Franka Panda RL - Door Manipulation Project 🦾🚪

> **University Project** - Reinforcement Learning per Manipolazione Robotica (Apertura e Chiusura Porte).
>
> 🇮🇹 **Vai alla versione Italiana** 🇬🇧 **Go to English Version**

🇮🇹 Versione Italiana

Questo repository ospita un progetto universitario volto all'esplorazione e applicazione di algoritmi di **Reinforcement Learning (RL)** per task di manipolazione robotica complessa utilizzando un braccio **Franka Emika Panda** nell'ambiente di simulazione **Robosuite**.

L'obiettivo principale è addestrare un agente capace di **Aprire** e **Chiudere** una porta, non solo in condizioni deterministiche ideali, ma dimostrando capacità di **Generalizzazione** attraverso tecniche di *Curriculum Learning* e *Domain Randomization*.

📌 Punti Focali del Progetto

1. **Algoritmo State-of-the-Art**: Utilizzo di **Soft Actor-Critic (SAC)** per la sua efficienza (sample efficiency) e capacità di esplorazione tramite massimizzazione dell'entropia.
2. **Reward Shaping Avanzato**: Progettazione di funzioni di ricompensa dense che guidano l'agente fase per fase (avvicinamento, manipolazione, mantenimento).
3. **Gestione Post-Successo (Return Stage)**: Implementazione di una logica che "educa" il robot a non collassare dopo il successo, ma a ritrarre il braccio o mantenere una posa stabile.
4. **Generalizzazione Robusta**:
    - **Apertura**: Curriculum a Stadi (*Teacher-Student*) che varia la difficoltà del goal e la fisica della porta.
    - **Chiusura**: Curriculum Adattivo che incrementa progressivamente l'incertezza sulla posizione della porta (*Domain Randomization*).

📂 Struttura della Codebase

- `config/`: Contiene le dataclass di configurazione (`TrainConfig`). Qui vengono definiti iperparametri del SAC (learning rate, batch size, tau) e pesi della Reward Function. Separare la configurazione dal codice rende gli esperimenti riproducibili.
- `train_open.py` / `train_close.py`: Script per l'addestramento nei task deterministici (senza randomizzazione). Utili come *baseline*.
- `open_generalized/`: Modulo per il task di **Apertura** generalizzato.
  - `teacher.py`: Classe `StageTeacher` che definisce i livelli di difficoltà.
  - `train_curriculum.py`: Loop di addestramento che interagisce con il Teacher.
- `close_generalized/`: Modulo per il task di **Chiusura** generalizzato.
  - `env_gen.py`: Estende l'ambiente base aggiungendo randomizzazione della posa della porta.
  - `train_gen.py`: Implementa una callback che alza il livello di difficoltà in base al *Success Rate*.

🧠 Dettagli Implementativi (Il "Motore" del Sistema)

1. L'Algoritmo: Soft Actor-Critic (SAC)

È stato scelto SAC (implementazione *Stable Baselines3*) perché è un algoritmo **Off-Policy** (riutilizza i dati passati tramite Replay Buffer) e **Maximum Entropy**. * **Entropia**: Il parametro `ent_coef="auto"` permette all'agente di bilanciare esplorazione e sfruttamento autonomamente. In task di manipolazione precisi come questo, è fondamentale per evitare minimi locali (es. il robot che si blocca vicino alla maniglia senza afferrarla).

2. Reward Function (Analisi Matematica)

La funzione di ricompensa R_t non è binaria (0 o 1), ma "shapartata" (dense) per guidare l'apprendimento.

Per il Task di Apertura:

La reward è composta da: $R_{total} = w_{prog} \cdot P + w_{delta} \cdot \Delta + R_{bonus} - R_{costi}$

- **Progress (P)**: Il progresso normalizzato [0, 1] dell'angolo della porta verso il target.
- **Delta (Δ)**: La velocità di apertura corrente. Premia il movimento attivo verso l'apertura.
- **Success Bonus (≈ 5.0)**: Un premio sparso grande quando la porta supera l'angolo target.

- **Return Stage (Anti-Collasso)**: Una volta aperta la porta, se `enable_return_stage=True`, l'agente riceve reward se riporta l'end-effector alla posizione iniziale. Questo previene che il robot, ottenuto il bonus, spinga eccessivamente o cada, garantendo un comportamento "pulito".

Per il Task di Chiusura:

Simile all'apertura, ma con un focus sulla *chiusura completa* (latch). * In `close_generalized`, viene aggiunta una **penalità di velocità** (`arm_vel`) quando la porta è chiusa, per insegnare al robot a stare "fermo" e non tremare una volta finito il lavoro.

3. Strategie di Generalizzazione

È qui che il progetto va oltre la semplice esecuzione.

A. Curriculum a Stadi (Apertura)

Definito in `open_generalized/teacher.py`. Utilizziamo un approccio a stadi discreti (S0 → S4). * **Variabili**: * `goal_angle`: All'inizio chiediamo di aprire la porta solo del 5-20% (S0). Alla fine, fino al 100% (S4). * `friction` & `damping`: Variano le proprietà fisiche dei cardini della porta. In S4, l'attrito può essere molto alto (porta pesante) o molto basso (porta leggera), costringendo la rete neurale ad adattare la forza applicata. * **Promozione**: Se il *success rate* negli ultimi 200 episodi supera l'**85%**, il `StageTeacher` promuove l'agente allo stadio successivo.

B. Curriculum Adattivo Continuo (Chiusura)

Definito in `close_generalized/env_gen.py` e gestito dalla callback in `train_gen.py`. * Non ci sono stadi fissi, ma un `curriculum_level` continuo $\alpha \in [0, 1]$. * **Randomizzazione**: La posizione (x, y, z) e rotazione (yaw) della porta vengono perturbate da un rumore proporzionale ad α. * $\text{Noise}_{pos} \sim U(-0.15\alpha, 0.15\alpha)$ * $\text{Noise}_{rot} \sim U(-0.30\alpha, 0.30\alpha)$ * **Logica**: L'agente inizia con la porta ferma (facile). Man mano che diventa bravo (SR > 0.85), la porta inizia a "spostarsi" e "ruotare" tra un episodio e l'altro, rendendo la policy robusta a errori di calibrazione o posizionamento.

4. Spazio delle Osservazioni e delle Azioni

- **Action Space (7-DoF + Gripper)**: Il robot è controllato in spazio di giunti o operativo. Le azioni sono normalizzate in [-1, 1].
- **Observation Space**: L'input alla rete neurale è un vettore piatto contenente:
  - `robot0_eef_pos`: Posizione (x,y,z) dell'end-effector.
  - `robot0_eef_quat`: Orientamento (quaternione) dell'end-effector.
  - `robot0_gripper_qpos`: Stato di apertura delle pinze.
  - `door_angle` (e info correlate): Stato fisico dell'oggetto da manipolare.
  - (Nel generalized) `goal_norm`: Il target corrente normalizzato e info sullo stadio del curriculum.

5. Iperparametri Chiave

Questi parametri (da `TrainConfig`) sono stati ottimizzati per garantire stabilità:

Parametro	Valore	Descrizione
Learning Rate	`3e-4`	Tasso di apprendimento standard per Adam.
Batch Size	`256`	Dimensione del batch per update del gradiente.
Buffer Size	`1,000,000`	Dimensione memoria (Replay Buffer).
Gamma (γ)	`0.99`	Fattore di sconto per reward futuri.
Tau (τ)	`0.005`	Soft update per le target networks.
Learning Starts	`10,000`	Step puramente casuali iniziali per popolare il buffer.
Architettura Rete	`[256, 256]`	Due layer nascosti per Actor e Critic.

🛠️ Istruzioni per l'Uso

Prerequisiti

- Python 3.10+
- Ambiente Virtuale raccomandato

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

Comandi Rapidi (Makefile)

Il file `Makefile` espone i comandi principali.

Azione	Comando	Descrizione
	`make train_open_gen`	Avvia il training Apertura con Curriculum a Stadi

Azione	Comando	Descrizione
Addestramento (Gen)		
Addestramento (Base)	`make train_open`	Avvia il training Apertura standard (deterministico)
Play / Test	`make play_open_gen`	Visualizza a schermo il modello addestrato (Apertura)
Chiusura	`make train_close_gen`	Avvia il training Chiusura con Randomizzazione Posizionale
Monitoraggio	`make board`	Avvia TensorBoard per vedere i grafici

🇬🇧 English Version

This repository hosts a university project exploring **Reinforcement Learning (RL)** for complex robotic manipulation tasks using a **Franka Emika Panda** arm within the **Robosuite** simulation framework.

The primary goal is to train an agent capable of **Opening** and **Closing** a door, demonstrating robustness through **Generalization** techniques such as *Curriculum Learning* and *Domain Randomization*.

📌 Project Focal Points

1. **State-of-the-Art Algorithm**: Utilization of **Soft Actor-Critic (SAC)** for its sample efficiency and entropy-based exploration capabilities.
2. **Advanced Reward Shaping**: Design of dense reward functions that guide the agent through phases (approach, manipulation, stabilization).
3. **Post-Success Management (Return Stage)**: Implementation of logic that teaches the robot to retract or hold a stable pose after success, rather than collapsing or drifting.
4. **Robust Generalization**:
    - **Opening**: Stage-Based Curriculum (*Teacher-Student*) varying goal difficulty and door physics.

- **Closing**: Adaptive Curriculum progressively increasing uncertainty in door placement (*Domain Randomization*).

📂 Codebase Structure

- `config/`: Contains configuration dataclasses (`TrainConfig`). separating hyperparameters (learning rate, weights) from logic ensures reproducibility.
- `train_open.py` / `train_close.py`: Training scripts for deterministic tasks (baseline).
- `open_generalized/`: Module for the **Generalized Opening** task.
  - `teacher.py`: The `StageTeacher` class defining difficulty levels.
  - `train_curriculum.py`: Main training loop interacting with the Teacher.
- `close_generalized/`: Module for the **Generalized Closing** task.
  - `env_gen.py`: Extends the base environment with door pose randomization.
  - `train_gen.py`: Implements a callback that increases difficulty based on Success Rate.

🧠 Implementation Details

1. The Algorithm: Soft Actor-Critic (SAC)

We chose SAC (*Stable Baselines3*) as it is an **Off-Policy** and **Maximum Entropy** algorithm. * **Entropy**: The `ent_coef="auto"` parameter allows the agent to autonomously balance exploration and exploitation, crucial for avoiding local minima in manipulation tasks.

2. Reward Function (Mathematical Logic)

The reward function R_t is dense to act as a proper shaping signal.

Open Task Reward:

$R_{total} = w_{prog} \cdot P + w_{delta} \cdot \Delta + R_{bonus} - R_{costs}$

- **Progress (P)**: Normalized door angle towards the target.
- **Delta (Δ)**: Current velocity of the door opening.
- **Time Penalty**: Small negative reward per step to encourage speed.
- **Return Stage**: If `enable_return_stage=True`, a reward is given for returning the end-effector to the start position *after* success. This ensures a clean "cycle" and safety.

Close Task Reward:

Similar structure but focuses on the latching mechanism. * Specific penalties for arm velocity are applied when the door is closed to encourage a steady, "frozen" state upon task completion.

3. Generalization Strategies

A. Stage-Based Curriculum (Opening)

Defined in `open_generalized/teacher.py`. We use discrete stages (S0 → S4). * **Variables**: * `goal_angle`: From opening a small fraction (5-20%) in S0, to fully opening (100%) in S4. * `friction` & `damping`: Physics properties of the door hinge vary, creating "heavy" or "light" doors. * **Promotion**: If the *Success Rate* over 200 episodes > **85%**, the agent promotes to the next stage.

B. Adaptive Curriculum (Closing)

Defined in `close_generalized/env_gen.py`. * Uses a continuous `curriculum_level` $\alpha \in [0, 1]$. * **Randomization**: Door position and yaw are perturbed. * $\text{Noise}_{pos} \sim U(-0.15\alpha, 0.15\alpha)$ * $\text{Noise}_{rot} \sim U(-0.30\alpha, 0.30\alpha)$ * **Logic**: Starts deterministic. As the agent improves, the environment becomes more chaotic, forcing the policy to generalize to spatial variations.

4. Observation & Action Space

- **Action Space**: Normalized actions [-1, 1] controlling the robot arm (joints/OSC) and gripper.
- **Observation Space**: Proprioceptive and Object state vectors concatenated:
  - `robot0_eef_pos`: End-effector XYZ coordinate.
  - `robot0_eef_quat`: End-effector orientation (quaternion).
  - `robot0_gripper_qpos`: Gripper finger position.
  - `door_angle`: Hinge angle of the door.
  - *(Generalized)* Task info such as `goal_norm` or `stage_idx`.

5. Key Hyperparameters

Configured in `TrainConfig` to ensure stability:

Parameter	Value	Description
Learning Rate	`3e-4`	Standard Adam optimizer rate.
Batch Size	`256`	Number of samples per gradient update.
Buffer Size	`1,000,000`	Replay Buffer capacity.

Parameter	Value	Description
Gamma (γ)	`0.99`	Discount factor for future rewards.
Tau (τ)	`0.005`	Soft update coefficient.
Learning Starts	`10,000`	Random steps before training.
Network Arch	`[256, 256]`	Two hidden layers for Actor/Critic.

🛠️ Usage

Prerequisites

- Python 3.10+
- Virtual Environment

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

Quick Commands (Makefile)

Action	Command	Description
Train (Gen)	`make train_open_gen`	Start Opening training with Stage Curriculum
Train (Base)	`make train_open`	Start Standard Opening training
Play	`make play_open_gen`	Visualize trained model (Opening)
Close	`make train_close_gen`	Start Closing training with Randomization
Monitor	`make board`	Launch TensorBoard