TED UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

Project Name: F&D Autonomous Driving

HIGH-LEVEL DESIGN REPORT

Web Page URL:https://fnd-autonomous.github.io/F-D-autonomous.github.io/

Team Members:

Taylan Kayalı

Can Soysal

Taha Ugan

Çağrı Şanlı

Mustafa Cem Hür

Supervisor:

Tolga Kurtuluş Çapın

Jury Members:

Deniz Cantürk

Hakkı Gökhan İlk

1. Introduction

1.1 Purpose of the System

The purpose of our system is to provide both a comprehensive, virtual autonomous driving simulation platform that enables the development of self-driving algorithms in a risk-free environment and the algorithm itself. F&D Autonomous Driving System aims to simulate a vehicle that can navigate complex scenarios without human intervention means that the algorithm will be responsible from the perception of the road network to high-level path planning. The system is not a physical vehicle; it does not operate on real-world roads or carry actual passengers. The platform is designed to serve as a high-fidelity testing ground, complementing physical testing by filtering out dangerous logic errors before they reach hardware.

The F&D will divide the developers as two groups: the AI developers (algorithm engineers) and the simulation management group. Developers use the Gazebo simulation environment to train the vision model, develop a planning algorithm (path planning mainly) and optimize energy efficiency.

The key functions of the system include a perception phase that includes both computer vision and localization, a dual-layer path planning architecture (global and local), a robust and a control system for actuation.

1.2 Design Goals

F&D Autonomous Driving System ensures if the system is reliable, efficient, and modular. It ensures the vehicles safety and reliability in dynamic environments. Also, we need to get some rapid responds to critical obstacles or traffic signs since it will need to give some immediate actions according to the changes in the environment. The system also will ensure that the stopping distance is maintained (e.g. <0.30m) without any delay, which is essential during emergency braking scenarios. The obstacle detection mechanisms aim to provide informative classification by detecting deviations in the road environment (like barriers or cones), rather than just raw distance measurements.

One of the most critical goals is energy efficiency. Unlike standard navigation stacks that prioritize the fastest route, F&D processes trajectory data to minimize sudden actions and optimize steering angles in order to ensure that the system works as energy efficient as possible. This requires precise algorithmic control, energy-aware pathfinding, and compliance with virtual physics constraints. Operational stability is one of our fundamental system requirements.

We also need to ensure that the algorithm supports real-time responsiveness, which enables continuous analysis of camera and ultrasonic sensor feeds through the Gazebo simulation. Because basically the perception stage will warn the planning stage. When a path is blocked, the re-routing must be accurate, and the system operator should be warned of the change. The important thing about this situation is that the new route must be energy efficient as well since it is one of our main goals. This way, it will be working real-time while covering our energy efficient path planning requirement.

1.3 Definitions, acronyms, and abbreviations

Acronyms and Abbreviations

API - Application Programming Interface

CPU - Central Processing Unit

FPS - Frames Per Second

GPU - Graphics Processing Unit

MPC - Model Predictive Control

PID - Proportional-Integral-Derivative (Control)

RGB - Red-Green-Blue (Color Model)

ROS2 - Robot Operating System 2

YOLO - You Only Look Once (Object Detection Algorithm)

1.4 Overview

Firstly we figured out the details about the architecture of the system. Then we mapped out our general software structure which is as follows: firstly, data is collected through simulated sensors or through importing some usable external data, after that with real-time processing and object detection, lastly with actuating the vehicle's steering and throttle. In order to ensure reliability, the cases that are outside of normal system behaviour like sensor failure or obstruction are also handled. After that we moved on to subsystems that make up the general architecture of our system. Lastly we brought all of these together to finalize the system.

2. Current Software Architecture

We are in the environment design stage of the project as for now. We designed the 3D models that we will use for the project such as the map, vehicle and signs. We are currently in the process of importing these models to our gazebo simulation environment.

3. Proposed Software Architecture

3.1 Overview

The proposed architecture for the F&D Autonomous Driving System follows a layered, modular design that separates concerns and allows components to be developed and tested independently. The architecture is built around the sense-plan-act paradigm common in robotics, but extended and refined to support our specific requirements for autonomous driving with energy efficiency.

At the highest level, the system consists of several major subsystems that work together to enable autonomous operation. The perception subsystem processes raw sensor data to understand the environment. Also it does the localization that determines where the vehicle is. The planning subsystem decides what the vehicle should do at multiple time scales (both global and local). The control subsystem executes planned actions by commanding the vehicle.

These subsystems communicate through well-defined interfaces using ROS2's publish-subscribe message system. This loose coupling means that subsystems do not need detailed knowledge of each other's internal implementation and they only need to know what messages they should publish or subscribe to. This makes the system more modular and easier to modify.

A key architectural principle is the separation of global and local planning. Global planning operates on a map scale, computing routes through the road network. Local planning operates at a shorter time window, generating detailed trajectories the vehicle will follow over the next few seconds. This separation allows each planner to use algorithms appropriate to its time scale without one becoming overly complex trying to handle both.

Another important principle is making energy efficiency the most important concern throughout the architecture. Rather than optimizing for energy as a post-processing step, energy considerations are built into the interfaces and algorithms of the planning and control subsystems. Planners can request energy consumption estimates when evaluating options. Controllers can optimize their execution for smooth, energy-efficient motion. This architectural support makes it easier to achieve our energy efficiency goals.

3.2 Subsystem Decomposition

The F&D Autonomous Driving System is decomposed into the following major subsystems, each with specific responsibilities and interfaces.

Perception Subsystem

The Perception Subsystem is responsible for converting raw sensor data into structured information about the environment that other subsystems can use. It processes camera images and ultrasonic sensor readings to detect and classify objects, identify lane markings, recognize traffic signs, and estimate distances to obstacles.

The vision processing handles camera data. It preprocesses images, runs object detection models

to identify obstacles and traffic signs, performs lane detection to find lane boundaries and markings, and extracts relevant features from the visual scene. This component uses machine learning models trained on simulation data to recognize the specific objects and features defined in our requirements.

The obstacle classification takes detected objects and classifies them according to our requirements, distinguishing track obstacles from parking obstacles based on their dimensions and visual characteristics. This classification determines how the system responds to different obstacles.

The traffic sign recognition specifically identifies stop signs and parking signs. It needs to be reliable enough to detect these signs with sufficient advance warning for the vehicle to respond appropriately.

The perception subsystem publishes its findings on several ROS2 topics: detected obstacles with their positions and classifications, recognized traffic signs with their types and locations, identified lane markings with their positions and types, and general environment state information. These messages provide the input that other subsystems need to make decisions.

Planning Subsystem

The planning subsystem maintains an estimate of the vehicle's position and orientation within the environment. In a real vehicle, this would typically combine GPS, inertial sensors, and visual odometry. In our simulation, we have access to ground truth position from Gazebo, but we still implement a localization system that mimics what would be needed in reality.

The subsystem publishes the current pose estimate along with uncertainty information at least once per second, meeting our requirement for continuous pose estimation with average position error under 0.50 meters. Other subsystems subscribe to this pose information to understand where the vehicle is for planning and control purposes.

The planning subsystem is also composed of two tightly related but distinct components: the global planner and the local planner.

The global planner operates at the route level. It takes as input the vehicle's current position and a list of target nodes that must be visited. It computes a path through the road network that visits all nodes while staying on drivable surfaces. The path is represented as a sequence of waypoints connected by road segments.

The global planner uses a graph based representation of the road network where intersections are nodes and road segments are edges. It applies path-finding algorithms like A* to find optimal routes. The optimization criterion considers not just geometric distance but also estimated travel time and energy consumption. When an obstacle blocks the planned route, the Global Planner updates the graph (marking blocked segments as not passable) and re-calculates the path.

The global planner must complete path computation within 10 seconds of receiving a route request, as specified in our requirements. For complex networks, this might require using heuristics or approximation techniques to stay within the time budget.

The local planner generates short term trajectories that the vehicle will follow over approximately. These trajectories are detailed paths specifying not just where the vehicle should go, but when it should be at each position along the path.

The local planner operates at a much faster timescale than the global planner, updating at least once per 3 seconds and potentially more frequently when the situation demands. It takes as input the global path, the current vehicle state, the perceived environment (obstacles, lanes, traffic signs), and current traffic rules. It outputs a trajectory that follows the global path while avoiding obstacles, staying in lane, obeying traffic signs, and optimizing for energy efficiency.

Energy efficiency is a key consideration in local trajectory generation. The local planner evaluates different trajectory options and selects ones that minimize unnecessary acceleration and braking. Smooth trajectories that maintain relatively consistent speed are generally more energy efficient than trajectories with frequent speed changes.

Control Subsystem

The Control Subsystem is responsible for making the vehicle actually follow the planned trajectories. It takes as input the current desired trajectory and the current vehicle state, and outputs steering angles, throttle positions, and brake commands to send to the simulated vehicle.

The controller needs to be responsive enough to track trajectories accurately (the requirement specifies trajectory tracking error under 0.25 meters) while being smooth enough to avoid unstable motion that wastes energy. This requires well-tuned control algorithms, likely using techniques like proportional integral derivative (PID) control or model predictive control (MPC).

The Control Subsystem operates at a high frequency. It continuously monitors how well the actual vehicle state matches the desired trajectory and adjusts control commands to minimize tracking error.

The subsystem also implements safety checks. If it detects that a collision is imminent despite following the planned trajectory, it can execute an emergency stop. This provides a final layer of defense against collisions.

3.3 Hardware/Software Mapping

The F&D Autonomous Driving System operates entirely within a simulated environment, which means our hardware/software mapping is actually a mapping to simulated hardware resources provided by Gazebo.

Computation Resources

All software components run on the same physical computer that is executing the Gazebo simulation. This computer needs sufficient CPU and GPU resources to run both the physics simulation and our software stack in real-time (for training purposes).

The GPU is primarily used for running the physics simulation and rendering the 3D environment in Gazebo, and also for accelerating neural network inference in our perception components. Modern object detection models like YOLO can run much faster on GPUs than CPUs, which is important for performance requirements.

CPU resources are shared among all the software components. ROS2 provides a multi-process architecture where each major component can run as a separate process which allows the operating system to distribute the computational load across multiple CPU cores.

Components with high computational demands like vision processing and patlı planning are designed to use CPU resources efficiently.

Simulated Sensors

Gazebo simulates the sensors our vehicle uses to perceive its environment. These simulated sensors generate data that matches what real physical sensors would produce, including realistic noise characteristics.

The camera sensor provides RGB images at a resolution and frame rate we specify. In our architecture, this camera data flows from Gazebo through the Simulation Interface to the Vision Processing component of the Perception Subsystem. The component processes these images to detect objects, lanes, and traffic signs.

Ultrasonic distance sensors provide distance measurements to nearby objects. These are simulated with realistic characteristics including limited range, beam width, and measurement noise. The Distance Estimation component combines ultrasonic data with visual information to get robust distance estimates.

Simulated Actuators

The vehicle in Gazebo responds to control commands for steering, throttle, and braking. These simulated actuators model realistic vehicle dynamics including acceleration limits, steering rate limits, and momentum effects.

Our Control Subsystem generates control commands based on desired trajectories, and these commands flow through the Simulation Interface to Gazebo's vehicle model. The vehicle's response to these commands follows realistic physics, meaning the Control Subsystem must account for dynamics like steering lag and momentum just as it would with a real vehicle.

Network and Communication

All inter component communication uses ROS2, which provides a distributed communication framework. While all components physically run on the same machine in our setup, ROS2's architecture would also support distributing components across multiple machines connected by a network if needed for increased computational resources.

ROS2 uses a publish-subscribe model with a discovery service that allows components to find and connect to each other automatically. This makes the system flexible and allows us to easily add or remove components during development.

3.4 Persistent Data Management

The F&D Autonomous Driving System's data management strategy focuses on what information needs to persist across runs versus what is transient and only exists during operation.

Short Term Operational Data

Most data in the system only exists during a run but does not need to be saved afterward. This includes sensor readings, perceived environment state, planned trajectories, control commands, and vehicle state. This data flows through the system via ROS2 messages and is used by components that need it, then discarded. The only exception is when the data is logged explicitly.

Environment and Map Data

The simulated environment itself is persistent data. The Gazebo world files describe the road layout, obstacle placements, traffic sign positions, and other environmental features.

We maintain a library of world files representing different scenarios which are simple test environments for basic functionality testing, complex environments for performance evaluation, and specific scenario worlds for testing particular capabilities like parking or stop sign compliance.

Machine Learning Models

Trained neural network models for object detection, lane detection, and other perception tasks are persistent data. These models are trained on collected data, then saved to disk so they can be loaded when the system runs.

As development progresses and we collect more diverse training data, we retrain models and evaluate whether new versions perform better than previous ones.

Development and Version Control

All source code, configuration files, world descriptions, and documentation are maintained in a version control system (likely Git). This provides a complete history of the project's development and allows team members to work on different aspects of the system simultaneously without conflicts.

We use branching strategies to manage development: a main branch containing stable, tested code; development branches for work in progress; and feature branches for implementing specific new capabilities. This organization helps coordinate the team's work.

3.5 Access Control and Security

Access control and security considerations for the F&D Autonomous Driving System are straightforward since it operates entirely in a simulation environment on local hardware without network exposure or sensitive data handling.

Development Environment Access

Since this is a student project for academic purposes rather than a system handling sensitive data or deployed in a production environment, we do not require elaborate security measures.

Code and Repository Access

The version control repository is accessible to all team members, with permissions to read and commit changes. We use standard Git workflows with pull requests for significant changes, allowing team members to review each other's code before it is merged into main branches. This review process improves code quality and helps share knowledge across the team.

Simulation Environment Isolation

The Gazebo simulation runs in a controlled environment on our development machines. There is no exposure to external networks of untrusted input. The simulation loads world files and configuration data from the local filesystem, and these files are under version control where we can implement changes.

Data Privacy

The system does not handle any personal data or sensitive information. All data is generated within the simulation. Logged data contains information about simulated vehicle behavior and environment state, but nothing that would raise privacy concems.

3.6 Global Software Control

The global control flow of the F&D Autonomous Driving System describes how the various subsystems coordinate their activities and how control passes between components as the system operates.

System Initialization

When the system starts up, initialization happens in a specific sequence to ensure all components are ready before operation begins.

First, the Simulation Interface initializes and establishes connection to the running Gazebo simulation. It verifies that it can receive sensor data and send control commands. Next, the core subsystems initialize in order. The Perception Subsystem starts up, loads its trained machine learning models, and begins accessing sensor data from the Simulation Interface. Then the localization system initializes its pose estimate, starting from a known initial position. Then the Planning Subsystem loads map data and prepares its planning algorithms. After that the Control Subsystem loads its configuration parameters and prepares to receive trajectories.

Normal Operation Cycle

During normal operation, the system executes a continuous sense-plan-act cycle. This cycle repeats at a high frequency throughout the vehicle's operation. The sensing phase begins with sensor data arriving from Gazebo through the Simulation Interface. The Perception Subsystem processes this data, updating its understanding of the environment. The localization incorporates sensor information to refine its pose estimate. These updates happen

asynchronously as new sensor data arrives. The planning phase happens on a regular schedule. At least once per second, the local planner generates a new short term trajectory based on the current global path, vehicle state, and perceived environment. The Global Planner runs when needed at system startup to compute the initial route, and during operation when replanning is required due to blockages or new navigation commands. The acting phase happens at the highest frequency. The Control Subsystem continuously computes control commands to follow the current trajectory and sends them to Gazebo.

Event-Driven Control Flow

Certain events trigger specific control flows outside the normal cycle. When a stop sign is detected, the Behavior Manager takes control: it gives a command to the Local Planner to generate a stopping trajectory, monitors execution until the vehicle is stopped, waits until the required time, then resumes motion. The normal planning cycle is temporarily modified to execute this sequence.

When an obstacle blocks the planned path and local avoidance is not possible, the Behavior Manager triggers global replanning. It gives a command to the Local Planner to stop the vehicle safely, updates the Global Planner's map to mark the blockage, then requests a new route and once replanning completes, resumes normal operation following the new route.

When a parking maneuver is required, the Behavior Manager switches the system into parking mode. This involves generating parking trajectories rather than driving trajectories and control parameters that allow reverse motion which is normally prohibited. After parking completes, the system switches back to normal driving mode.

Error Handling and Recovery

When components detect errors, they publish error messages. The Behavior Manager looks at these messages and implements recovery strategies.

If a component stops publishing expected messages, the Behavior Manager detects the timeout and can take action which consists of typically stopping the vehicle safely and logging the error for later analysis. If the Perception Subsystem reports consistently poor detection confidence, the Behavior Manager can reduce vehicle speed to increase safety margins. If localization uncertainty exceeds thresholds, the vehicle might stop until localization can be reestablished. Some errors like loss of communication are unrecoverable within the system.

3.7 Boundary Conditions

Boundary conditions describe how the system behaves at the edges of normal operation—during startup, shutdown, and when encountering unusual situations.

System Startup

At startup, the system begins in an uninitialized state. Components initialize themselves and signal when they're ready. The Behavior Manager waits for all critical components to report readiness before allowing operation to begin.

If initialization fails for any component, the system does not proceed to operation. Instead, it reports which component failed and why, then shuts down. This prevents partial initialization that could lead to unpredictable behavior.

During startup, the vehicle must be in a safe initial state which is stationary, on a drivable surface, with a known initial pose. The system verifies these conditions before proceeding. If the initial pose is unknown or uncertain, the Localization Subsystem establishes localization before the vehicle can move.

Normal Operation Boundaries

During operation, the system maintains awareness of its operational boundaries and responds appropriately when approaching them.

If the vehicle approaches the edge of the mapped area, the Planning Subsystem prevents plans that would take it outside known territory. If the navigation task requires going beyond the mapped area, the system reports that the task cannot be completed rather than attempting to operate without map information.

If sensor quality degrades significantly due to simulated lighting conditions or sensor noise, the Perception Subsystem reports. The Behavior Manager responds by reducing vehicle speed or even stopping if perception becomes too unreliable for safe operation.

If computational resources become constrained and components cannot maintain their required update rates, the Diagnostics Subsystem detects this and reports it. The Behavior Manager responds by reducing vehicle speed to effectively slow down time, giving components more real-world time to complete their computations.

Task Completion

When the vehicle successfully visits all target nodes and reaches the final destination, the Behavior Manager recognizes task completion. It brings the vehicle to a controlled stop, reports successful completion with relevant metrics like time taken, energy consumed, any violations and transitions to a completed state where it awaits either a new navigation task or shutdown command.

Task Failure

Some situations prevent task completion. If the Global Planner determines that no valid path exists to reach required nodes like blockages or map limitations it reports this to the Behavior Manager. The Behavior Manager stops the vehicle safely and reports task failure and explains why completion was not possible.

If the system exceeds time limits specified for task completion, the Behavior Manager recognizes timeout. It stops the vehicle and reports a timeout failure with information about how much of the task was completed.

If critical failures occur like repeated collision attempts despite avoidance algorithms, or sustained inability to maintain localization, the Behavior Manager declares the task failed and initiates controlled shutdown.

Error Recovery Boundaries

The system attempts to recover from errors when possible, but recognizes boundaries beyond which recovery is not possible.

Transient errors like temporary sensor glitches or momentary planning failures trigger retry logic. The system might stop briefly, reassess the situation and attempt to continue.

Persistent errors that continue despite retry attempts eventually exceed recovery boundaries. After a defined number of retry attempts, the system gives up on recovery and either fails the current task or shuts down entirely which depends on the error.

Some errors are immediately recognized as unrecoverable. In these cases, the system does not waste time on recovery attempts but immediately initiates controlled shutdown.

Safety Boundaries

Safety boundaries are the most critical. If the system detects that continuing operation would violate safety constraints, it takes immediate action.

If collision is imminent and cannot be avoided through normal planning and control, the Control Subsystem executes an emergency stop regardless of what other components are doing. This emergency stop has priority over all other commands.

If the vehicle deviates from its lane beyond safety margins despite the Control Subsystem's attempts to maintain lane keeping, the Behavior Manager stops the vehicle to prevent further deviation. Operation does not resume until the problem is corrected.

If any critical safety component fails health checks like the Perception Subsystem being unable to detect obstacles reliably the system does not allow continued operation. At this point it is better to stop safely than to continue with degraded safety capability.

4. Subsystem Services

This section summarizes the main services provided by each subsystem and the interfaces they expose to the rest of the system.

4.1 Perception Subsystem

The Perception Subsystem processes sensor data to understand the environment.

- **Vision Processing:**
  Detects obstacles, lane markings, and traffic signs using camera data.

- **Distance Estimation:**
  Fuses ultrasonic sensors and vision-based depth to estimate distances to detected objects.

- **Localization State:**
  The localization state estimates the vehicle's position and motion.

4.2 Planning Subsystem

The Planning Subsystem generates global paths and local trajectories.

- **Global Planning:**
  Computes routes based on navigation goals.

- **Local Planning:**
  Generates short-term, dynamically feasible trajectories.

4.3 Control Subsystem

The Control Subsystem converts trajectories into vehicle commands.

- **Trajectory Tracking:**
  Subscribes to local trajectories and publishes steering, throttle, and brake commands.

- **Emergency Stop:**
  Performs immediate maximum safe braking when triggered.

- **Control Status:**
  Publishes tracking error, control mode, and emergency status.

4.4 Simulation Interface

The Simulation Interface connects the system to Gazebo.

- **Sensor Data:**
  Republishes simulated sensor data in standard ROS2 formats.

- **Control Commands:**
  Translates commands into Gazebo-compatible commands.

- **Simulation Control:**
  Provides reset, pause, and speed control.

5. Glossary

A* Algorithm: A graph search algorithm commonly used in path planning that finds the shortest path between nodes using a heuristic to guide the search efficiently.

Actuator: A component that converts control signals into physical action. In our simulated vehicle, actuators control steering, throttle, and braking.

Autonomous Driving: The capability of a vehicle to operate without human intervention, including perception, decision-making, and control functions.

Depth Estimation: The process of determining the distance to objects in a scene, typically from camera images.

Energy Efficiency: The optimization of energy consumption relative to task completion, a core design principle of the F&D system.

FPS (Frames Per Second): The rate at which a vision system processes image frames, indicating the temporal resolution of visual perception.

Gazebo: An open-source 3D robotics simulator that provides realistic physics simulation and sensor modeling for robot development and testing.

Global Path Planning: The computation of a complete route from a starting location to a destination through a road network, typically represented as a graph.

Latency: The delay between when an event occurs and when the system responds to it. Lower latency means faster response.

Local Path Planning: The generation of short-term, detailed trajectories that the vehicle will follow over the next few seconds, accounting for immediate obstacles and constraints.

Localization: The process of determining the vehicle's position and orientation within the environment, typically through a combination of sensor data and map information.

Machine Learning (ML): Algorithms and techniques that enable computers to learn patterns from data rather than following explicitly programmed rules.

Model Predictive Control (MPC): A control technique that uses a model of the system to predict future behavior and optimize control actions over a time horizon.

Node: In path planning, a location in the road network graph representing an intersection or waypoint. Also refers to a ROS2 process.

Object Detection: The computer vision task of identifying and locating objects of interest (like obstacles or traffic signs) in images.

Perception: The subsystem responsible for processing raw sensor data to extract meaningful information about the environment.

PID Control (Proportional-Integral-Derivative): A common control algorithm that adjusts control outputs based on current error, accumulated past error, and rate of error change.

Publish-Subscribe: A messaging pattern where components publish messages to topics and other components subscribe to topics to receive messages, without direct coupling between publishers and subscribers.

Real-Time Performance: The capability of a system to respond to events within strict time constraints, typically meaning the system keeps up with the rate of incoming data and external events.

ROS2 (Robot Operating System 2): A flexible framework for writing robot software, providing tools and libraries for inter-process communication, device drivers, and visualization.

Trajectory: A time-parameterized path that specifies not just where the vehicle should go, but when it should be at each position, implicitly defining its velocity profile.

Ultrasonic Sensor: A distance sensor that uses sound waves to measure distances to nearby objects, commonly used in parking assistance and obstacle detection.

Virtual Environment: A simulated representation of the real world for testing autonomous systems safely without physical vehicles or infrastructure.

Waypoint: A specific point along a path or route that the vehicle should pass through or near.

YOLO (You Only Look Once): A family of real-time object detection algorithms that process entire images in a single pass through a neural network, making them suitable for applications requiring high frame rates.

6. References

1. N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sendai, Japan, 2004, pp. 2149–2154.

2. S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, May 2022.

3. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, 2016, pp. 779–788.

4. A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035.

5. S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA, USA: MIT Press, 2005.

6. X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3D object detection network for autonomous driving," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017.

7. European Commission, "Ethics guidelines for trustworthy AI," 2020.

8. ROS2 Documentation, "ROS 2 Design," [Online]. Available: https://design.ros2.org/

9. Gazebo Documentation, "Gazebo Tutorials," [Online]. Available: http://gazebosim.org/tutorials

10. OpenCV Documentation, "OpenCV Documentation," [Online]. Available: https://docs.opencv.org/

11. PyTorch Documentation, "PyTorch Tutorials," [Online]. Available: https://pytorch.org/tutorials/