Intern Challenge: RRR Robot End-Effector Control

Overview

You will design and implement an end-effector control system for a planar 3-DOF RRR robot to follow a moving green target. The goal is to demonstrate your ability to implement a motion control strategy.

Technical Context

- **Target Motion (Theoretical Reference)** *(not available at runtime)*:
  - `X(t) = 2L`
  - `Y(t) = L · sin(2πft)` with `f = 5 Hz`
- **Target Position Data**: Provided at **30 Hz** to the robot controller
- **Robot Control Loop**: Runs at **1 kHz**

Assignment Tasks

Task A: Basic 2D Cartesian Tracking

Implement controller for the RRR robot's end effector to track the green target using only the target position data (at 30 Hz).

Task B: Change target position frequency and control loop frequency

Change target position frequency to 5Hz and control loop frequency to 50 Hz.

Explain what do you observe?

(optional) Obstacle Avoidance + Joint limit constraints

Extend your solution to avoid a **fixed circular obstacle**:

- Position: `X = L`, `Y = 0.5L`
- Diameter: `L / 4`

Your controller should prevent the end effector from entering the obstacle's area.

Additionally, add and enforce the following constraints in your control algorithm:

- Joint angle position limits

Deliverables

Your submission should include:

1. **Self-contained runnable code**:
    - Either packaged with **Docker** or using a **Python virtual environment** (e.g. via `venv` or `conda`)
    - Include **all necessary dependencies** and simulated inputs
2. **Documentation**:
    - A clear **README.md** with:
        - How to set up and run the code
        - Description of your approach
        - Dependencies and system requirements
    - Well-documented source code
3. **Visualization**:
    - Include a **live or saved visualisation** of the robot and the moving target
    - The visualization should clearly show:
        - The green target trajectory
        - The robot's end effector following it
        - The red obstacle (when applicable)

Evaluation Criteria

✅ **Pass** if:

- Code runs out of the box on another Linux machine using Docker or virtualenv
- Visualization clearly demonstrates the robot following the target
- Code is modular, documented, and logically structured

❌ **Fail** if:

- The code **does not run**
- The **visualization is missing or non-functional**

Notes

- In reality, you do **not** know the theoretical trajectory — your implementation must only use the **measurements** at runtime.
- The robot model and solution can use any custom or OTS library

---