1. Simulation Codes

You are responsible for preparing **all runnable, well-documented simulation code** described below.
The final code package must include **MATLAB/Simulink code**, **ROS2/Python code**, **simulation results**, and **documentation**.

Note: Your simulation should be able to **accept state estimates provided by an external estimator** (e.g., EKF/UKF/PF outputs), either as:

- prerecorded data, or
- ideal ground-truth state with optional noise.

1.1 MATLAB / Simulink Code Requirements

Prepare the following MATLAB components:

Path Planning

Implement the following planners:

- A*
- RRT*
- Hybrid A* **or** BIT* / FMT* (choose at least one advanced method)

Your simulation should allow planners to be called from a single main script.

Control

Implement:

- PID
- LQR
- MPC

Include all tuning parameters used in simulations.

CBF Safety Layer

Implement a Control Barrier Function (CBF) module that:

- Monitors predicted trajectories
- Enforces safety constraints (collision avoidance, minimum distance)
- Computes modified control commands using a real-time QP

Simulation Scripts

Provide:

- A main script that executes planning → CBF → control
- Metric functions (runtime, path length, success rate, smoothness)

- Plotting functions for trajectories and performance visualizations

Use of State Information

Your simulation must **accept** state-estimation inputs.
You may use:

- Perfect ground truth,
- Ground truth with added noise, or
- Provided external estimator logs.

1.2 ROS2 / Python Code Requirements

A minimal but functional ROS2 package must be included.

ROS2 Package Structure

```
vehicle_nav/
  package.xml
  setup.py
  vehicle_nav/
    astar_node.py
    rrtstar_node.py
    pid_node.py
    lqr_node.py
    mpc_node.py
    cbf_node.py
  launch/
    sim_launch.py
  config/
    params.yaml
```

Requirements

- Each node must run and demonstrate core functionality
- Planner node accepts current state (from external estimator or synthetic data) and produces a trajectory
- CBF node filters the control signal for safety
- Controller node executes the trajectory

1.3 Documentation Requirements

Prepare the following documentation files:

README.md

Include:

- Installation instructions
- MATLAB version & ROS2 version
- How to run each simulation
- Example commands for planners, controllers, and CBF
- Folder structure explanation

File Manifest

List:

- Every file
- Its purpose
- Inputs/outputs

Open-Source License

MIT, BSD, or Apache 2.0.

1.4 Simulation Results

Provide:

- Simulation outputs used in the report
- .mat, .csv, or .log output files
- Figures (trajectories, error plots, timing plots, safety margin plots)

2. REPORT RESPONSIBILITIES

You will write a **technical report** describing the simulation environment, algorithms, and results.
Your report must include **three main sections** as follows:

MAIN SECTION 1: Simulation Environment & Software Architecture

Describe how the simulation system is built.

1.1 MATLAB / Simulink Environment

Explain:

- Overall simulation architecture
- Planning, control, and CBF modules

- How state estimates are **used**, not generated
- Scenario setup (maps, obstacles, noise models)
- Interaction between planners → CBF → controllers
- Logging/metrics computation

Include diagrams (flowcharts or block diagrams).

1.2 ROS2 Environment

Describe:

- ROS2 package structure
- Nodes (planner, controller, CBF)
- Message topics for state input and control output
- Launch file setup
- Assumptions or simplifications

Include architecture diagrams.

MAIN SECTION 2: Algorithm Implementations

Describe what you implemented.

2.1 Path Planning Modules

For A*, RRT*, advanced planner:

- Algorithm description
- Heuristic tuning (A*)
- Sampling strategy (RRT*)
- Collision checking method
- Pseudocode
- Complexity

2.2 Control Modules

For PID, LQR, MPC:

- Control law
- Tuning strategy
- Stability considerations
- How controllers track planned trajectories

2.3 CBF Safety Layer

Include:

- Safety constraints and barrier function formulation
- Real-time QP used for control filtering
- Integration with the controller
- How the module ensures safe trajectories

2.4 Integration Pipeline

Describe the full autonomy loop:

state input (from external estimator) → planning → CBF safety refinement → control → vehicle dynamics

Include diagrams and specify update rates.

MAIN SECTION 3: Simulation Results & Analysis

Document the performance of the implemented modules.

3.1 Planning Performance

Include:

- Path length comparison
- Runtime comparison
- Smoothness metrics
- Success rate
- Example trajectories

3.2 Control Performance

Include:

- Tracking error
- Control input smoothness
- Response to disturbances

3.3 CBF Safety Layer Performance

Include:

- Collision avoidance demonstrations
- Safety margin plots
- Control modifications by the CBF layer
- Effect of CBF on tracking accuracy and smoothness

3.4 Discussion

Summarize:

- Which algorithms performed best and why
- Trade-offs between planners, controllers, and CBF
- Limitations of the current system
- Recommendations for future improvements