Stark Industries Pvt. Ltd.

PROBLEM STATEMENT – PS2: THE VOLATILE CARGO

1. Overview

STARK autonomous convoys carrying fragile Stark lab cargo must maintain comfort, stability, and safety over uneven roads. In this problem, you will design an Active Suspension controller for a 2-DOF quarter-car model and evaluate its performance on a set of pre-generated road profiles.

During this challenge, you are required to:

Load the provided dataset (road displacement vs. time) along with accelerometer signals from both the sprung and unsprung masses.

Implement or reuse a 2-DOF quarter-car dynamic model for a single wheel station using the given parameters.

Design your own Active Suspension controller that outputs the command c(t) within the given limits using only the accelerometer measurements and vehicle constraints (plus any causal internal memory).

Include a 20 ms actuator delay between the commanded c(t) and the applied c(t) in the dynamics.

Run the closed-loop model on all five road profiles and record the displacement responses of the sprung mass (required) and unsprung mass (bonus).

Compute and submit the required per-profile metrics in the specified format and include displacement plots in your report.

This problem tests your understanding of control systems, dynamic modelling, numerical simulation, and performance evaluation in a realistic automotive context.

2. Quarter-Car System Description

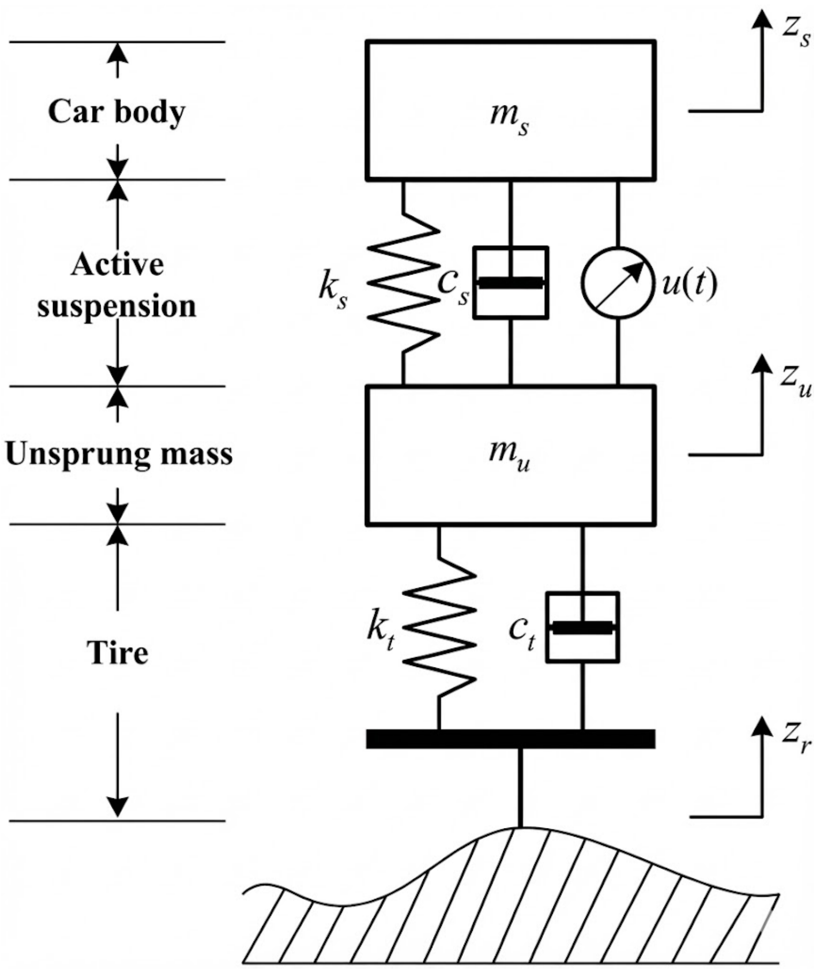
You will simulate a standard 2-DOF quarter-car model, representing one corner of a vehicle (one wheel and the associated body mass above it). The physical structure consists of:

- **Sprung mass (m_s):** portion of vehicle body mass supported by the suspension (cabin + cargo).
- **Unsprung mass (m_u):** wheel, hub, and associated suspension components.
- **Suspension spring (k_s):** elastic element between sprung and unsprung mass.
- Active suspension actuator command (c(t)): controllable suspension command applied between sprung and unsprung mass (you choose c(t) within limits).

- **Tire stiffness (k_t):** vertical stiffness between unsprung mass and road surface.
- **Road displacement r(t):** vertical motion of the road at the tire contact patch (given in the dataset).

Sprung-mass accelerometer a_s(t): measured vertical acceleration of the body/cargo mass (provided in the dataset).

Unsprung-mass accelerometer a_u(t): measured vertical acceleration of the wheel assembly (provided in the dataset).



You are free to choose the exact state-space representation and integration scheme, as long as it is physically consistent (for example: avoid negative damping, respect the command bounds, and ensure numerical stability).

STARK INDUSTRIES

2.1 Quarter-Car Parameters (Use These Values)

Use the following nominal parameters in all your simulations.

Parameter	Value	Description
m_s	290 kg	Sprung mass (vehicle body above suspension)
m_u	59 kg	Unsprung mass (wheel + hub + part of suspension)
k_s	16 000 N/m	Suspension spring stiffness
k_t	190 000 N/m	Tire vertical stiffness
c_min	800 N·s/m	Minimum damping coefficient (fully soft)
c_max	3 500 N·s/m	Maximum damping coefficient (fully stiff)

Your controller must always output a damping command c(t) that satisfies:
c_min ≤ c(t) ≤ c_max.
Any strategy that violates these bounds will be considered physically invalid.

2.2 Actuator Delay

In reality, dampers and their control valves do not react instantaneously. To reflect this, you must model a finite actuator delay between the damping value requested by your controller and the damping actually applied in the dynamics.

For this problem, assume:
• Sampling time $\Delta t = 0.005$ s (200 Hz)
• Actuator delay = 20 ms
⇒ Effective delay = 4 simulation steps

One simple way to implement this is to maintain a FIFO buffer of length 4 for the damping command:
1. At each time step k, compute c_request(k) using your controller and the current state.
2. Clip c_request(k) into [c_min, c_max].
3. Push c_request(k) into the back of a list.
4. Pop the oldest value from the front of the list and apply it as c_applied(k) in the dynamics.

You may use any equivalent discrete-time implementation that produces an effective 20 ms delay between the commanded and applied damping values.

3. Road Profile Dataset

You are given a CSV file containing five pre-generated road profiles. Each profile is a vertical road displacement signal r(t) describing how the road moves under the tire as the vehicle travels forward.

Profile 1: Two half-sine bumps (isolated obstacles).

Profile 2: Smooth wavy road (low-frequency undulations).

Profile 3: Rough asphalt (noise-like high-frequency content).

Profile 4: Speed breaker followed by a sharp dip.

Profile 5: "The coffee run" - mixed sine waves, noise, and a pothole around $t \approx 15$ s.

All profiles are sampled at a fixed rate and stored together in a single CSV file:

File: road_profiles.csv

Columns (exact header format):

- t – time in seconds (0 to 20 s)
- For each k ∈ {1..5}, the dataset includes:

  – profile_k – road displacement r_k(t) in meters

NOTE: You must evaluate the accelerometer's data for both sprung and unsprung mass by yourself to generate inputs for the controller. Refer to the diagram provided.

Header:

t,profile_i

Sampling frequency: 200 Hz ($\Delta t = 0.005$ s)

Duration: 20 s per profile => 4,000 samples per profile.

4. Your Task

You must complete the following steps:

Step 1 — Load Dataset: Read road_profiles.csv, extract the time vector t and, for each profile, the road displacement profile_k and both accelerometer signals acc_s_profile_k and acc_u_profile_k.

Step 2 — Implement Quarter-Car Dynamics: Set up your 2-DOF quarter-car model using the given parameters. Choose a suitable numerical integrator (e.g., explicit Euler, semi-implicit Euler, or 4th-order Runge-Kutta).

Step 3 — Design an Active Suspension Controller: Implement a control law that outputs a command c_request(t) in [c_min, c_max]. The controller may use the sprung and unsprung accelerometer measurements (a_s(t), a_u(t)), vehicle constraints/parameters, and any causal internal memory/state you maintain.

Step 4 — Implement Actuator Delay: Convert c_request(t) into the applied command c_applied(t) using a 20 ms (4-step) delay as described in Section 2.2.

Step 5 — Simulate on All Road Profiles: For each of the five profiles, simulate the full time horizon and record the sprung-mass displacement z_s(t) (required) and unsprung-mass displacement z_u(t) (bonus), along with any signals needed for metric computation (e.g., a_s(t) and jerk).

Step 6 — Compute Metrics: From each simulation, compute the performance metrics defined in Section 5.

Step 7 — Generate submission.csv: Write one row per profile with the required metrics in the specified format and submit this file to Kaggle.

Mandatory report: For each profile, include displacement plots of z_s(t) versus time. Include z_u(t) versus time as well (bonus points if you also keep the unsprung motion well-behaved).

5. Metrics and Cost Function

For each profile, you must compute the following metrics from your simulation results:

- RMS sprung-mass displacement — rms_zs (cargo/body motion)

- Maximum sprung-mass displacement — max_zs (peak cargo/body excursion)

- RMS jerk (sprung mass) — rms_jerk (how rapidly acceleration changes; linked to discomfort/coffee spill)

- Comfort score — comfort_score (a combined index based on displacement and jerk)

Let $z\_s(t)$ and $z\_u(t)$ be the sprung and unsprung vertical displacements, and let $a\_s(t)$ be the sprung-mass vertical acceleration signal used for jerk (typically the same signal your controller observes as the sprung accelerometer). Define displacement relative to the initial value to avoid offset effects:

- Relative sprung displacement: $z\_s\_rel(t) = z\_s(t) - z\_s(0)$

Then compute:

- $rms\_zs = RMS(z\_s\_rel)$
- $max\_zs = max\_t\ |z\_s\_rel(t)|$

Define jerk using a finite-difference approximation ($\Delta t = 0.005$ s):

- $rms\_jerk = RMS(jerk)$
- $jerk\_max = max\_t\ |jerk(t)|$

The comfort score is defined as:

comfort_score = 0.5 · rms_zs + max_zs + 0.5 · rms_jerk + jerk_max

Additional reporting requirement (displacement):

• Compute and record displacement at every time step: For each simulation, record z_s(t) and z_u(t) for all t.

• Include displacement plots in the report: Plot z_s(t) and z_u(t) versus time for each road profile and include these graphs in the report submission.

• Design objective note: The controller should aim to keep the sprung-mass displacement response as steady (low-variation) as practical while also avoiding sharp changes in motion (low jerk).

6. Submission Format

You must submit a CSV file named submission.csv, with one row per profile and the following columns:

Header: profile,rms_zs,max_zs,rms_jerk,comfort_score

Example rows:

profile_1, ..., ..., ..., ...

profile_2, ..., ..., ..., ...

...

profile_5, ..., ..., ..., ...

The profile column must exactly match the identifiers used in the evaluation (profile_1 to profile_5). All metric columns must be numeric and use the same units as defined above. Any missing or malformed values may result in your submission being rejected or scored as zero.

7. Evaluation (Kaggle Leaderboard)

On the backend, the evaluation system has a hidden file reference_metrics.csv containing the metrics produced by Stark's reference controller for each profile.

For each metric $m \in$ {rms_zs, max_zs, rms_jerk, comfort_score}, an accuracy score is computed as:

$$\text{Accuracy\_m} = \left(1 - |\text{pred\_m} - \text{ref\_m}| \;/\; |\text{ref\_m}| \right) \times 100\%$$

Perfect match gives 100% accuracy. A 10% relative error yields 90% accuracy. Negative accuracies (from extreme errors) are clamped to 0 before aggregation.

Per-profile accuracies are combined into a single overall score per profile using fixed weights:

$$\begin{aligned}\text{overall\_accuracy(profile)} = \; & 0.35 \times \text{Accuracy\_rms\_zs} \\ & + 0.30 \times \text{Accuracy\_max\_zs} \\ & + 0.20 \times \text{Accuracy\_rms\_jerk} \\ & + 0.15 \times \text{Accuracy\_comfort\_score}\end{aligned}$$

STARK INDUSTRIES

8. Final Deliverables

In addition to your Kaggle submission, you are asked to submit the following through a separate form at the end of the hackathon:

• Python source file(s) implementing your controller, quarter-car model, and metric computation.

• Your final submission.csv file (profile,rms_zs,max_zs,rms_jerk,comfort_score).

• A short report (2–4 pages) describing your approach, including: modelling choices, controller design, tuning strategy, and key observations.

The report must include displacement plots of z_s(t) and z_u(t) versus time for each road profile (sprung mass is required; unsprung mass handling is evaluated as a bonus).

STARK INDUSTRIES

Include the following details on the cover of your report:

NAME: ______________________________

ENROLMENT NUMBER: ____________________

BRANCH: _____________________________

YEAR: _______________________________

APPROACH (short summary): ______________________________

GIT REPOSITORY LINK: ______________________________