**Ayush Vasantbhai Patel | patelayushvasantbhai@gmail.com**

**Technical Challenge – Hand/Gripper MVP Skill (Option 1)**

PyBullet + Franka Panda (parallel gripper) | Stable grasp with failure detection + mitigation

## 1. Summary

I implemented a minimal manipulation skill that grasps a single object (a cube on a table) using the Franka Panda two-finger parallel gripper in PyBullet. Finite state machine (FSM) is used as the main logic with clear motion primitives (IK-based approach/descend/lift and open/close). One failure mode (bad grasp due to pose error) is detected and mitigated via a recovery routine that retries with adapted parameters.

## 2. Tools, Technologies and Gripper Model

• Simulator: PyBullet (OS: Ubuntu 22.04, Python 3.10)

• Robot model: Franka Panda arm + two-finger gripper (URDF from pybullet_data)

• Scene: plane + table + cube_small (pybullet_data assets)

• Control: PyBullet inverse kinematics + joint position control

## 3. Sensing Assumptions and Interaction Logic

Sensing assumption: The object pose is taken from the simulator (ground truth). In a real system, this would be provided by a perception module (e.g., RGB-D pose estimate) and would include noise.

Policy: The behavior is implemented via FSM and is deterministic.

## 4. Code Structure

src/env.py – simulation setup, world (table/cube), Panda loading, joint/link indexing

src/kinematics.py – grasp target generation and EE orientation helpers

src/motion.py – motion primitives (step_sim, IK move_ee_pose, open/close gripper)

src/policy.py – FSM policy (states, transitions, grasp verification, recovery adaptation)

src/main.py – CLI entry point, wiring, demo mode selection

## 5. How to Run

1) Clone the code from github at a suitable location:
   `git clone https://github.com/PAyush15/gripping-challenge-neura.git`
   `cd gripping-challenge-neura`
2) Create and activate a virtual environment:
   `python3 -m venv pybullet_sim_env && source pybullet_sim_env/bin/activate`

2) Install dependencies:
`pip install -r requirements.txt`
3) Run (success demo):
`python -m src.main`
4) Run (recovery demo):
`python -m src.main --recovery-demo`

### 6. Interaction Logic (FSM)

States: RESET → APPROACH → DESCEND → CLOSE → VERIFY_GRASP → LIFT → HOLD → SUCCESS, with RECOVER as the failure branch and FAIL after 3 retries.

• APPROACH: move end-effector to a safe pregrasp pose above the object.
• DESCEND: move to grasp pose computed from object pose + z_offset.
• CLOSE: close both finger joints.
• VERIFY_GRASP: check contact points between robot and object (min_contacts threshold).
• LIFT/HOLD: lift to a fixed height and hold for a short duration.

### 7. Failure Mode and Mitigation (Recovery Demo)

Failure detected: Unsuccessful grasp (insufficient contacts after closing), which corresponds to a bad approach pose or insufficient grasp depth.

Mitigation: Bounded recovery with retry and parameter adaptation.

Recovery behavior:
1) Open gripper and move to a safe pose above the object.
2) Recompute grasp targets from the current object pose (simulates calculating the object pose again from sensor reading in real systems).
3) Adapt grasp depth: Grasp_Z is reduced by a fixed step per retry until it reaches a min value (to avoid collision with table).
4) Retry from APPROACH (max_retries bounded).

Recovery demo scenario:
• Attempt 1: inject a small XY pose error (models perception/calibration error) so the fingers miss the cube.
• Attempt 2: XY error corrected, but grasp_z is intentionally kept above the object, causing it to fail due to insufficient grasp.
• Attempt 3: RECOVER reduces grasp_z, enabling stable grasp and successful lift.

### 8. Use of AI Tools During Development

AI tools were used for following tasks: clarifying PyBullet API usage, suggesting modular file organization, implementating argument parse to better run the demo (--recovery-demo) and documentation. All manipulation/control decisions (state definitions, transitions, failures, recovery strategy, and parameter tuning) were implemented by me.