[NUS'24] Differentiable Particles for General-Purpose Deformable Object Manipulation

1. Link: https://arxiv.org/pdf/2405.01044
2. Arthurs and institution: Siwei Chen, Yiqing Xu, Cunjun Yu, Linfeng Li and David Hsu from NUS **TL;DR** DiPac represents a deformable object as a set of particles and uses a differentiable particle dynamics simulator to reason about robot manipulation. It combines learning, planning, and trajectory optimization through differentiable trajectory tree optimization to find the best manipulation action.

comments and critisims

1. can other params, such as the number of particles, mass be optimized
2. how does a MLS-MPM system measure rigidity
3. why use two simulators
4. the policy seems could be replaced by a diffusion policy

Related Works

1. Model-based DOM (deformable object manipulation)
    1. manually design the model
    2. learn from data
    3. online adaptation in real-world
2. Data-driven DOM
    1. LfD
    2. struggle to generalization in unseen scenarios
3. Differentiable Dynamics
    1. serve as a dynamic model
    2. control problem
    3. sim-real gap in dynamics
        1. learn model param from real-world data

Deformable Object Manipulation With Differentiable Particles

Definitions

1. differentiable simulator serves as $x_{t+1} = f(x_t, \mu_t; \varphi)$, where x is state of object, u is robot action and φ is dynamic parameters
2. objective function of the problem

$$J(x_0, U_{0:T-1}) = \sum_{t=0}^{T-1} c(x_t, u_t).$$

$$c(x_t, u) = \sigma(x_t, x^{\mathrm{g}}) - \sigma(x_{t-1}, x^{\mathrm{g}}) + \zeta(x_t, u_t),$$

   where σ is chamfer distance and ζ is a contact cost of l2 distance btw gripper and object.

Inputs and Outputs

1. inputs RGBD images
2. outputs manipulation actions

Particle State Construction

1. assumptions
   1. no occlusion
   2. good segmention is obtainable
2. steps
   1. get maske point clouds from each view
   2. merging them together
   3. unobserved interior is then filled with uniformly sampled particles

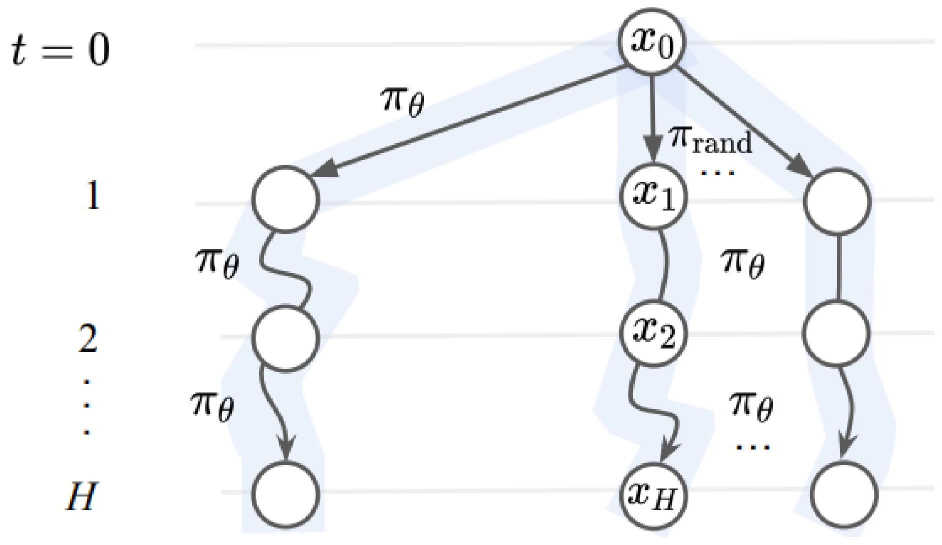
Dynamics Model Calibration

1. parameter involved
   1. stiffness
   2. friction coefficient
2. purpose
   1. for better action selection
3. steps

1. collection expert demostration data from real-world
2. update the params though gradient descent from simulator

$$\varphi = \arg\min_{\varphi} \sum_{t=0}^{T-1} \sigma(f(x_t, u_t; \varphi), x_{t+1}),$$

Integrating Learning, Planning, and Optimization



1. Objective function

$$U^*_{0:H-1} = \arg\min_{U_{0:H-1}} \; J(x_0, U_{0:H-1}) = \sum_{t=0}^{H-1} c(x_t, u_t),$$

2. Steps
    1. use an initial policy to roll out actions
    2. random sample particle with angle and position as explorations, with an action from initial policy
    3. use gradient descent to optimize the actions over each trajectory
    4. select and excute the action with minimal cost
3. benefits
    1. good inital policy help to find a solution more efficiently
        1. initialize us with a near-optimal trajectory
        2. enable us to optimize the gradients on a good optimization landscape

    2. balances sample complexity, computation speed, and final performance

Details

1. hardware
    1. kinova gen3 robot
2. simulation
    1. PlasticineLab for rope pushing and bean sweeping, Plastic
    2. DaX for Cloth Hanging, Water Pouring and Soup
3. param

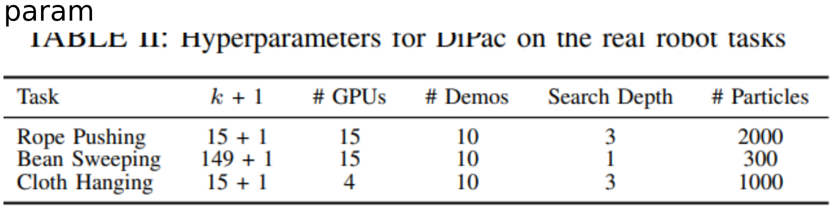


TABLE II: Hyperparameters for DiPac on the real robot tasks

Task	k + 1	# GPUs	# Demos	Search Depth	# Particles
Rope Pushing	15 + 1	15	10	3	2000
Bean Sweeping	149 + 1	15	10	1	300
Cloth Hanging	15 + 1	4	10	3	1000

4. computation time
    1. take about 20 - 40 seconds for model param learning
    2. real world task

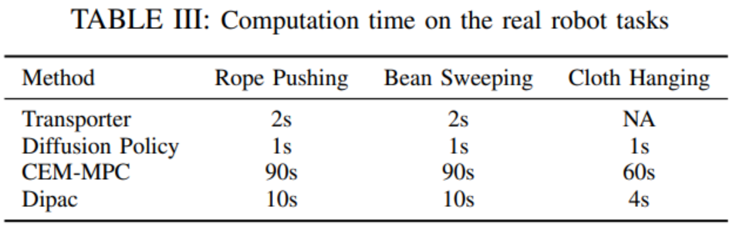


TABLE III: Computation time on the real robot tasks

Method	Rope Pushing	Bean Sweeping	Cloth Hanging
Transporter	2s	2s	NA
Diffusion Policy	1s	1s	1s
CEM-MPC	90s	90s	60s
Dipac	10s	10s	4s

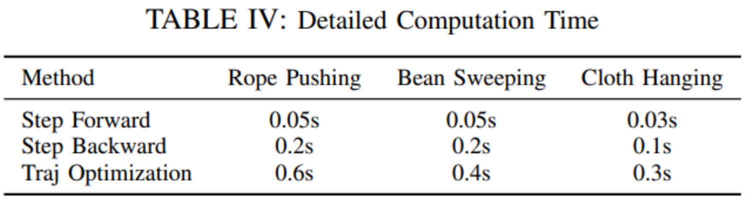


TABLE IV: Detailed Computation Time

Method	Rope Pushing	Bean Sweeping	Cloth Hanging
Step Forward	0.05s	0.05s	0.03s
Step Backward	0.2s	0.2s	0.1s
Traj Optimization	0.6s	0.4s	0.3s

5. MLS-MPM

1. Particle to Grid

Particle to Grid (P2G). In the process of P2G, the MLS-MPM first updates its deformation gradient $\mathbf{F}^{n+1}$ given the previous $\mathbf{F}^n$ and velocity gradient $\mathbf{C}^n$. It then computes the affine velocity $\mathbf{A}^n$ to update grid velocity $\mathbf{v}$ as shown below:

$$\begin{aligned}
\mathbf{F}_p^{n+1} &= \left(\mathbf{I} + \Delta t \mathbf{C}_p^n\right) \mathbf{F}_p^n \\
\mathbf{A}_p^n &= m_p \mathbf{C}_p^n - \frac{4\Delta t}{\Delta x^2} \sum_p V_p^0 \mathbf{P}\left(\mathbf{F}_p^{n+1}\right)\left(\mathbf{F}_p^{n+1}\right)^T \\
(m\mathbf{v})_i^{n+1} &= \sum_p w_{ip} \left\{ m_p \mathbf{v}_p^n + \mathbf{A}_p^n \left(\mathbf{x}_i - \mathbf{x}_p^n\right)\right\}.
\end{aligned}$$

We denote i as the grid index, p as the particle index, and n as the number of updates. The $\mathbf{P}$ involves object properties such as rigidity to affect the deformation process. MLS-MPM aggregates the velocities of all the particles into a grid, which enables modeling the interactions among all particles without the need to search for neighbors of each individual particle.

2. Grid Operation

the need to search for neighbors of each individual particle.

Grid Operation (GO). The MLS-MPM then normalizes the grid velocity and computes boundary conditions (BC) as below:

$$\begin{aligned}
\hat{\mathbf{v}}_i^{n+1} &= (m\mathbf{v})_i^{n+1} / m_i^{n+1} \\
\mathbf{v}_i^{n+1} &= \mathrm{BC}\left(\hat{\mathbf{v}}_i^{n+1}\right).
\end{aligned}$$

3. Grid to Particle

$$\begin{aligned}
\mathbf{v}_p^{n+1} &= \sum_i w_{ip} \mathbf{v}_i^{n+1} \\
\mathbf{C}_p^{n+1} &= \frac{4}{\Delta x^2} \sum_i w_{ip} \mathbf{v}_i^{n+1} \left(\mathbf{x}_i - \mathbf{x}_p^n\right)^T \\
\mathbf{x}_p^{n+1} &= \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1}.
\end{aligned}$$