Stark Industries Pvt. Ltd.

PROBLEM STATEMENT – PS3: THE SENTRY ROVER

1. Background

Following a high-altitude EMP strike, Stark Industries' primary autonomous defence systems are offline. The four-wheel sentry rover platform is unusable, and both JARVIS and FRIDAY are corrupted. A low-power bicycle-dynamics failsafe module inside an old traction-control subsystem is the only control hardware still operational.

This module simulates a single "bicycle rover" moving along a one-dimensional track with varying slope and surface conditions. The central command can no longer send complex trajectories; it can only send simple gear commands at a high rate.

Your role is to act as Stark's control systems engineer and design the Eco-Gear controller – a compact gear-shifting brain that keeps the bicycle rover moving efficiently over hostile terrain.

This problem statement defines PS3, which will run as a 15-day Report-based hackathon. Submissions of Git Repo and report will be evaluated at the end for a leaderboard.

2. Objective

You will be given a complete simulation codebase implementing a bicycle-style dynamics model and a template controller file. Your goal is to design and implement a gear-shifting controller that:

- Completes the track by reaching the finish line.
- Respects a strict time limit enforced by the simulator.
- Minimises total energy consumption over the full run.

The IITRMS Autonomous team will evaluate your controller on hidden test tracks. You will not modify the physics engine or track definitions; you only implement the decision logic that chooses the gear. Doing otherwise would hamper the results at the end.

3. What You Interact With

3.1 Track

Each run takes place on a one-dimensional track of fixed length. The track is divided into segments with different terrain and surface properties. For each position along the track, the simulator knows the relevant terrain information.

The exact internal formulas and parameters are handled inside the simulator. You do not need to know the equations; you only see the effect through the state information passed to your controller.

3.2 Controller Interface

At every simulation time-step, the simulator calls your controller function:

```
def get_gear_ratio(x: float,
                   v: float,
                   slope: float,
                   mu: float,
                   track_info: dict) -> float:
```

where, conceptually,

- **x:** Current position along the track (in metres).
- **v:** Current forward velocity (in m/s).
- **slope:** Current terrain gradient (positive = uphill, negative = downhill).
- **mu:** Current surface friction indicator (grip level).
- **track_info:** A dictionary containing metadata about the track, such as:
  - List of track segments.
  - Finish line position.
  - Possibly information about the upcoming segment(s).

Your function must return a single floating-point value called the gear ratio:

- **gear_ratio = 0.0 : coasting (no drive torque; movement only by momentum and gravity).**
- **gear_ratio > 0.0 : powered drive (the engine applies torque and consumes energy).**

There is an upper limit on the allowed gear ratio, defined in the codebase. Any return value outside the allowed range will be clipped or treated as invalid by the evaluation environment.

After each call to your function, the simulator:

• Applies the internal physics model using your chosen gear.

• Updates the bicycle's position, velocity and internal state.

• Tracks cumulative energy usage and simulation time.

• Checks whether the run has finished successfully or has failed due to stalling or time limit violations.

3.3 What You Are Allowed to Do

Inside get_gear_ratio(…) you may:

• Use the inputs x, v, slope, mu and track_info to decide the gear ratio.

• Implement any logic you like using standard Python (conditions, simple state machines, mathematics, etc.).

• Read the provided codebase and documentation to understand the available fields and helper functions.

You may not:

• Change the function signature of get_gear_ratio(…).

• Modify the core simulator, physics, or track-generation code used by the organisers for final evaluation.

• Use external network calls or heavy I/O from within your controller.

• Depend on non-standard external libraries during evaluation.

Your controller should be fast, deterministic and self-contained (Time based evaluation).

4. Codebase and Local Usage

A starter repository will be provided with:

• The bicycle dynamics simulator and core logic.

• A simple visualiser to watch the bicycle traverse the practice track.

• A template controller file: controller_template.py.

• A README with setup instructions and usage examples.

4.1 Getting Started

1) Clone the repository

https://github.com/DSGxRMS/The-Sentry-Rover-Synapse-Drive-25-.git

2) Install dependencies

Follow the instructions in README.md.

3) Run the simulator

python main.py

This will run the simulator on a practice track using the baseline controller implementation. You should see:

- The bicycle moving along the track.
- Basic on-screen information such as time and energy (depending on the current version of the UI).

You are free to operate on the simulator to test your designs.

4) Implement your controller

- Open controller_template.py.
- Implement your own logic in the function get_gear_ratio(…).
- Re-run python main.py to test how your controller performs on the practice track.

You are encouraged to run many local experiments, log relevant data and iterate on your controller design to improve reliability and energy efficiency.

STARK INDUSTRIES

5. Evaluation & Scoring

For official scoring, we will:

- Run your submitted controller on multiple hidden test tracks (not shared in advance).
- For each run, record whether the bicycle:
  - Reaches the finish line inside the time limit, or
  - Fails due to stalling, time limit violations or runtime errors.

For successful runs, the simulator will compute a total energy usage value. For failed runs, a large penalty value will be applied.

Your overall score will primarily be based on energy consumption across all evaluation runs. Lower energy is better, provided the track is completed successfully. Additional metrics such as completion time and stability indicators may be used as tie-breakers between closely performing controllers.

You do not need to hard-code any of these scoring details in your controller; they are handled by us. However, you are expected to maintain a good repository structure and a detailed report of your calculations and evaluation of the physics of the dynamic model.

6. Required Deliverables

Each team must submit the following at the end of the hackathon:

A. Controller Code Repository (Mandatory – via Git)

• A single Python file containing your implementation of get_gear_ratio(…). This will typically be your modified version of controller_template.py.

• **Requirements:**

– Must define get_gear_ratio**(x, v, slope, mu, track_info)** -> float. (If using a language other than python)

– Must not import or require third-party libraries beyond what is already present in the starter repository.

– Must not perform network calls or rely on local external files at evaluation time.

– Must be efficient enough to run at the simulator's internal time-step without timeouts.

The organisers will integrate your controller file into a clean copy of the simulation repository and run the official evaluation suite on hidden tracks.

B. Short Technical Report (Mandatory)

A brief report (maximum 3–4 pages) describing your approach. At minimum, include:

- **Name, enrolment number, branch and year.**
- A clear summary of your controller strategy (how you react to different terrain and conditions).

And preferably:

- Any experiments or observations from practice runs (for example, what helped reduce energy usage while still finishing reliably).
- Known limitations or interesting behaviours of your controller.

Your goal is simple and strict:

Design a robust gear-shifting controller that helps Stark Industries' bicycle failsafe rover complete the track within the time limit while consuming as little energy as possible, across a range of unseen terrain conditions.

ALL THE BEST!