Implementation of DIAL-MPC for a Robotic Arm Lift Task in ManiSkill3

April 2, 2025

Abstract

This document presents an outline for adapting and implementing DIAL-MPC (Diffusion-Inspired Annealing for Model Predictive Control) to solve a robotic arm lift task in the ManiSkill3 environment. The approach leverages GPU-based parallel rollouts and a diffusion-style annealing framework to perform online trajectory optimization without requiring offline training.

Contents

1 Introduction

DIAL-MPC [1] is a sampling-based Model Predictive Control framework that combines large-scale parallel rollouts with a diffusion-inspired dual-loop annealing strategy. Originally developed for legged robot control, we aim to adapt it here for a robotic manipulator lift task in ManiSkill3.

- No offline training needed, broad global search plus local convergence, real-time optimization at each control timestep.
- Handling high-dimensional manipulator state, contact forces, environment constraints, and ensuring stable real-time performance.

2 Environment Setup and Parallel Simulation

2.1 ManiSkill3 Environment Configuration

- Select a Lift scenario in ManiSkill3 (environment ID: PickCube-v1) that involves grasping and lifting an object with a Panda robot arm.
- Ensure that relevant packages (SAPIEN, GPU drivers) are correctly installed, and GPU-based parallel simulation is enabled.
- Verify that each environment provides consistent observations (joint states, object position, etc.).

2.2 Creating Parallel Environments

- Instantiate N parallel instances of the lift environment using ManiSkill3's vector interfaces or custom multi-process solutions. (My GPU is 3060ti, the parallel environment should be small).
- Use identical initial states if needed, or randomize according to the task's setup.
- Each environment must support rollouts up to the planning horizon H in parallel.

3 Cost Function Design and Reward Integration

3.1 Analyzing Built-in Rewards

- ManiSkill3 `Lift` tasks often provide multi-stage rewards (grasp success, stable lift, final height).
- Identify each reward term to decide how they translate into cost terms.

3.2 Converting Rewards to Costs

- DIAL-MPC expects a cost to minimize, but environment typically gives reward.
- Define cost as cost = $-$(reward); optionally re-scale or offset if needed.

3.3 Accumulating Cost

- For each candidate action sequence, sum the per-step cost:

$$J(u_0, \ldots, u_H) \;=\; \sum_{t=0}^{H} \Big[- r_t \Big].$$

- If needed, add extra terms (e.g., torque limit penalties, collision cost).

4 Core Implementation of DIAL-MPC

4.1 Initialization

- Define a horizon H, an initial guess $U^{(0)} = \{u_0, \ldots, u_H\}$, and initial covariance $\Sigma_h^{(0)}$ for sampling each time step h.
- Copy the current state from the real or main environment into the parallel environments.

4.2 Outer Loop: Diffusion-Inspired Annealing

- Perform N_{outer} iterations of the annealing process.
- Gradually decrease covariance from large (global exploration) to small (local refinement).
- Example: exponential schedule for $\det(\Sigma^i)$.

4.3 Inner Loop: Action-Level Annealing

- Different time steps in the horizon can have different noise scales.
- Let near-term actions be more precise (small noise), while far-future actions keep more exploration (larger noise).

4.4 Parallel Rollouts and Weighted Update

- Sample N_{samples} sequences around the current solution; evaluate cost by rolling out environment up to horizon H in parallel.
- Use an exponential weighting scheme:

$$w_j \;=\; \exp\Big(- \frac{J_j}{\lambda} \Big),$$

then update U^+ via the weighted average of perturbations.

5 Control Loop and Execution

5.1 Action Execution

- Once the solution is found, apply the first action u_0 to the real manipulator.
- Maintain the rest of the sequence for the next time step's warm-start.

5.2 Sliding Window Mechanism

- Shift $U \leftarrow (u_1, u_2, \ldots, u_H, \text{new guess})$.
- This ensures continuity of control actions from one MPC iteration to the next.

5.3 State Synchronization

- After applying u_0 in the real environment, measure new state (joint angles, object pose, etc.).
- Reset parallel environments to that state for the next iteration's rollouts.

6 Parameter Settings and Practical Tips

6.1 Control Frequency and Horizon

- A 10–20 Hz MPC loop is common for manipulator tasks.
- Horizon $H \approx$ 10–20 steps (0.5–1.0 s lookahead).

6.2 Parallel Sampling Quantity

- The number of samples N_{samples} per iteration depends on GPU capacity.
- Typical range: 256–2048.

6.3 Annealing Schedule and Temperature Parameter

- Use exponential or piecewise-linear scheduling for Σ.
- Adjust λ (temperature) to match cost magnitudes and avoid numerical instability.

6.4 Cost Function Tuning

- Add minor penalties for large joint velocities or excessive torque to ensure stable motion.
- If the arm exhibits jitter, increase the final annealing stage length or reduce noise.

7 Testing and Debugging

7.1 Offline Simulation Testing

- Validate that the arm can indeed lift the object in a pure simulation environment.
- Monitor the cost convergence and final success rate.

7.2 Parameter Tuning

- Adjust $N_{\text{outer}}, N_{\text{samples}}, \Sigma$ schedules, and λ iteratively.
- Too large a noise leads to divergence; too small a noise leads to local minima.

7.3 Benchmark Comparison

- Compare with SAC algorithms in terms of success rate and policy generalization.
- Evaluate energy consumption or torque usage as additional metrics.

7.4 Real-World Deployment

- Account for latency, sensor noise, domain gap between real manipulator and simulation.
- Possibly learn an online correction model for friction or contact differences.

References

[1] H. Xue et al., *Diffusion-Inspired Annealing for Legged MPC (DIAL-MPC)*, arXiv:2409.15610, 2024.