational challenges were encountered; however, these were addressed as the project progressed. As an initial step, the laboratory was equipped with the necessary tools and resources required to support the planned activities and ensure efficient execution.

2 Planned activities

Research:
We began with a technical review of the overall system. The first step was to understand how the system operates, with a focus on the communication flow between the NUCLEO and the Brain. More specifically, we investigated how the embedded system communicates with the Brain and how the Brain communicates back to the embedded system.

Environment preparation:
We planned to run the simulator, validate that it works correctly, and test a small script within it. In parallel, we planned hardware improvements with emphasis on mechanical components. We also planned to 3D print the first traffic signs and purchased the semaphores, with the intention of making them operate via the server.

Development:
We planned to send basic commands to the NUCLEO from a Raspbian terminal using Minicom, a text-based terminal emulator and serial communication program, in order to directly test serial command input and device responses. After understanding how the NUCLEO receives input and returns responses, we proceeded to analyze the Brain, where we identified that the Brain sends messages to the NUCLEO through threadWrite. As a third development task, we planned to implement a lane detection algorithm; therefore, we downloaded a Lane Keeping Assist implementation using OpenCV, Python, and Fuzzy Logic as a reference to support the development of our own lane detection solution.

3 Status of planned activities

Setting up the simulator
Status: completed

Implementation:
A major challenge when running the Brain project on a laptop was the incompatibility of the picamera2 and PiDNG hardware libraries. These libraries are native to the ARM architecture (Raspberry Pi) and cannot be installed on the x86_64 architecture (PC). This dependency prevented the code from starting in simulation mode.
The implemented technical solution was a conditional mocking mechanism. The application entry point (main.py) was modified to detect the system architecture (platform.machine() == 'x86_64'). When running on a laptop, virtual objects (unittest.mock.MagicMock) are injected instead of the real hardware libraries, allowing the code to initialize correctly and connect to the simulator without import errors.

Difficulties:
Hardware-specific libraries were not compatible with the PC architecture, blocking execution in simulation mode.

3D Prints
Status: ongoing

Implementation:
We successfully printed the first three traffic signs.

Difficulties:
Not reported during this period.

Hardware improvements
Status: ongoing

Implementation:
We observed abnormal noise while the vehicle was moving forward and noticed that the front wheels did not fully return to the neutral position after turning left. We opened the front and rear transmissions, cleaned the components, and made mechanical adjustments to improve smoothness and wheel movement.

Difficulties:
We still have issues with the rear differential, as it does not fit perfectly in the plastic housing. For this reason, we are planning to replace it with a new unit of the same model. Regarding the front wheels, we suspect the servo motor may be defective and plan to repair or replace it.

Semaphores
Status: ongoing

Implementation:
We opened the semaphores and connected them to an Arduino. A basic control program was developed to change the semaphore colors.

Difficulties:
The current functionality is minimal and not yet integrated into the server-based control as initially planned.

Simple commands using allMessages
Status: completed

Implementation:
A simple AUTO behaviour was implemented to control the car by sending SpeedMotor and SteerMotor commands using the allMessages queue system (messageHandlerSender). The process listens for StateChange, and when the AUTO state is detected, it executes a timed sequence.

To ensure reliable reception and interpretation of motor commands, modifications were made to the Serial Handler receive logic (threadRead), which is responsible for parsing incoming serial messages from the NUCLEO before forwarding them into the system.

Safer message splitting:
The parsing logic was updated from a generic string split to a controlled split (split(":", 1)), ensuring that messages containing additional : characters do not cause runtime errors. This prevents ValueError exceptions when unexpected delimiters appear in incoming data.

Why this approach is better:
The updated threadRead implementation represents a hardened parsing layer. By sanitizing incoming serial data and validating numeric values before conversion, the system becomes significantly more resilient to timing issues, message concatenation, and noisy serial input. This is especially important when speed and steering commands are sent close together, as it prevents parsing errors and improves overall system stability.

Difficulties:
Initially, closely timed serial transmissions resulted in malformed values or parsing inconsistencies due to insufficient temporal separation between commands. In particular, sending speed and steering commands simultaneously caused unreliable behavior on the NUCLEO side. This issue was addressed by introducing a short, controlled delay between consecutive speed and steering messages, ensuring that each command is transmitted and processed independently and reliably.

Line detection
Status**:** completed

Implementation:
The lane detection functionality was implemented as a modular perception component integrated into the system using a process–thread architecture. Camera frames are received through the message queue system and processed in real time using OpenCV techniques. The algorithm detects lane markings by applying image preprocessing, region-of-interest filtering, and line detection methods, and computes the lateral lane error relative to the vehicle's position.

At this stage, the computed lane information is used for visualization and debugging purposes only. Detected lane markings are overlaid on the camera stream, allowing continuous monitoring and validation of the detection algorithm. The module operates independently and is synchronized with the system's state machine and messaging infrastructure, enabling future integration with vehicle control logic.

Difficulties:
Lane detection was affected by varying lighting conditions, shadows, reflections, and low contrast between lane markings and the road surface. Image noise from low-resolution camera input also introduced false edges, leading to unstable detections. Defining an effective region of interest required balancing computational efficiency with detection reliability.

Solutions:
Image preprocessing techniques such as grayscale conversion, Gaussian blurring, and Canny

edge detection were applied to reduce noise and improve robustness under different lighting conditions. A focused region of interest was used to limit processing to relevant areas of the image, improving detection stability. Probabilistic Hough Line Transform was selected to better handle discontinuities in lane markings. Lane detection was implemented as a visual overlay within the camera processing pipeline, allowing validation without affecting vehicle control logic.

4 General status of the project

At the current stage, the vehicle is able to detect lane markings and visualize them on the camera stream. However, during operation, the lane detection occasionally produces unstable or inaccurate results, with detected lane lines sometimes appearing curved, misaligned, or inconsistent.

The line detection algorithm therefore requires additional refinement, particularly in the line fitting and visualization stages. At this point, it is not yet fully determined whether these detection inaccuracies will affect the lane-keeping intelligence once it is fully integrated, and further testing and tuning are required to assess and improve robustness.

Overall, the perception pipeline for lane detection is functional, but further development is necessary to achieve stable and reliable detection under varying visual conditions.

5 Upcoming activities

- Complete the 3D printing of all road components
- Configure and integrate the semaphore server
- Build and finalize the main map layout
- Implement lane-keeping functionality
- Implement traffic sign detection
- Enable semaphore recognition and interpretation