Report

Tutorial

To run the teleoperation interface:

```
python teleop.py
```

This launches a GLFW window with a keyboard-driven UI. The robot gripper can be moved in real time, and teleoperation data is logged automatically.

Keyboard controls:

- **Translation**
  - `W/S` +/− Y
  - `A/D` −/+ X
  - `T/G` +/− X
- **Rotation (about local axes)**
  - `B/N` Yaw
  - `C/V` Pitch
  - `Z/X` Roll
- **Gripper control**
  - `[` Close
  - `]` Open

At the end of a trial (either when the peg insertion is successful or when the maximum step limit is reached), the simulator shuts down and all log data is written to **`teleop_log.csv`**.

To run the simple_sim

```
python simple_sim.py
```

You can adjust the RTF in the realtime factor of the sim.

Description

The system implements a physics-rich peg-insertion environment with:

1. Real-time teleoperation using keyboard controls.
2. Cartesian gripper control via mocap.
3. Detailed logging of commanded vs. actual states, peg pose, and contact forces.

4. Modular input handling (keyboard can be replaced by other devices).

This setup can be used as a base for testing human teleoperation, scripted agents, or reinforcement learning policies in a contact-rich manipulation task.

The simulation was developed in **MuJoCo**

Scene composition:

- **Gripper:** Gripper mjcf was downloaded from the mujoco menagerie.
- **Peg:** Modeled as a box collider with a mesh overlay. Give .stl file was used for visual mesh
- **Cuboid:** The provided STL was used for visuals. For collisions, an analytic box-with-hole geometry was defined to ensure robust peg–hole alignment.
- **Table:** Infinite plane geometry with frictional contact.

Please see scene.xml for more details.

Handling Contacts and Gripper Constraints

- MuJoCo's contact pipeline was used to detect and resolve peg–table, peg–cuboid, and gripper–peg contacts.
- At each step, `mj_contactForce` extracts normal (`fn`) and tangential (`ft1`, `ft2`) forces. Contacts are aggregated and logged per pair.

Unachievable gripper commands:

- The gripper opening is commanded via a 1-DOF floating-point "jaw separation" parameter.
- If the commanded jaw separation is smaller than the peg width, the physics engine prevents penetration:
  - The action command moves the fingers inward.
  - Collision detection halts further motion.
  - The actuator applies maximum available force, approximating a realistic "stall" condition without object interpenetration.

Potential Improvements

- **More accurate contact modeling:** Replace box approximations of the hole with a true CSG-defined geometry or voxel-based collision mesh.
- **Force sensors:** Add virtual 6-DoF force/torque sensors on the gripper fingers for richer feedback.
- **Learning agents:** Implement reinforcement learning policies for autonomous peg insertion, using the provided `step()` and `reset()` APIs.

- **Better haptics in teleoperation:** Extend the interface to support a joystick or VR controller for more intuitive control than keyboard keys.
- **Stability at fine tolerances:** Explore constraint-based insertion (e.g., using MuJoCo equality constraints or contact margin tuning) to make the final millimeter-scale alignment more robust.

Implementation

`keyboard_glfw.py`

This module defines the **keyboard input device**. The central class is:

```
class GlfwKeyboardDevice(InputDevice):
```

- Handles **GLFW key callbacks**, mapping each key to a Cartesian or rotational velocity.
- Defines **per-axis speed gains** (e.g., XY, Z, roll, pitch, yaw), which control how aggressively inputs are applied.
- Provides a `poll()` method that outputs the current command vector.

The base class `InputDevice` is intentionally generic, so the keyboard controller can easily be swapped with another device such as a joystick, gamepad, or VR controller.

`peg_insertion_env_mujoco.py`

This file defines the **simulation environment** in an OpenAI Gym–style API.

- **Model loading:** Loads the provided MJCF (XML) model into MuJoCo.

- **Mocap-based control:** The gripper is controlled via a mocap body welded to the base. This allows direct Cartesian control (position + orientation) without needing a low-level joint-space controller.

Core methods:

- `step(action)`
  Takes an 8D action vector `[px, py, pz, qw, qx, qy, qz, grip]`.
  - Updates the mocap pose and jaw separation.
  - Calls `mj_step` to advance the physics.
  - Returns `(obs, reward, terminated, info)`.
  - Currently, `reward = 0.0`, but this is designed so RL reward functions can be added later.
  - `info` includes gripper pose, jaw separation, and peg pose.
  - `obs` includes raw `qpos` and `qvel`.

- `reset(qpos=None, qvel=None)`
  Resets the simulation to its initial state (or to a given configuration).
- `set_mocap(pos, quat)`
  Helper to set the mocap-controlled gripper pose.
- `has_peg_inserted()`
  Determines task success by checking whether the peg tip site is below the hole entry site, within a distance threshold.
- `_label_from_geom(geom_id)`
  Maps MuJoCo geom IDs to semantic labels (`"gripper"`, `"peg"`, `"cuboid"`, `"table"`). All finger/tongue/pad geoms are collapsed into `"gripper"`.
- `_contacts_summary()`
  Aggregates contacts between tracked bodies. Returns total normal force, tangential forces, and the number of active contact points.
- `get_jaw_separation()`
  Computes current jaw opening from fingertip site positions.

`teleop.py`

This is the **main entry point** for teleoperation.

- Initializes the peg insertion environment and opens a GLFW rendering window.
- Implements **camera manipulation** (orbit, pan, zoom) via mouse inputs.
- Runs the teleop loop in **real time**:
  - Physics steps are integrated using MuJoCo's internal `dt`.
  - A pacing loop compares simulation time with the wall clock and sleeps as needed, so the simulation speed matches real time (or adjusted by a real-time factor).
- Uses a logger to save all commanded actions, gripper states, peg states, and contact forces into a CSV file for later analysis.

Deliverables

- **Source code:** Environment (`peg_insertion_env_mujoco.py`), teleop interface (`teleop.py`), keyboard input device (`keyboard_glfw.py`), and simulation implementation for sending constant command to sim (`simple_sim.py`)
- **Demo video:** Shows the full sequence from pickup to insertion.
- **Log file:** `teleop_log.csv` records commanded poses, actual poses, peg states, jaw separation, and contact forces.
- **This report:** Summarizes design choices, contact handling, challenges, and improvements.