Project Member Application

MAPS

2026-27

For

Mathematics Club

Centre for Innovation
IIT Madras

Instructions

General Instructions

- Mention the following details at the start of your application.

Name:		Insti Name:
Roll:	Room:	Hostel:
Phone (WA):	CGPA:	Email:

- Join the Aspiring PMs Whatsapp Group for further updates.
- The recommended font is a standard font size 11-13.
- The applications have to be submitted in PDF format, named as:
  `<First_name>_<Roll_Number>_Mathematics_Club_Project_Member.pdf`

  For example, `Nihal_EE24B129_Mathematics_Club_Project_Member.pdf`.
- You can upload the finished applications in this Google Form.
- You may submit the completed application on or before **11:59 PM, 4th June 2026.**

Note:

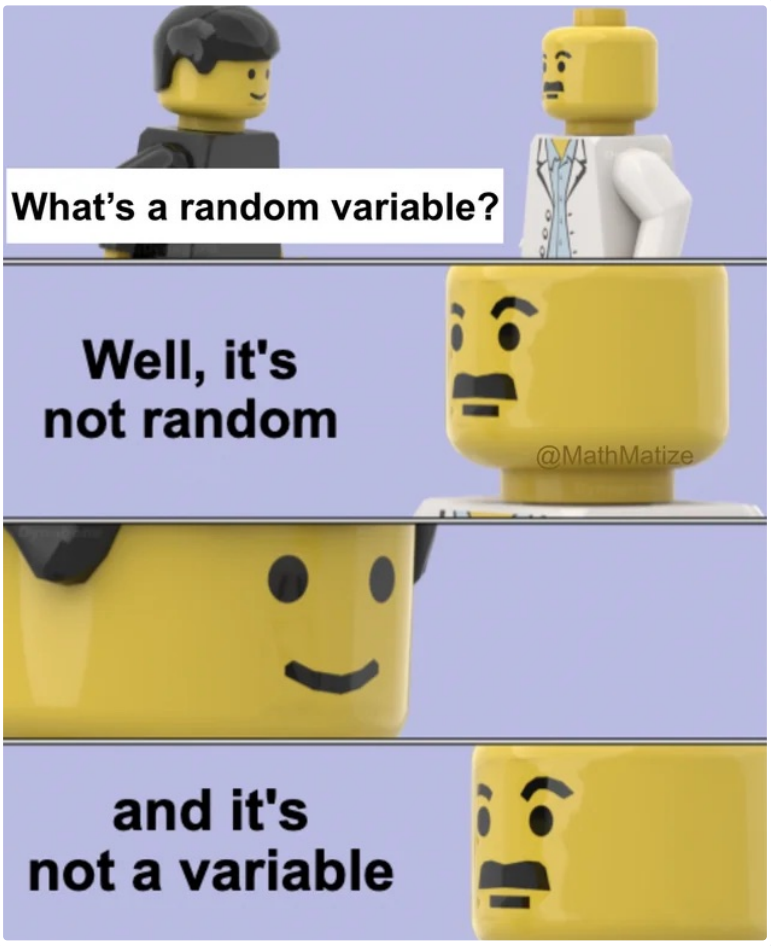
- Clearly explain your reasoning for each question. Your understanding and approach matter more than the final answer.
- Feel free to use online resources for reading and reference while attempting the application. However, your submitted responses should reflect your own understanding and should not be AI-generated.
- It is fine if you don't solve the entire app. Complete as much as you can.
- Focus on the non-bonus questions before attempting the bonus questions.
- If you have any queries, you can reach out to the PLs anytime you want:
  - Nihal Pinisetti (EE24B129) : +91 93848 60255
  - Hafiz Rahman Ellikotil (EE24B104) : +91 95397 90357

Contents

HR

0. Tell us a bit about yourself. Assume you are a master of flexing and proceed.
1. Why do you want to join this project? What skills or qualities do you possess that make you suitable for the work involved?
2. Mention all PoRs/activities you are planning to take this year. Weekly, how much time do you think you will be able to commit to this project? How much time will you commit to other PoRs and academics?
3. What would keep you motivated throughout the period of two semesters? How would you ensure that you are consistent in contributing to the project throughout the tenure?

§1 Dealing with Noise



We already know how to solve perfect deterministic systems. Given the initial conditions, we can calculate the exact state of a system at any future time t.

But physical hardware doesn't care about perfect math. The real world is noisy ~~just like your readings in PH1030 and CY1002, except now you cannot fake them~~. Rover wheels slip, wind gusts randomly, and voltages drift. If we write control loops assuming a frictionless vacuum, our robot will crash.

To build safe autonomous systems, we must mathematically model the unknown. We stop treating physical states as exact numbers and start treating them as Continuous Random Variables. Our goal is to build systems that ensure safety, even in completely unpredictable environments. You can find the basic definitions for general terms such as PDFs, and Expected Values and Variance.

§ The Blind Spot of Averages

Nihal is building a custom proximity sensor for a prototype drone. The sensor outputs a continuous reading $X \in [0, \infty)$ represents the estimated distance to a hazardous obstacle. Due to noise, the accuracy of the sensor drops dramatically at longer ranges. Hafiz analyses the telemetry data and determines that the Probability Density Function (PDF) of the sensor reading X is given by:

$$f_X(x) = \frac{2x}{(1+x^2)^2}, \quad \forall x \geq 0$$

Problem 1.1. Show that $f_X(x)$ is a valid probability density function and derive the Cumulative Distribution Function $F_X(x)$.

Problem 1.2. Nihal decides to use the expected value of the sensor readings $\mathbb{E}[X]$ as the nominal baseline for his control loop. Calculate the expected value $\mathbb{E}[X]$. *Take this as practice for Int Bee :)*

Problem 1.3. To ensure the drone doesn't crash under sudden noise spikes, Nihal wants to calculate the variance $\mathrm{Var}(X)$. Based on your mathematical findings, explain why using standard deviation is completely useless for this specific sensor.

§ The Limits of Uncertainty

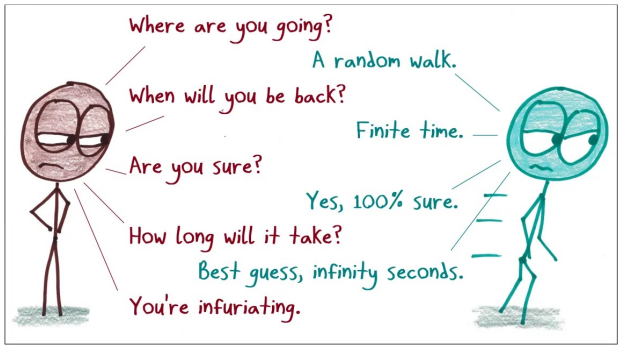
The latency X_i (in ms) at each communication step is an i.i.d. random variable following a standard exponential distribution, $X_i \sim \mathrm{Exp}(1)$. Let $S_n = \sum_{i=1}^{n} X_i$ represent the total cumulative latency over a horizon of n steps.

Problem 1.4. Use Markov's Inequality and Chebyshev's Inequality to find two different upper bounds for the probability that the total latency doubles its expected value, i.e., $\mathbb{P}(S_n \geq 2n)$. Compare how these two bounds behave when the system scales up from a short horizon ($n = 4$) to a long horizon ($n = 100$).

Problem 1.5. Approximate $\mathbb{P}(S_n \geq 2n)$ for $n = 100$ using the Central Limit Theorem (CLT) in terms of the standard normal CDF $\Phi(z)$. Then, write a quick Python script to simulate 10,000 independent runs of S_n for both $n = 2$ and $n = 100$, plotting their normalised histograms against the theoretical CLT bell curves.

Problem 1.6. Look closely at your generated plot for $n = 2$. You will notice that the symmetric CLT curve spills over, assigning a non-zero probability to the event $\{S_n < 0\}$. Explain why this happens.

§ (Bonus) Stochastic Random Walks



The key to any marriage is good communication.

A particle starts at position $x_0 = 0$ and at each step moves as:

$$x_{k+1} = x_k + \xi_k$$

where ξ_k are i.i.d. random variables with $\mathbb{P}(\xi_k = +1) = \mathbb{P}(\xi_k = -1) = \frac{1}{2}$.

Problem 1.7. Define $S_k = \sum_{j=0}^{k-1} \xi_j$. Without assuming independence directly, use the formal definition of variance to show that $\mathrm{Var}(x_k) = k$ by expanding $\mathbb{E}[S_k^2]$ as a double sum:

Problem 1.8. The particle is absorbed at $+N$ (success) or $-N$ (failure). Let $p(x)$ denote the probability of reaching $+N$ before $-N$ starting from position x.

1. Write down the exact functional difference equation that $p(x)$ must satisfy for all internal states $-N < x < N$ and state the two boundary conditions.

2. Solve the system explicitly and evaluate the survival probability from the centre $p(0)$.

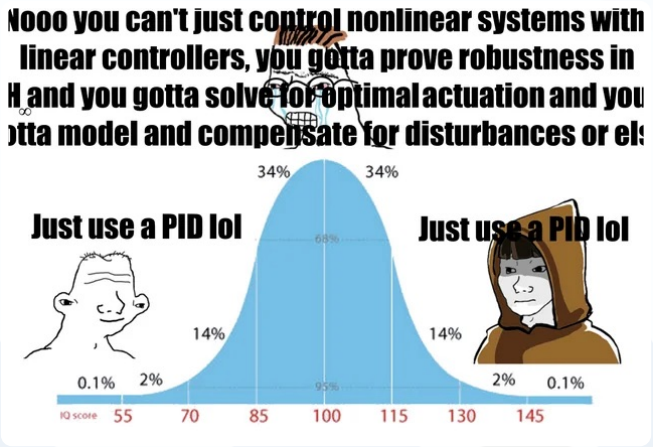
3. Now suppose the walk is instead biased with $\mathbb{P}(\xi_k = +1) = p \neq \frac{1}{2}$. Without solving the full system, use symmetry to explain how $p(0)$ changes when $p > \frac{1}{2}$ versus when $p < \frac{1}{2}$.

Problem 1.9. A constant drift $\mu > 0$ is added to the system dynamics, yielding:

$$x_{k+1} = x_k + \mu + \xi_k$$

Compute $\mathbb{E}[x_k]$ and $\mathrm{Var}(x_k)$.

§2 Understanding the Controller



§ The Bot That Wouldn't Listen

Hafiz and Nihal are testing a wheeled land bot on a steep incline. The bot must drive to a target position $x_{\text{set}} = 10$ m up the ramp, but a constant gravitational rollback force $d = 2$ pulls it backward down the slope. The bot's motion follows the following dynamics:

$$v_{k+1} = v_k + u_k - d \qquad x_{k+1} = x_k + v_k$$

where u_k is the motor force, $e_k = x_{\text{set}} - x_k$ is the tracking error and the bot starts at rest $(x_0 = 0, v_0 = 0)$.

Part 1: PID Mechanics & Stability

Problem 2.1. Nihal first deploys a pure Proportional (P) controller: $u_k = K_p e_k$. Prove mathematically that no finite gain K_p can ever achieve zero tracking error ($e_{\text{ss}} = 0$) at steady state,
To fix this, Hafiz upgrades the system to a Proportional-Integral (PI) controller by adding an accumulation term:

$$u_k = K_p e_k + K_i \sum_{j=0}^{k} e_j$$

Show that if this PI system stabilizes, the steady-state error is forced to be exactly zero. Physically explain where the continuous upward force comes from to balance gravity if the P-term outputs nothing when $e_{\text{ss}} = 0$.

Problem 2.2. Before running the full PI code, Nihal accidentally runs a test with zero integral action ($K_i = 0$) and an aggressively high proportional gain $K_p = 5$.

1. Cast this closed-loop system into standard state-space form $\mathbf{z}_{k+1} = \mathbf{A}_{\text{CL}}\mathbf{z}_k + \mathbf{b}$ using the state vector $\mathbf{z}_k = [x_k, v_k]^T$.
2. Calculate the eigenvalues of $\mathbf{A}_{\text{CL}}$ and evaluate their magnitudes to prove mathematically why this aggressive gain causes the bot to oscillate violently and become unstable.

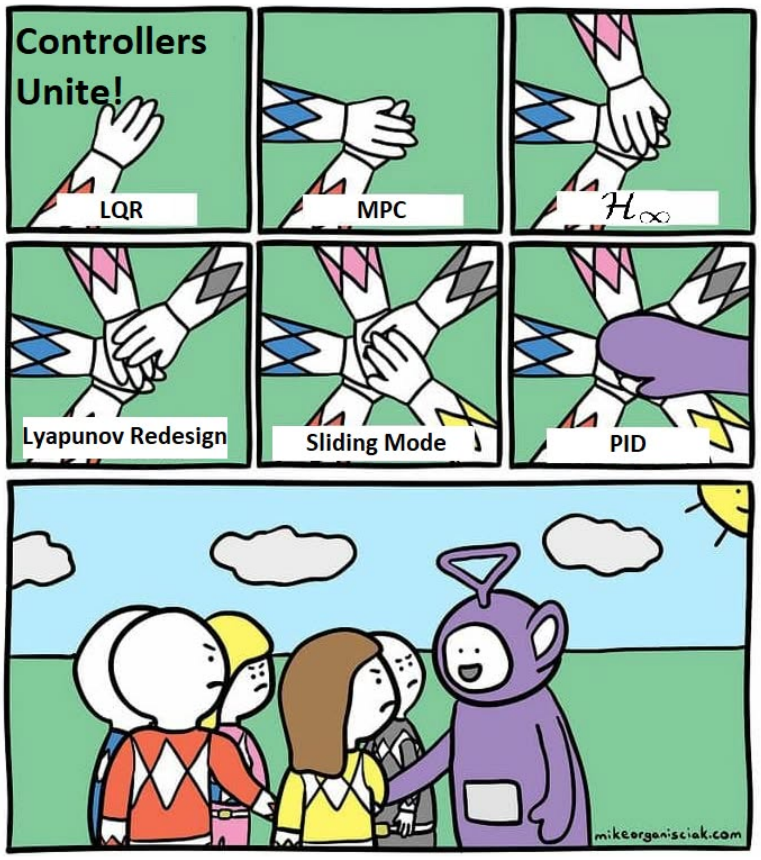
Part 2: PID Pitfalls

Problem 2.3 (Integral Windup). The team commands the bot to make a long run up to a charging dock at the top of the incline located at $x_{\text{set}} = 20$ m. Real-world motors have physical limits, capping the force at $U_{\max} = 3$. During the prolonged climb, the PID equation demands far more than 3, so the physical motor pins at $u_k = 3$.
Explain what happens to the running sum $\sum e_j$ in memory during this climb. When the bot finally crosses the 20 m mark and the error turns negative, explain mathematically why the motor stays completely stuck at maximum thrust, causing a dangerous overshoot past the dock.

Problem 2.4 (Derivative Kick & Sensor Noise). To combat the overshoot, Nihal adds a derivative term to the loop, yielding the complete control law:

$$u_k = K_p e_k + K_i \sum_{j=0}^{k} e_j + K_d(e_k - e_{k-1})$$

The wheel encoder measuring position has small random noise η_k, meaning the computer reads a noisy error $\tilde{e}_k = e_k + \eta_k$. Explain why the derivative term $K_d(\tilde{e}_k - \tilde{e}_{k-1})$ violently amplifies this high-frequency noise even when the bot is barely moving and describe what this rapid switching does to the physical motors.



§ Upgrading to MPC

Hafiz abandons the PID loop entirely. Instead of reacting to past errors, he writes a controller that uses the bot's known physics directly. The bot's continuous-time dynamics on the ramp are:

$$\dot{x} = f(x) + g(x)\, u$$

where the state is $x = [p,\ v]^T$, and explicitly:

$$\dot{p} = v, \qquad \dot{v} = -d + u$$

so $f(x) = [v,\ -d]^T$ captures passive rollback and $g(x) = [0,\ 1]^T$ captures how motor force enters the system. The MPC selects the force u at each instant by minimising the quadratic cost:

$$J = (v_{\text{next}})^2 + R\,u^2$$

where v_{next} is the predicted upcoming velocity and $R > 0$ penalises excessive motor effort.

Problem 2.5 (Deriving u_{MPC}). To predict where the bot will be one step ahead, use a forward Euler prediction with $\Delta t = 1$:

$$v_{\text{next}} = v + \dot{v} = v - d + u$$

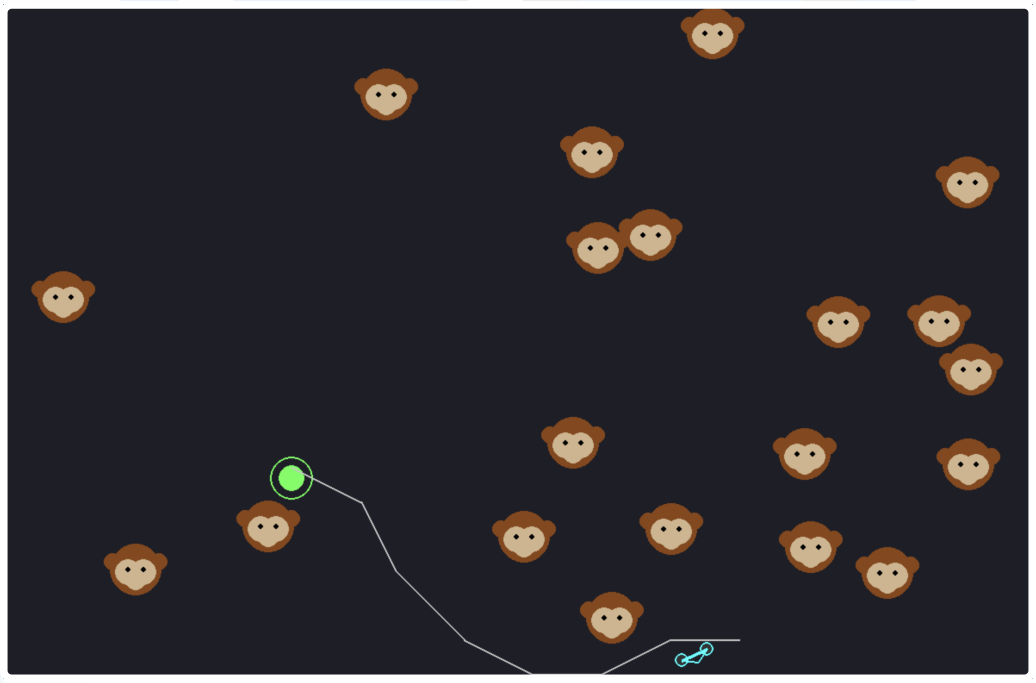
Minimise by computing $\dfrac{\partial J}{\partial u} = 0$ and derive the optimal control law u^*_{MPC} in terms of v, d, and R.
Your answer should clearly separate into two terms: a **feedback term** *proportional to current velocity* v, and a **feedforward term** that directly *cancels the rollback* d. Explain why this feedforward means the bot counteracts gravity instantly, without the slow error accumulation that caused integral windup.

Problem 2.6 (The R Tradeoff). Analyse u^*_{MPC} under two engineering extremes:

1. $R \to 0$**:** Motor effort is free. What does u^*_{MPC} reduce to and what does the velocity dynamics $\dot{v}$ become? What does this mean physically?

2. $R \to \infty$**:** Energy is critically scarce. What happens to u^*_{MPC} and how does the bot behave on the incline?

§3 The Meth behind the Controller



The image above is a digital representation of your life in hostel next year!

If you try to use basic control formulae (like a PID controller) to navigate to class, you are going to get jumped. Basic formulae don't understand the concept of "physical walls" or "aggressive monkeys". To survive the swarm without losing your food, you have to treat your movement as an **Optimization Problem**: *"Find the absolute fastest acceleration to reach my 8 AM, but mathematically guarantee I don't run into a monkey."*

§ The Feasible Region

When a monkey lunges, your Cycle has around 16 milliseconds to dodge. General optimization algorithms are useless here.

Imagine the math your computer navigates as a jagged maze. The solver acts as a marble dropped inside–guessing, backtracking, and getting trapped in "local minima", dead ends that only seem safe. While the math hesitates, you get caught.

Now imagine the problem as a perfectly smooth bowl. Drop the marble anywhere, and it rolls straight to the true bottom every time, finding the optimal escape route in microseconds without getting stuck.

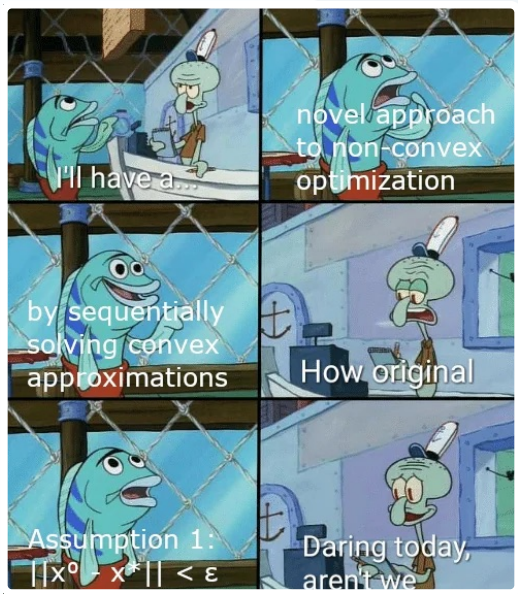
In optimization, that deadly maze is **Non-Convex**, and that smooth bowl is **Convex**.

To solve a Convex Problem, we first need to define a Convex Set

Problem 3.1. Let's model a single monkey as a circle of radius $R = 30$ at the origin. To survive, you must stay outside the monkey, meaning your safe space is defined as $x^2 + y^2 \geq 900$. Using the mathematical line-segment definition of a convex set, prove why this safe region is non-convex.

Problem 3.2. To handle crosswinds, your Cycle's MPC uses a coupled cost function to penalise extreme acceleration: $f(u_x, u_y) = 2u_x^2 + ku_xu_y + 2u_y^2$, where k is an adjustable tuning parameter. Calculate the Hessian matrix. For the solver to guarantee a perfectly convex bowl, what is the strict mathematical range for k? If you mistakenly tune k outside this range, what geometrical shape does the "bowl" turn into, and why can it cause the Cycle's solver to crash?

§ Making the Region Feasible



In **Problem 3.1**, you saw that navigating around a circular obstacle creates a non-convex safe space. As we established, non-convexity is computationally lethal. If you feed that bumpy mathematical maze into your Cycle's real-time solver, the algorithm will hesitate, get trapped in a local minimum and your Cycle will inevitably be overrun by the swarm.

This means we must mathematically force the obstacle to be convex. The Cycle doesn't actually need to know the entire shape of the monkey; it only cares about the exact point on the monkey closest to hitting it right now. If our safety system can dynamically draw an invisible, infinitely long straight line perfectly tangent to that closest point, we can issue an incredibly simple command: *"Forget the circle. Just stay on this side of the flat line."*

A straight line slicing through space is a perfectly convex half-space. But how does a computer calculate that line dynamically every 16 milliseconds based on its state?

We achieve this using Control Barrier Functions.

(Note: The material is mainly meant to introduce the basic ideas behind Control Barrier Functions (CBFs). Some parts go a bit deeper into the math, but you are not expected to follow every derivation; just focus on understanding the intuition and how CBFs work. Feel free to use other resources as well if they help build your understanding.)

Problem 3.3. Based on the given material, explain your understanding of CBFs in your own words. What geometric or physical intuition can you use to describe how they keep a system safe?

Problem 3.4 (Intuition test). When a system reaches the exact boundary of the safe set ($h(x) = 0$), what does the fundamental CBF inequality $\dot{h} \geq -\alpha(h(x))$ mathematically force $\dot{h}$ to be? Explain how this guarantees forward invariance.

§ Understanding Some Code

Knowing the math is necessary, but computers need it formatted perfectly. Our Cycle uses Python solvers (like CVXPy), which categorically refuse to read normal algebra. You have to format your survival instincts into strict Matrix Standard Form:

$$\text{Minimise } \frac{1}{2}x^T Px + q^T x \quad \text{subject to } Ax \leq b$$

Problem 3.5. Below is part of the code for a safety filter written by Nihal. (The equations have been altered; the u_{des} had been generated by a planner.) However, he does not follow good coding practices and has not written any comments.

```python
import cvxpy as cp
import numpy as np

u        = cp.Variable(2)
u_des    = np.array([5.0, 5.0])

objective    = cp.Minimize(cp.sum_squares(u - u_des))

nx, ny           = 0.8, 0.6
boundary_limit   = -2.0

constraints = [u[0]*nx + u[1]*ny >= boundary_limit]

prob         = cp.Problem(objective, constraints)
prob.solve()
```

Can you explain what is going on in the code? Deconstruct the math that is being carried out. Based on your expansion, what are the exact numerical values for the 2×2 matrix P and the 2×1 vector q to put this into the $\frac{1}{2}u^T Pu + q^T u$ format?

Problem 3.6. Hafiz flags this fatal flaw to the team. Eppa agrees that Nihal's code will crash the game and suggests that they rewrite it by introducing a slack variable to act as an emergency pressure valve:

```python
slack        = cp.Variable(1)
objective    = cp.Minimize(cp.sum_squares(u - u_des) + 100000 * slack**2)
constraints = [
    u[0]*nx + u[1]*ny >= boundary_limit - slack,
    slack >= 0
]
```

Explain exactly why this suggestion is a lifesaver. Consider the scenario where the swarm surrounds the Cycle from multiple directions simultaneously, imposing CBF constraints that point in contradictory directions, making the original hard constraint in Problem 3.5 mathematically infeasible with no valid u satisfying it. How does the slack variable mathematically resolve this impossibility and why does the large penalty weight of 100000 ensure the safety boundary is only violated as a last resort?

§4 (Bonus) Going Down the Rabbit Hole

The problems you just survived lay down the foundational math for the Cycle's safety system. But these are just the basics. The real world is infinitely more complicated. If you found this stuff interesting and want to see how deep the robotics rabbit hole goes, here is some further reading.
(Note: This section is entirely optional. We just left these here for the people interested.)

1. **Multi-Agent Collision Avoidance**: This paper uses older game theory with zero CBFs. Why read it? Because upgrading this exact chaotic scenario with modern CBF math is the entire point of this project!
2. **High-Order CBFs (HOCBFs)**: We treated the Cycle as a simple moving dot. Real robots have momentum. HOCBFs handle the brutal reality of acceleration, jerk, and inertia.
3. **Lyapunov Theory**: You probably noticed this word haunting every single resource on CBFs. While CBFs guarantee safety (not crashing), Lyapunov functions guarantee stability (actually reaching the goal without oscillating forever). We skipped it here to spare your sanity.
4. **Some Ancient Control Theory**.

You can share your thoughts on some of these papers :).
If you actually went down the rabbit hole, tell us how it felt! Drop your takeaways, a wild idea for the swarm, or a question that's currently keeping you up at night.


YOU'RE STILL HERE?
IT'S OVER. GO HOME.