The Software Stack That Achieved Global Second Place at the Shell Eco-marathon APC 2025

Hazem Abuelanin, Abdelwahab Khaled, Ahmed Mohammed, Abdelrahman Salah, Amr Rafik, Akram Khaled, Mohamed Hany, Mohanad Mohamed, George Emil, Essam El-din Hesham, Ahmed Mahmoud, Abdelrahman Afify

***Abstract*— This report describes our approach to designing and evaluating a modular autonomous software stack focused on achieving competitive energy efficiency and validated through simulation. The system was developed using ROS and achieved a global 2nd-place finish in the Shell Eco-marathon Autonomous Programming Challenge (APC) 2025. Our architecture consists of four distinct layers: Perception, Behavior Planning, Local Path Planning, and Control. Using multi-sensor data, we performed object detection with YOLOv11-Nano and classical LiDAR methods, enabling real-time object tracking with a Kalman filter. Our local path planner, a Frenet Optimal Path with energy-aware penalties, generated highly efficient trajectories. To track these paths, our control system utilized a hybrid approach, combining PID, Pure Pursuit, and Model Predictive Control (MPC) for stable, energy-efficient actuation. The software was implemented in C++ and Python and rigorously tested using the CARLA simulator, ROSBag files, and competition scoring protocols to ensure robustness and performance.**

I. INTRODUCTION

The Shell Eco-marathon is a prestigious global competition that challenges student teams to push the boundaries of energy efficiency in vehicle design. The competition has evolved to include an innovative autonomous category, the Autonomous Programming Challenge (APC), which focuses exclusively on developing software for a simulated vehicle. This virtual discipline requires teams to design a software stack capable of navigating a complex track while minimizing energy consumption. The ultimate objective is to complete the course as efficiently as possible, with the final score based on a single metric: kilometers per kilowatt-hour (km/kWh). Rule violations such as collisions or speeding add significant penalties and disqualifications, reinforcing the demand for a robust and safe systems.

Our team, **Shoubra Racing Team Autonomous Division**, designed and implemented a modular autonomous software stack for the 2025 competition. Our approach to perception, planning, and control secured a global **2nd place finish**. The following sections of this paper detail the engineering design and on-simulation performance of our system.

Fig. 1. The Tesla Model 3 Car in Carla Simulator and RVIZ.

II. DESIGN GOALS

As a first-time competitor in the Shell Eco-marathon Autonomous Programming Challenge, our main goal for the 2025 season was to design a robust and functional autonomous system capable of safely and efficiently completing the competition track. To achieve this, we focused on two primary design goals for our autonomous software stack.

Achieve a High-Performance, Reliable Software Stack

Our analysis of the competition's unpredictable environment led us to prioritize the stability and reliability of our core autonomous pipeline. This translated into specific objectives for each module:

- **Perception:** To handle a wide range of simulated conditions, we aimed to develop a multi-sensor fusion system to provide a reliable understanding of the environment. Our goal was to accurately detect and track objects and traffic light states without false negatives.
- **Behavior Planning:** We sought to create a state machine capable of reliably transitioning between different driving modes—such as proceeding, following a vehicle, or executing a maneuver—to ensure the vehicle's safe and predictable behavior in complex scenarios.
- **Path Planning:** A key objective was to develop a planning system that could generate and follow a global path to a high degree of precision while also having the flexibility to dynamically generate new trajectories for safe obstacle avoidance and overtaking.

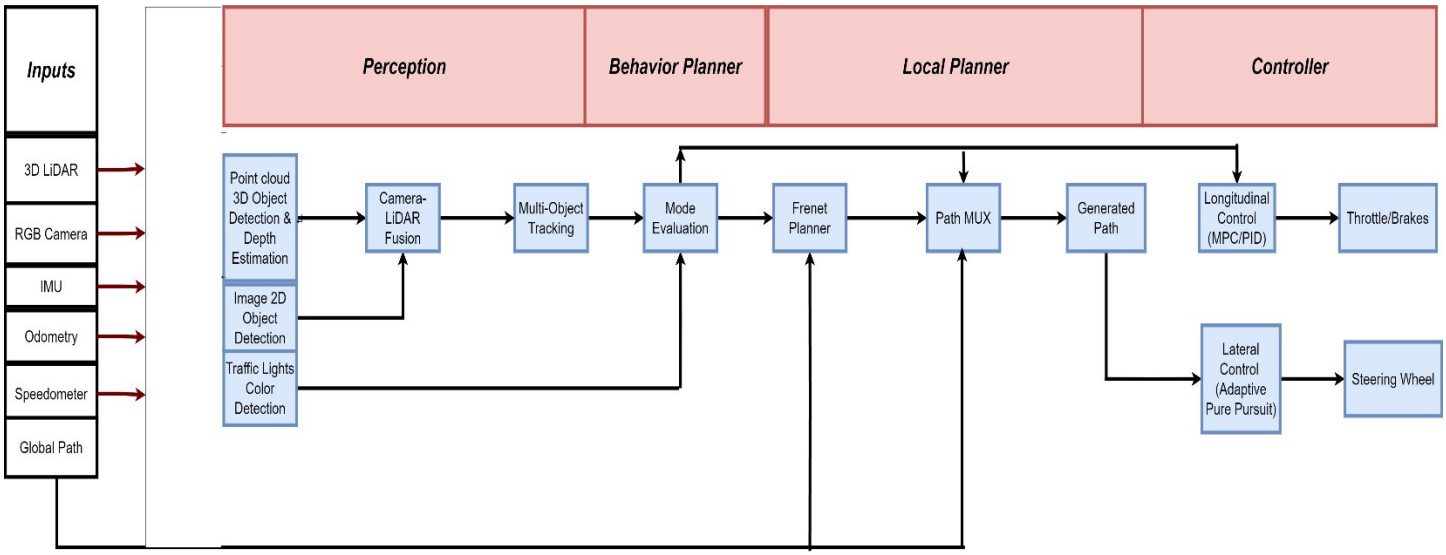

Fig. 2. System Overview

III. System Overview

The autonomous system software runs centrally within the CARLA simulation environment, which provides the necessary setup to run our autonomous software stack in real time. Our system is implemented used the Robot Operating System (ROS) framework, supporting both ROS1 (Noetic) and ROS2 (Humble). Most components are implemented in C++ for performance-critical code, while smaller utilities and high-level logic are realized in Python,

The central processing pipeline begins with the Perception layer. The environmental data, consisting of object detections and track boundaries, is then processed by the Behavior Planning layer, which makes high-level decisions. Based on these decisions, a target trajectory is planned and then realized by the control system. This entire process is constantly monitored by a high-level logic that performs sanity checks to ensure a safe drive. Additionally, our use of the CARLA simulation is capable of testing all parts of the pipeline outside of a real-world car, aiding us in fine-tuning the system and reducing necessary test time.

Figure 2 provides a high-level overview of the communication within our autonomous system. It visually represents our modular architecture and illustrates the data flow from raw sensor input to the final actuation commands sent to the vehicle in the simulation.

IV. Perception

The perception system is responsible for interpreting raw sensor data to build a comprehensive, high-fidelity understanding of the vehicle's environment. Our approach is a hybrid model that fuses data from both the LiDAR and camera to ensure a robust and reliable representation of our surroundings in the simulation.

A. LiDAR System

The core of this pipeline is to filter and cluster the raw point cloud data to isolate obstacles. To manage the data from the Velodyne 16-channel 3D LiDAR, we first applied a **Voxel Grid Filter** for downsampling, which reduced the point cloud size while preserving geometric information. A **Statistical Outlier Removal** filter then eliminated isolated noise points. The final step was **Ground Plane Removal**, where a plane was fitted using **RANSAC** (Random Sample Consensus) to precisely identify and remove ground points, leaving only potential obstacles. The filtered points were then grouped into clusters, each representing a distinct object. We used **Euclidean clustering** to group nearby points based on a distance threshold, and from each resulting cluster, we extracted key features such as bounding box dimensions and orientation.

B. Camera System and 2D Detection

*Our system uses an RGBD camera to add semantic meaning (object type) and for traffic light detection. We utilized a **YOLOv11-Nano** model to perform 2D object detection, identifying objects and traffic lights within the camera's field of view. For traffic light state detection, we used **OpenCV color detection** on the detected bounding box to reliably determine its state (red, yellow, or green), which is a crucial input for the behavior planner.*

C. Sensor Fusion and Multi-Object Tracking

*Our system's high-level understanding of the environment is achieved through sensor fusion. We combined the LiDAR's precise depth and positional data with the camera's class labels to create a complete, fused object list. This fused information was then used for **Multi-Object Tracking** via a **Kalman Filter** and the **Hungarian Algorithm**. This pipeline allowed us to predict each object's future position and maintain its identity over time, ensuring a reliable and continuous understanding of the dynamic environment.*

V. Behavior Planning

The Behavior Planning layer acts as the central decision-making unit of the autonomous software stack. It operates as a state machine, evaluating the current environmental context and the vehicle's own state to select the most appropriate driving mode. The planner's output commands the downstream modules to execute a specific behavior. Our system is capable of transitioning between four primary modes:

- **Proceed:** This is the default state when the path ahead is clear of obstacles or traffic. The system commands the vehicle to follow the pre-computed global path at a consistent, energy-efficient speed.
- **Stop:** This mode is triggered when the perception system detects a vehicle stopped in the path ahead or a red traffic light. The planner commands the vehicle to slow down and execute a safe, calculated stop.
- **Follow:** This mode is an **Adaptive Cruise Control (ACC)**-based behavior. It is activated when a moving vehicle is detected in the ego car's lane. The planner commands a safe following distance and speed based on the relative velocity and distance of the lead car.
- **Maneuver:** This is a conditional state for overtaking. The planner first checks for safety and legality (e.g., no oncoming traffic, clear lane). If the conditions are met, it transitions the system and commands the path planner to generate an overtaking trajectory.

Based on the rqt_graph provided, the diagram illustrates the connections and data flow between the various nodes and topics within the ROS autonomous system. It serves as a comprehensive visual overview of the software stack's modular components and their real-time communication.

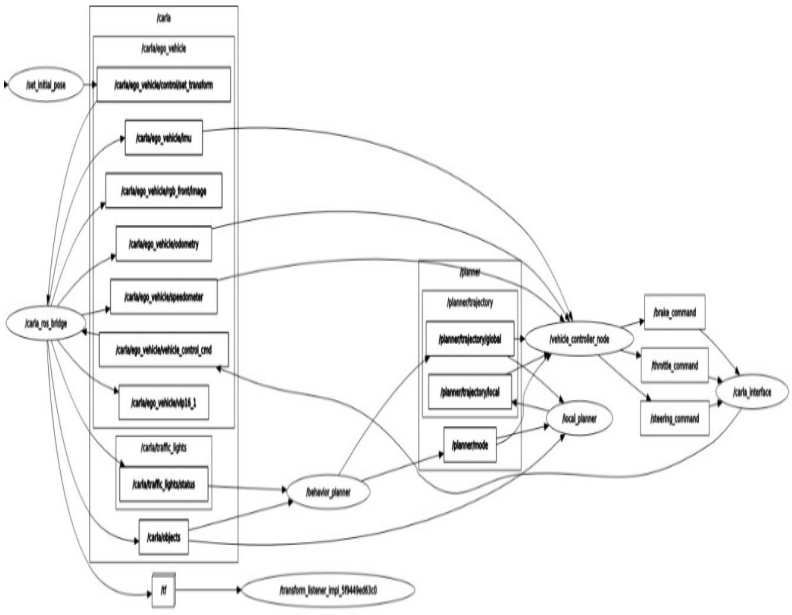

Fig. 3. The rqt_graph of the nodes and topics of the system.

VI. LOCAL PLANNING

The autonomous system employs a two-tiered planning strategy to balance computational efficiency with dynamic maneuverability. The vehicle's primary guidance is provided by a pre-computed global path. The local planning module then works to generate real-time trajectories that are both safe and energy-efficient.

Our central component of this strategy is a **path multiplexer (mux)** that acts as a switch, taking its input from the behavior planner. Based on the current driving mode, this mux determines which path the vehicle should follow: the pre-computed global path or a dynamically generated local path.

For the purpose of local trajectory generation, we utilize a **Frenet Optimal Planner**. This planner is particularly well-suited for the competition's structured urban roads, as it generates paths relative to a reference trajectory. This approach provides a significant advantage in structured environments where a clear, pre-defined road exists. The Frenet planner is only computationally active when the behavior planner's mode is "Maneuver." This on-demand approach is highly efficient as it saves significant computational power during standard driving, when the vehicle is simply following its global route.

Figure 4 provides a detailed visualization of the map and the 14 target points that defined the competition route. It clearly shows the spatial relationships of the checkpoints, providing context for the vehicle's navigation tasks.

The Frenet planner evaluates and generates a set of safe and energy-efficient paths for overtaking. Each path is scored against a multi-objective cost function that includes both safety and energy-related penalties. This function heavily penalizes high curvature, sharp acceleration/deceleration, and any paths that would lead to a collision. By explicitly including an energy cost, our system is able to consistently select the trajectory that is not only safe but also the most energy-efficient for the given maneuver.

Fig. 4 Layout of the map

VII. CONTROLLING

The control layer is the final stage of the autonomous software stack, responsible for translating the planned trajectory into physical commands for the vehicle's actuators. Our system uses a hybrid approach for both lateral and longitudinal control, which was tuned to prioritize stability and, most importantly, energy efficiency.

For lateral control, we utilized an **Adaptive Pure Pursuit** controller. The lookahead distance was dynamically scaled based on the vehicle's speed, allowing for optimal performance across the entire velocity range. This approach ensured exceptional stability in corners, even at relatively high speeds.

Our tuning resulted in a lateral tracking error of less than 8 cm, demonstrating the high precision and stability of our implementation.

For longitudinal control, we chose between a **Model Predictive Control (MPC)** and an **Adaptive PID** controller. The primary tuning target for both controllers was to manage the throttle effectively and avoid sudden speed changes, high-speed travel, and uncontrolled accelerations—all factors that significantly consume energy. We found that the MPC was the superior option in terms of performance and energy optimization but came with a high computational cost. As a result, the Adaptive PID controller served as a highly effective backup that still delivered excellent results. This controller achieved a significant **30% reduction in energy consumption** when compared to a standard PID controller, directly contributing to our high ranking in the competition.

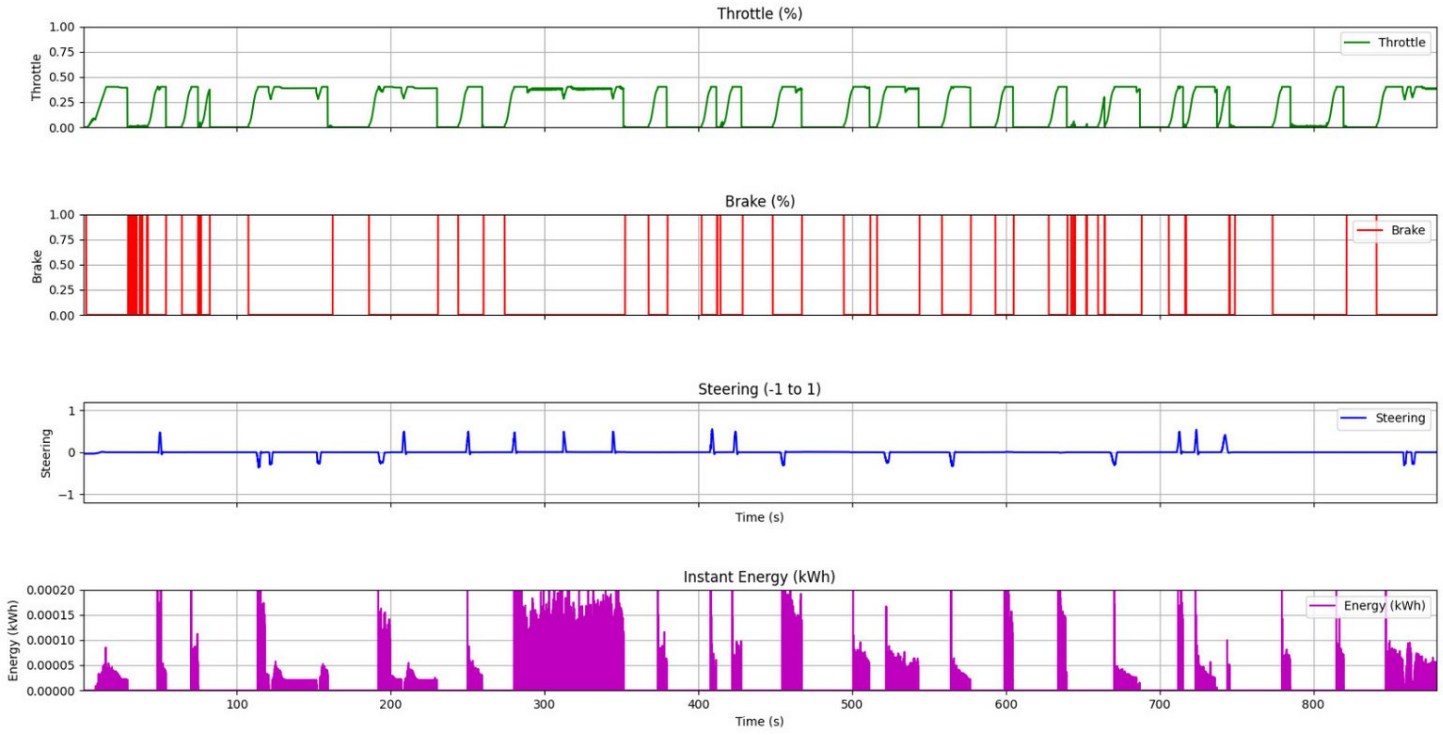

Fig. 6 Throttle, Brakes, Steering and Energy Consumption graphs

VIII. RESULTS

Our team's performance in the Shell Eco-marathon Autonomous Programming Challenge 2025 was a direct validation of our software stack's design goals. After a series of six total submissions, our highest-scoring attempt secured a **global 2nd place finish** out of 24 teams worldwide. This result demonstrates that our systematic approach to energy efficiency, combined with a robust and reliable architecture, was a winning strategy.

The key metrics from our top-performing run are summarized as follows:

Goals Reached: We successfully reached all 14 designated goals on the course, validating the effectiveness of our TSP solver and global path planning.
Total Time: The run was completed in 384.3 seconds.
Energy Efficiency: Our final score was **15.98 km/kWh**, a testament to the meticulous tuning of our longitudinal and lateral controllers.

Our system successfully completed the entire run with zero collisions, highlighting the reliability of our perception and behavior planning modules. The high energy efficiency was a direct result of our conscious design choices to avoid energy-wasting behaviors, such as sudden accelerations and braking. This performance validates our approach that a focus on intelligent planning and smooth, energy-aware control is paramount for success in an efficiency-focused competition. The strong result solidifies our belief that a well-engineered software stack is the key to achieving a high-ranking finish in this challenging autonomous competition.