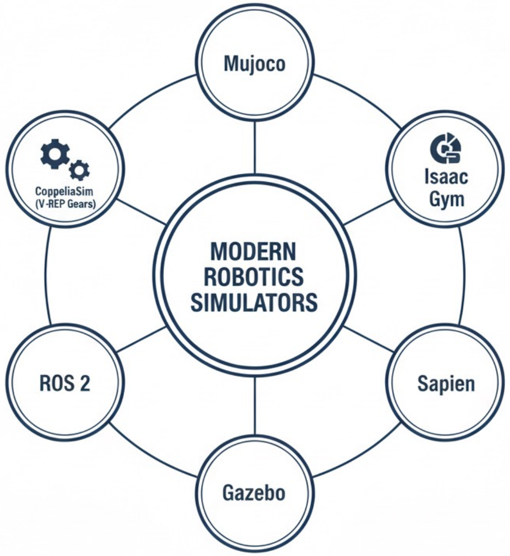
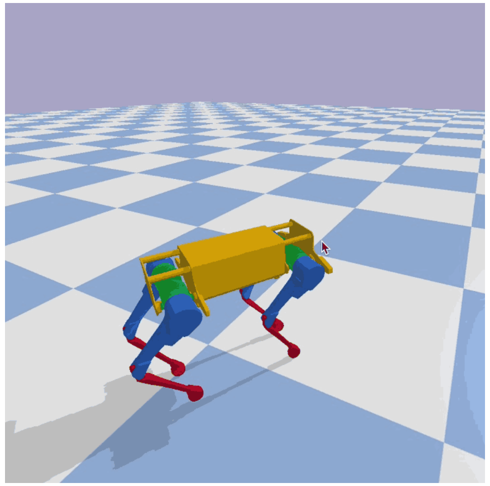
From **PyBullet** to **CALVIN**: Getting Started with Robotic Simulation

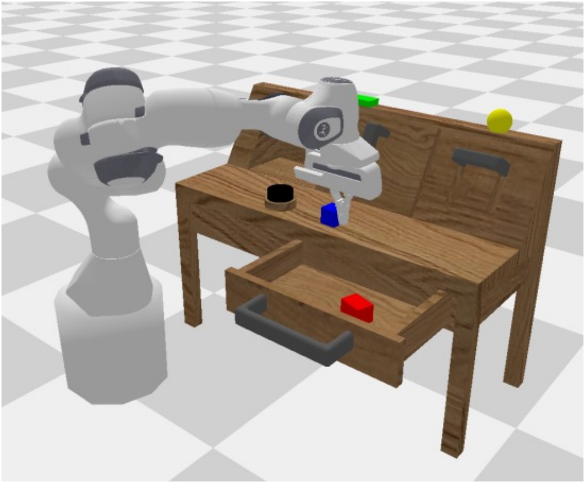
Embodied AI: Perception, Representation and Action

Outline

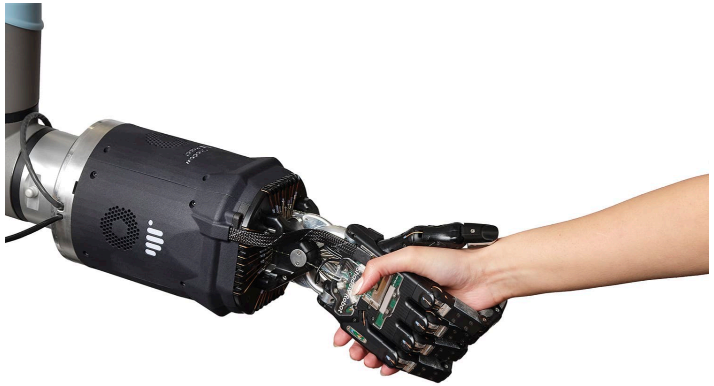
Simulation Engine ↔ **PyBullet** ↔ **CALVIN**

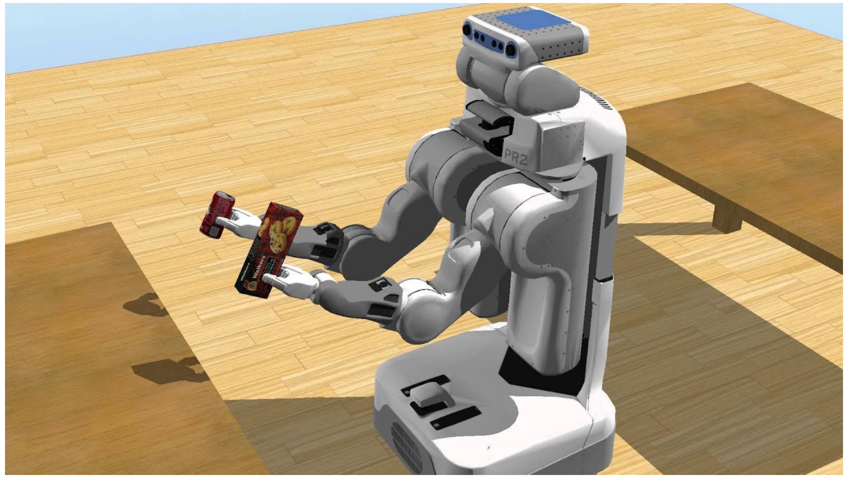






Why simulation?





Real Robots

Cost: Robots are expensive and fragile.

Safety: Bad code can damage hardware or injure people.

Time: Real-world training is slow (1:1 time).

Simulation Envs

Scalability: Run thousands of environments in parallel.

Speed: Train faster than real-time.

Reproducibility: Standardized environments for fair comparison.

Prior to deployment on physical hardware, it is crucial to validate algorithms in a simulated environment.

Robotics Simulator

High-Fidelity Visuals

•*Examples:* NVIDIA Isaac Sim, AI2-THOR, SAPIEN.

•*Pros:* Photorealistic rendering, great for Computer Vision.

•*Cons:* Requires high-end GPUs, complex installation, steep learning curve.

System Integration

•*Examples:* Gazebo (Classic/Ignition).

•*Pros:* Best for ROS/ROS2 integration, mobile robots, and sensors.

•*Cons:* Can be unstable, harder to use purely with Python.

Physics & Learning

•*Examples:* **PyBullet**, MuJoCo.

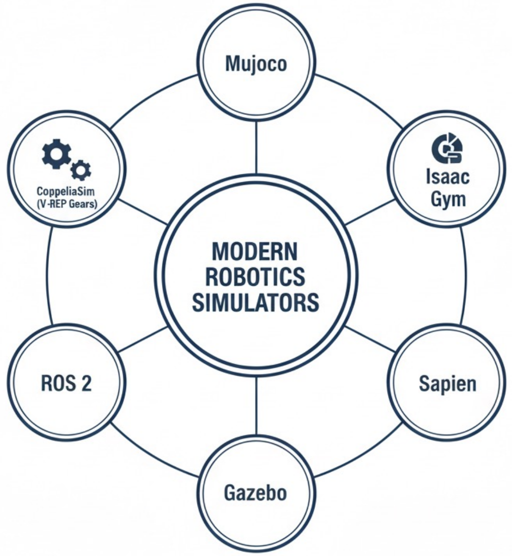
•*Pros:* **Fast**, accurate contact physics, simple Python API, easy to install.

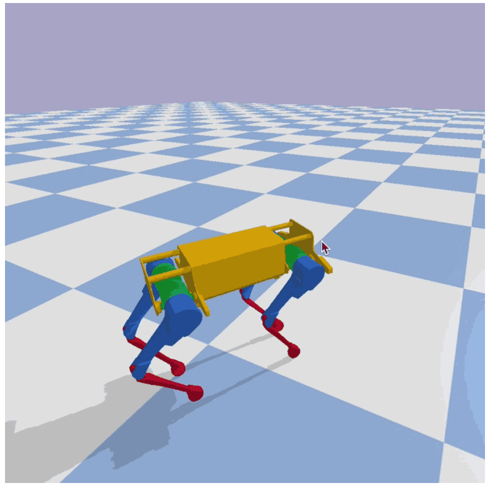
•*Focus:* Ideal for Reinforcement Learning and Robot Manipulation.

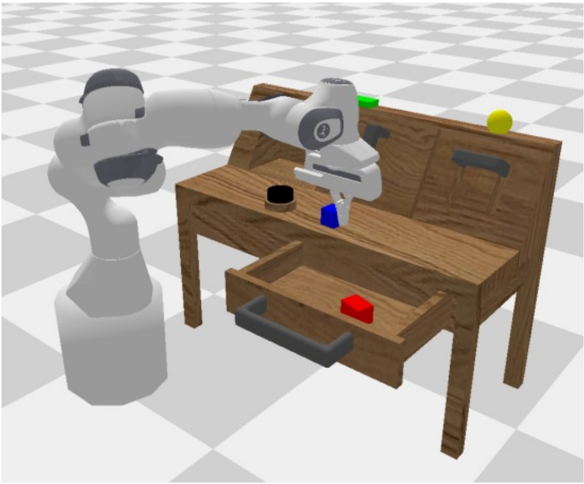
Which simulator should we getting started?

Outline

Simulation Engine ↔ PyBullet ↔ CALVIN







PyBullet: The simplest simulation engine

PyBullet: An open-source physics engine for robotics, games, and visual effects

- **Why PyBullet?**
  - **Easy to use:** Simple Python API. `pip install pybullet`
  - **Standard:** Supports **URDF** (Unified Robot Description Format).
  - **Lightweight:** Runs on CPU (no heavy GPU requirement for basic physics).

- **Core Function:** Calculates rigid body dynamics, collisions, and constraints.

“Hello World” for **PyBullet**

Goal: Set up a basic world with gravity and a floor.

```python
import pybullet as p
import pybullet_data
import time

# 1. Connect to the physics server (GUI mode)
p.connect(p.GUI)
p.configureDebugVisualizer(p.COV_ENABLE_GUI, 0)

# 2. Load the ground plane and set gravity
plane_id = p.loadURDF("plane.urdf")
p.setGravity(0, 0, -9.8)

# 3. The Simulation Loop
while True:
    p.stepSimulation()
    time.sleep(1./240.)
```

Loading a Robot into PyBullet

URDF (Unified Robot Description Format): An XML file that defines the robot's visual and collision properties.

URDF File Structure: Links & Joints

```
# Load a robot at the origin [0,0,0]

robot_start_pos = [0, 0, 0]
robot_start_orn = p.getQuaternionFromEuler([0, 0, 0])
robot_id = p.loadURDF("franka_panda/panda.urdf",
robot_start_pos, robot_start_orn, useFixedBase=True)
```

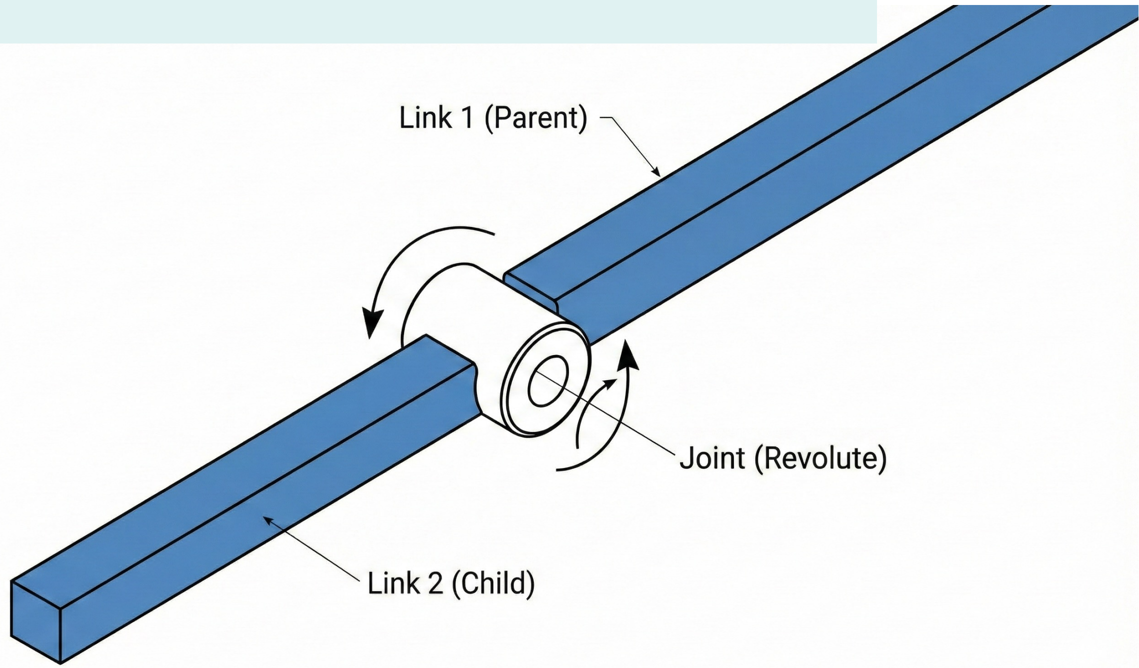
Understanding URDF: The Robot's "DNA”

How Robots are Defined in Simulation?

What is URDF? **U**nified **R**obot **D**escription **F**ormat.
•It is an **XML file** that tells the simulator: "Here is a robot, here are its parts, and here is how they move."
•Think of it as the **blueprint** or **DNA** of the robot.

The Structure: A Tree of Links and Joints

- **Links:** The rigid body parts (e.g., the forearm, the wrist). They possess mass and shape.
- **Joints:** The hinges that connect two Links (Parent → Child). They define motion (rotate, slide, fixed).
- **The Rule:** A robot is a chain. Base Link → Shoulder Joint → Shoulder Link...



The Components of URDF

The 3 Key Components of a Link:

1.<visual>: What the camera sees.

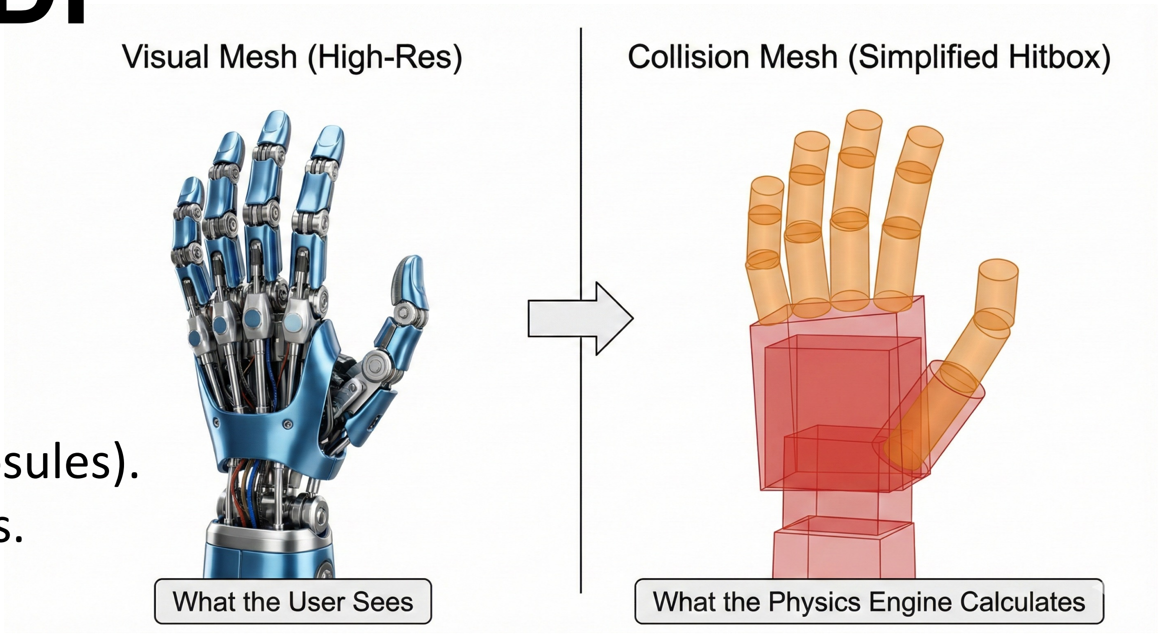
1. Usually high-resolution 3D meshes (.obj, .stl).
2. *Purpose:* Rendering for human eyes.

2.<collision>: What the physics engine "feels".

1. Usually simplified shapes (Cylinders, Boxes, Capsules).
2. *Purpose:* Fast calculation of contacts and bumps.

3.<inertial>: The physics properties.

1. Mass (kg) and Inertia Matrix.
2. *Purpose:* Gravity and Force calculations.




```xml
<link name="upper_arm">
  <visual>
    <geometry><mesh filename="arm.obj"/></geometry>
  </visual>
  <collision>
    <geometry><cylinder radius="0.05" length="0.3"/></geometry>
  </collision>
</link>
```

Controlling the Robot in PyBullet

Motor Control: We don't just "teleport" the robot; we apply forces or set targets for motors.

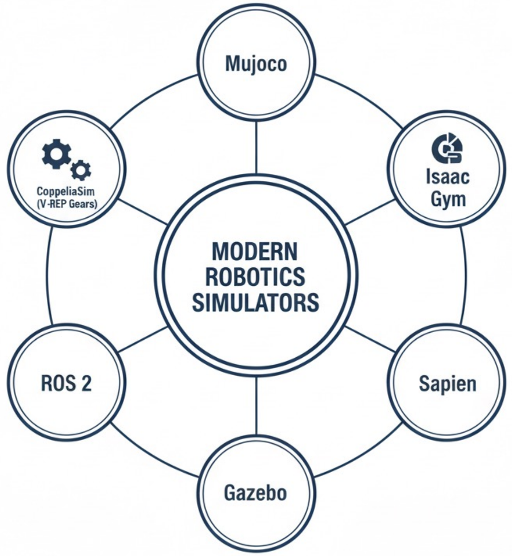
Control Mode: POSITION_CONTROL (most common for beginners).

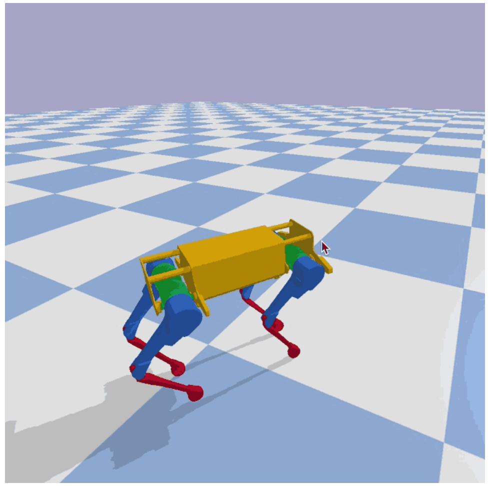
```python
# Move Joint 1 to a specific angle
target_position = 1.57 # Radians (approx 90 degrees)

p.setJointMotorControl2(
bodyUniqueId=robot_id,
jointIndex=1,
controlMode=p.POSITION_CONTROL, # POSITION, VELOCITY, TORQUE
targetPosition=target_position,
force=500
)
```

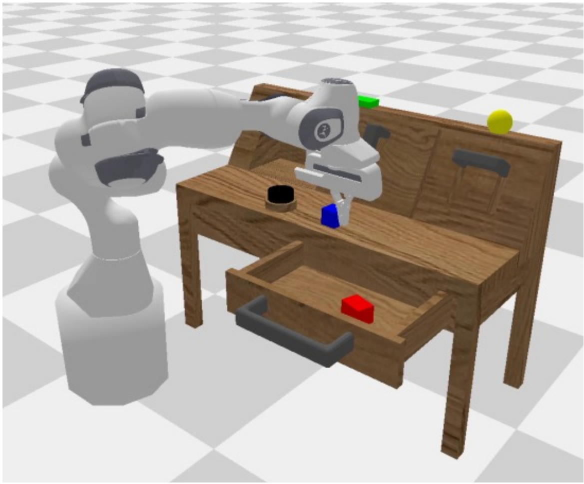
Demo on PyBullet
From zero to a simulation env

Outline

Simulation Engine ⟷ PyBullet ⟷ CALVIN





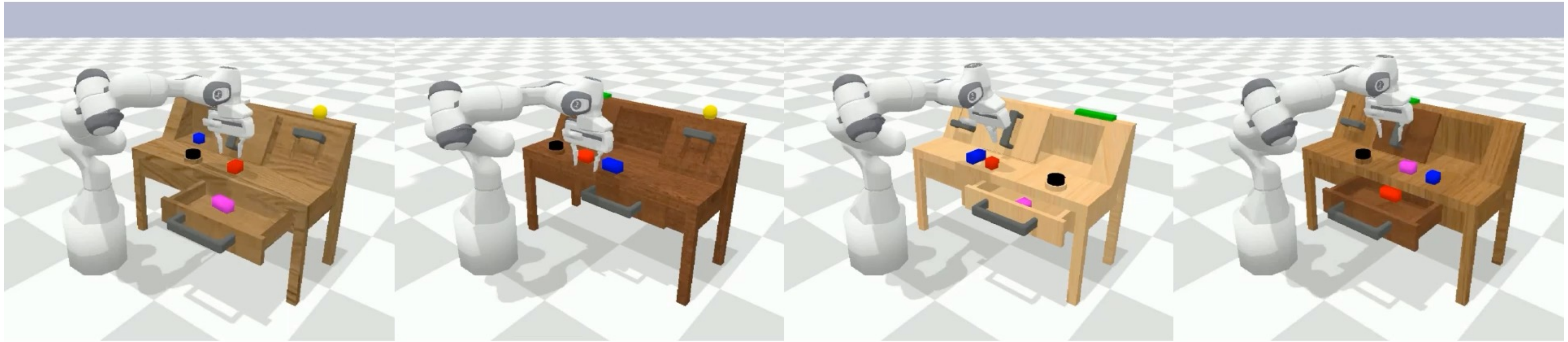


From Engine to Benchmark: CALVIN

Composing **A**ctions from **L**anguage and **V**ision **IN**strumentation.

The Setup:

- **Robot:** Franka Emika Panda (7-DoF Arm).
- **Scene:** A desk with interactive objects (drawers, sliding doors, blocks, switches).
- **Sensors:** Static cameras & Gripper camera.

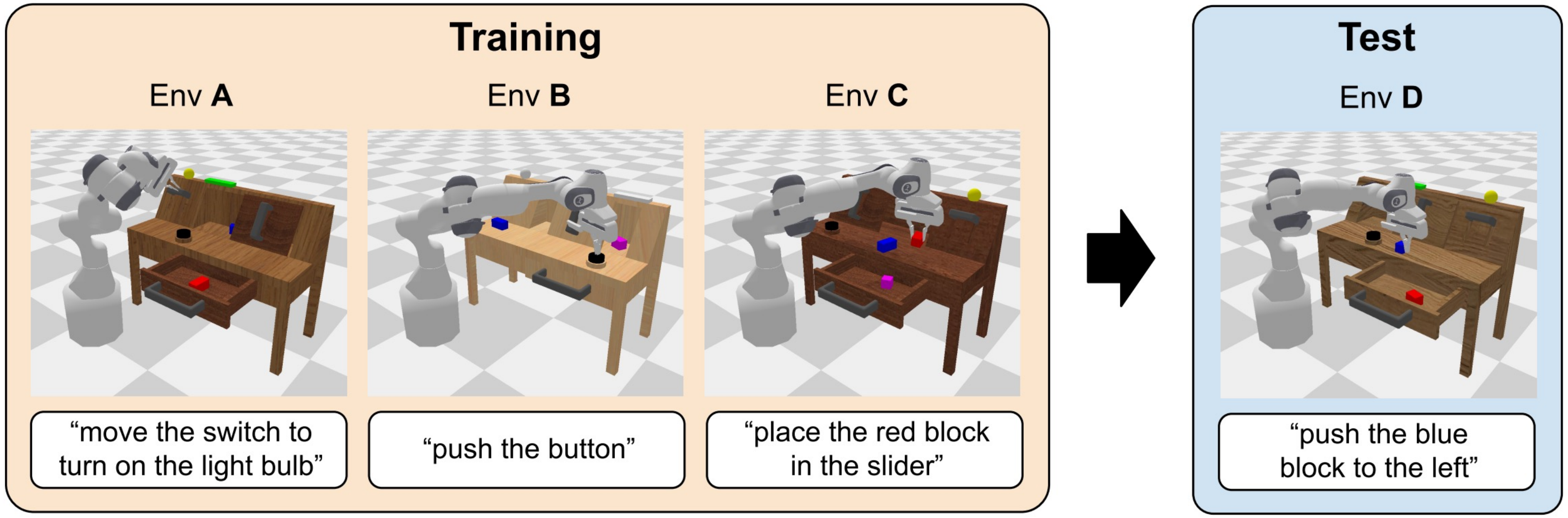


"Rotate the red block to the right" "Open the drawer" "Place the red block in the sliding cabinet" "Move the sliding door to the right"

CALVIN Challenge

The Challenge: Robots are notorious for **Overfitting**. If you train a robot on a white table, it often fails on a wooden table.

CALVIN provides **4 distinct environments** (Env A, B, C, and D) with different desk textures, lighting, and object placements.
Training (Seen): You train your agent on Environments **A, B, and C**.
Evaluation (Unseen): You test your agent on Environment **D**.



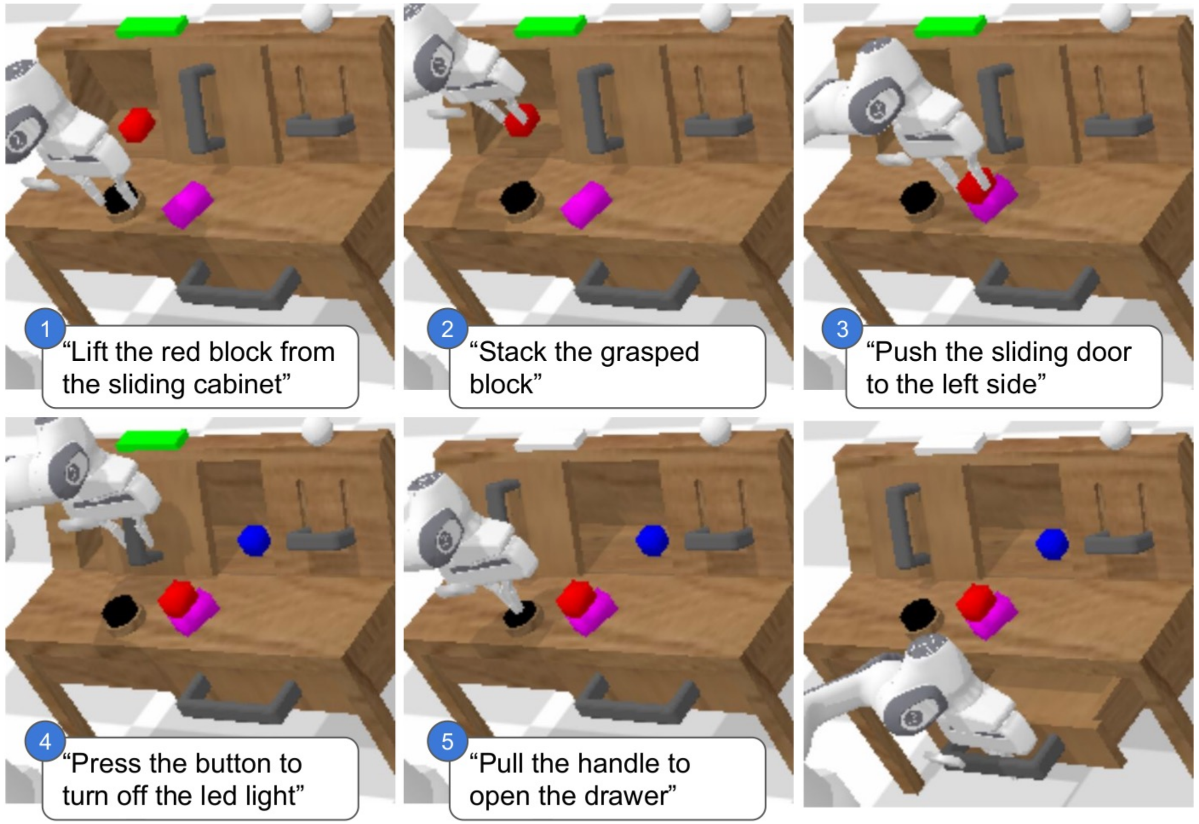
Long Horizon Task

Short-Horizon (Single Task):

- The robot executes one isolated command, then resets.
- *Example:* "Open the drawer." (If it succeeds, task over).

Long-Horizon (CALVIN's Focus):

- The robot must execute a **sequence of instructions** continuously in the exact same environment without any human intervention or resets.
- *Example Sequence (5 steps)*



Why is this so difficult?

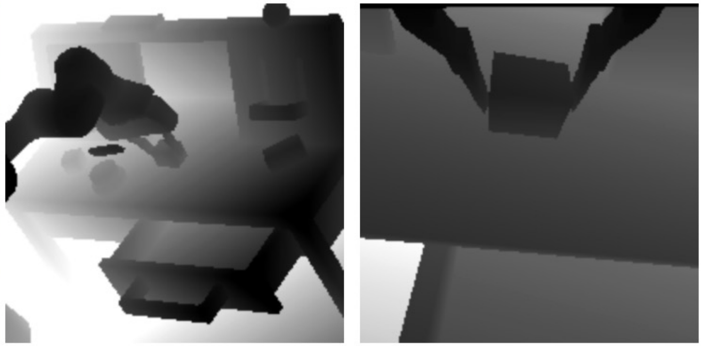
→ Compounding Errors

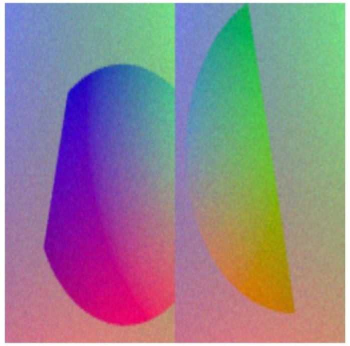
Long Horizon Task Demos

CALVIN Observations

What does the robot "see"?

- **Visual (rgb_obs):** Images from
the front camera and the gripper camera.
- **Depth (depth_obs):** Distance information for each pixel.
- **Vision-based Tactile Sensing**
- **Proprioception (robot_obs):** The robot's own joint states (angles, velocities) and gripper width.
- **Language Instruction:** Text goals like *"Open the drawer"* or *"Push the blue block"*.





CALVIN Action Space

How do we control the CALVIN robot?

Action Format: A 7-dimensional vector (Relative Motion).

•[dx, dy, dz]: Change in End-Effector Position.

•[dr, dp, dy]: Change in End-Effector Orientation (Euler).

•[gripper]: Binary command (-1 to close, +1 to open).

Why Relative? It is often easier for AI to learn "move a little to the left" than "go to coordinate X,Y,Z".

Homework 1

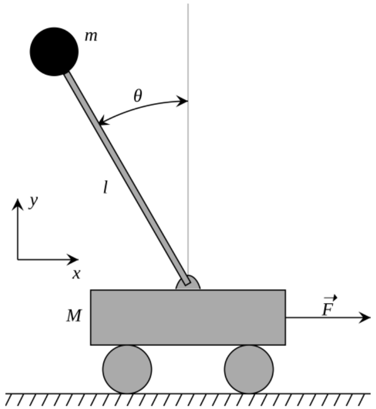
Includes three parts:

1. Forward Kinematics & Inverse Kinematics
2. Dynamics
3. PID & Computed Torque Control

DDL: FEB 27 23:59

Group Submission with a report and supplementary materials

The detailed instruction and reference code are posted on **Moodle.**



Cart-Pole System

Thanks for listening

Q & A