Six Bricks Unified: LLM-Driven Robotic Stacking

Overview

The project focuses on a precise manipulation task: stacking bricks into given wall structure using a Kuka IIWA robot arm. It leverages LLMs to generate context-aware plans for different phases of the manipulation process, ensuring adaptability and robustness.

Key Features

- **LLM-Driven Planning**: Utilizes LLMs (e.g., GPT-4omini) to plan robotic actions.
    - **Multi-Agent Mode**: Specialized agents for each phase (Pre-Grasp, Descend, Close, Lift, Place, Release).
    - **Single-Agent Mode**: A unified agent planning the entire sequence (for comparison).
- **Robust Control System**:
    - **Force Feedback**: Adaptive release strategies based on contact forces.
    - **Angle Optimization**: Automatic correction of brick orientation during placement.
    - **State Verification**: Real-time verification of execution status at each step.
- **Simulation Environment**:
    - Built on **PyBullet**.
    - Simulates a Kuka IIWA robot with a parallel gripper.
    - Dynamic scene generation and physics simulation.

Installation

1. Clone the repository.
2. Install the required dependencies:

```
pip install -r requirements.txt
```

Usage

Running the Demo

To run the main demonstration of the six-bricks stacking task:

```
python run_six_bricks_demo.py
```

This will launch the PyBullet GUI and execute the stacking task set in 'configs\kuka_six_bricks.yaml'.

To set the

Configuration

The main configuration file is located at `configs/kuka_six_bricks.yaml`. You can adjust parameters such as:

- **LLM Settings**: Model, API key, mode (multi_agent/single_agent).
- **Robot & Scene**: Friction, gravity, brick size/mass.
- **Control**: PID gains, thresholds.
- **Goal Layout**: Define custom stacking patterns.

Project Structure

- `run_six_bricks_demo.py`: Main entry point for the demo.
- `parallel_evaluation.py`: Tool for batch evaluation.
- `configs/`: Configuration files (YAML).
- `control/`: Low-level controllers (IK, Gripper, Force Feedback).
- `env/`: PyBullet environment and robot modeling.
- `llm_prompt/`: Prompt templates for different manipulation phases.
- `modules/`: Core logic modules.
  - `motion_executor.py`: Main motion execution state machine.
  - `motion_visualizer.py`: Visualization and debugging helpers.
  - `motion_llm.py`: LLM interaction handler for motion planning.
  - `grasp_module.py`, `state_verifier.py`, `llm_planner.py`: Other core modules.
- `planning/`: Geometric planning algorithms.
- `math3d/`: 3D transformation utilities.