

A REVERSE SIMULATION VIA PHYSICS-CONSTRAINED OPTIMIZATION

To derive the reverse simulator, we need to first understand the mechanism of the forward simulation. The forward simulation is formulated as the following unconstrained optimization:

$$\arg \min_{p_{t+1}} \Psi(p_{t+1}, p_t, p_{t-1}, c_t), \quad (5)$$

where we denote by p_t the kinematic state at the t th time instance, and c_t is the control input at the t th time instance. In our setting of 2D rigid bodies, p_t is the concatenation of p_t^i with p_t^i being the kinematic state of the i th rigid body, i.e. $p_t^i = (x_t^i, y_t^i, \theta_t^i)^T$ and the complete state s_t used in DRL is a concatenation of kinematic state and velocity, i.e. $s_t = (p_t, p_{t-1})$, where velocity can be recovered from finite difference. The function Λ in our main paper is defined as $\Lambda(s_{t+1}, s_t, c_t, q(a_t)) = \nabla_{p_{t+1}} \Psi(p_{t+1}, p_t, p_{t-1}, c_t)$. In Section A.1 we would introduce the formulation of the objective function Ψ . For now, we assume that Ψ is twice-differentiable in all the parameters. In the reverse simulator, we consider an entire trajectory of h timesteps, denoted as $p = (p_{t-h}^T, \dots, p_{t-1}^T)^T$, with the associated control inputs $c = (c_{t-h+1}^T, \dots, c_t^T)^T$, where we further denote by p (resp. c) (without subscript) the concatenation of p_t (resp. c_t) over all the time indices. Here, we assume $s_{t+1} = (p_{t+1}, p_t)$ is known as fixed. We would like to optimize the sequence of control inputs to optimize the following objective function:

$$J(p, c) = \sum_{i=1}^M (x_{t-h}^i, y_{t-h}^i) d(a_t) + \lambda \sum_{k=0}^{h-1} \|c_{t-k}\|^2 + P_{\perp}(p_{t-h}), \quad (6)$$

where our primary goal is to move all M rigid bodies along the negative pushing direction $d(a_t)$ as far as possible, while fixing the final state. Note that our objective encourages the robot to push all M rigid bodies, since the set of rigid bodies to be pushed simultaneously is unknown to us a priori. Note that when the robot cannot reach certain rigid bodies, these bodies will not move due to our physics constraints, despite our objective function encourages the bodies to be moved. Further, we also add a small control regularization with a small coefficient λ . Finally, we introduce a regularization energy $P_{\perp}(p_{t-h})$ to ensure the initial state satisfies the collision-free constraints, which is defined in Section A.1. During our optimization, we need to always ensure that Equation 5 is satisfied, which guarantees physical correctness. Combining Equation 5 and Equation 6, we propose to solve the following constrained optimization:

$$\arg \min_{p, c} J(p, c) \quad \text{s.t.} \nabla_{p_{t-k+1}} \Psi(p_{t-k+1}, p_{t-k}, p_{t-k-1}, c_{t-k}) = 0 \quad \forall k = 0, \dots, h-1. \quad (7)$$

Under the assumption that the function Ψ is twice-differentiable and thus the function Λ is differentiable, we can efficiently solve Equation 7 using SQP.

A.1 OPTIMIZATION-BASED 2D RIGID BODY SIMULATOR

In this section, we consider the dynamics of multiple 2D rigid bodies, for which we derive the concrete form of the objective Ψ and its derivatives. Our starting point is the 2D version of the dynamic simulator (Huang et al., 2024). The energy Ψ consists of five terms: the inertia term $I(p_{t+1}, p_t, p_{t-1})$ and damping term $I_D(p_{t+1}, p_t)$, the normal collision potential $P_{\perp}(p_{t+1})$, the frictional collision potential $P_{\parallel}(p_{t+1}, p_t)$, and finally the external force potential $P_E(p_{t+1}, c_t)$. Specifically, we have:

$$\Psi(p_{t+1}, p_t, p_{t-1}, c_t) = I(p_{t+1}, p_t, p_{t-1}) + I_D(p_{t+1}, p_t) + P_{\perp}(p_{t+1}) + P_{\parallel}(p_{t+1}, p_t) + P_E(p_{t+1}, c_t).$$

We present the concrete formula for each and every term above.

Inertia & Damping Term: In the original formula for the dynamic simulator (Huang et al., 2024), the inertial term is designed for soft bodies instead of rigid bodies. Instead, we follow Pan & Manocha (2018) to formulate the rigid body inertia term as follows:

$$I(p_{t+1}, p_t, p_{t-1}) = \sum_{j=1}^M \int_{\Omega_j} \frac{\rho}{2\Delta t^2} \|X(x, p_{t+1}^j) - 2X(x, p_t^j) + X(x, p_{t-1}^j)\|^2 dx, \quad (8)$$

where ρ is the rigid body density, Δt is the timestep size, and $\Omega_j \subset \mathbb{R}^2$ is the volume taken by the j th rigid body. Finally, $X(x, p_t^j)$ is the world-space position of x under configuration p_t^j , defined as:

$$X(x, p_t^j) = \begin{pmatrix} \cos(\theta_t^j) & -\sin(\theta_t^j) \\ \sin(\theta_t^j) & \cos(\theta_t^j) \end{pmatrix} x + \begin{pmatrix} x_t^j \\ y_t^j \end{pmatrix}.$$

The dynamic simulator discretizes the acceleration by finite difference over three time instances. Following the similar logic to the inertial term, we can define the following damping term that penalizes the velocity at every timestep:

$$I_D(p_{t+1}, p_t) = k_D \sum_{j=1}^M \int_{\Omega_j} \frac{\rho}{2\Delta t^2} \|X(x, p_{t+1}^j) - X(x, p_t^j)\|^2 dx,$$

with k_D being the damping coefficient. We refer readers to [Pan & Manocha \(2018\)](#) for the computational evaluation of these terms.

Normal Collision Potential: Without a loss of generality, we can assume the i th rigid body has the geometry of a convex polyhedron with K_i vertices denoted as $(v_{i,1}, \dots, v_{i,K_i})$. We now define the smoothed signed distance function of a point p to the i th rigid body to be $d_i(p)$. We follow the method of incremental potential contact used by [Huang et al. \(2024\)](#) and define:

$$P_{\perp}(p_{t+1}) = -\nu \sum_{j=1}^M \sum_{i \neq j} \sum_{k=1}^{K_i} \log(d_j([v_{i,k}^j]_{t+1})),$$

where we choose ν as a small positive coefficient and $X^{-1}(\bullet, p_j^{t+1})$ is the inverse function of $X(\bullet, p_j^{t+1})$. Here we define $[v_{i,k}^j]_{t+1} = X^{-1}(X(v_{i,k}, p_{t+1}^j), p_j^{t+1})$, with is the position of $v_{i,k}$ in j th object's local frame of reference at time instance $t+1$. In other words, P_{\perp} requires that every vertex of a rigid body to be non-penetrating with other rigid bodies.

Frictional Collision Potential: The frictional potential is formulated in a similar manner following the idea of incremental potential contact used by [Huang et al. \(2024\)](#). We first compute each contact force between $v_{i,k}$ and the j th rigid body from the last timestep, which is:

$$f_{\perp,j,i,k} = \nu \left\| \frac{\partial \log(d_j([v_{i,k}^j]_t))}{\partial [v_{i,k}^j]_t} \right\|.$$

We then formulate the frictional damping term as:

$$f_{\parallel,j,i,k} = \beta f_{\perp,j,i,k} \left\| \text{Proj}_{\parallel} \left[\frac{X(v_{i,k}, p_{t+1}^j) - X(v_{i,k}, p_t^j)}{\Delta t} - \frac{X([v_{i,k}^j]_t, p_{t+1}^j) - X([v_{i,k}^j]_t, p_t^j)}{\Delta t} \right] \right\|,$$

where Proj_{\parallel} is the projection to the tangential plane. Finally, we define:

$$P_{\parallel}(p_{t+1}, p_t) = \nu \sum_{j=1}^M \sum_{i \neq j} \sum_{k=1}^{K_i} f_{\parallel,j,i,k},$$

with β being the frictional coefficient. Intuitively, we damp the relative tangential velocity between $v_{i,k}$ on the i th object and $[v_{i,k}^j]_t$ on the j th object in contact.

External Force Term: We control the dynamic system using external force and torque. Without the loss of generality, we can assume the first rigid body is the robot end-effector, which can be controlled by $c_t = (f_t^x, f_t^y, \tau_t)$ with (f_t^x, f_t^y) being the external force and τ_t being the external torque. Then the external force term is $-f_t^x x_t^1 - f_t^y y_t^1 - \tau_t \theta_t^1$. However, the above formula might introduce excessively large forces, which is unrealistic. We can regularize the situation by introduce a bound B_f on the force magnitude and enforcing $-B_f \leq f_t^{x,y} \leq B_f$. Similarly, we introduce a bound B_{τ} on the torque magnitude and enforce $-B_{\tau} \leq \tau_t \leq B_{\tau}$. Such constraint can be achieved by using the tanh activation function and defining:

$$P_E(p_{t+1}, u_t) = -B_f \tanh(f_t^x) x_t^1 - B_f \tanh(f_t^y) y_t^1 - B_{\tau} \tanh(\tau_t) \theta_t^1.$$

A.2 SEQUENTIAL QUADRATIC PROGRAMMING

We provide the complete detail of our SQP algorithm. We first define the constraint vector $C(p, c) = (\Lambda_{t-h+1}, \dots, \Lambda_t)$, where we abuse notation and write $\Lambda_{t-k} = \Lambda(s_{t-k+1}, s_{t-k}, c_{t-k})$. We adopt the variant of SQP guided by the following l_1 -merit function [\(Boggs & Tolle, 1995\)](#): $\Theta(p, c, \eta) = J(p, c) + \eta \|C\|_1$. We start from the initial guess $p^0 = (p_{t+1}, \dots, p_{t+1})$, $c^0 = (0, \dots, 0)$,

i.e., we initialize the trajectory to be static at state p_{t+1} with all zero control forces and torques. SQP iteratively updates the solution p, c to reduce the merit function until a critical point is achieved. To update a solution p, c , we solve the following quadratic programming by using quadratic approximation of the objective function and linear approximation of all the constraints:

$$\begin{aligned} \arg \min_{\Delta p, \Delta c} \quad & \frac{\partial J^T}{\partial p} \Delta p + \frac{\partial J^T}{\partial c} \Delta c + \frac{1}{2} \Delta p^T \frac{\partial^2 J}{\partial p^2} \Delta p + \frac{1}{2} \Delta c^T \frac{\partial^2 J}{\partial c^2} \Delta c \\ \text{s.t.} \quad & C + \frac{\partial C}{\partial p} \Delta p + \frac{\partial C}{\partial c} \Delta c = 0. \end{aligned} \quad (9)$$

The above Quadratic Program (QP) has all-linear equality constraints with a quadratic objective. This is because we use the tanh soft activation function to model control force and torque limits in the external force term P_E from Section A.1, which transforms the inequality control limits into the equality constraints after linearization. This is key to the fast solution of trajectory optimization, since the QP sub-problem can be efficiently solved via the following KKT linear system:

$$\begin{pmatrix} \frac{\partial^2 J}{\partial p^2} & \frac{\partial C^T}{\partial p} \\ \frac{\partial^2 J}{\partial c^2} & \frac{\partial C^T}{\partial c} \\ \frac{\partial C}{\partial p} & \frac{\partial C}{\partial c} \end{pmatrix} \begin{pmatrix} \Delta p \\ \Delta c \\ \lambda \end{pmatrix} = \begin{pmatrix} -\frac{\partial J}{\partial p} \\ -\frac{\partial J}{\partial c} \\ -C \end{pmatrix}, \quad (10)$$

with λ being the associated Lagrangian multiplier. Note that by definition, the mixed derivatives $\frac{\partial^2 J}{\partial p^2} c$. For now, we assume the KKT-system can be readily solved, then SQP proceeds by choosing a step size η^j such that:

$$\Theta(p + \eta^j \Delta p, c + \eta^j \Delta c, \eta) < \Theta(p, c, \eta) + \eta^j D\Theta(p, c, \Delta p, \Delta c, \eta), \quad (11)$$

where $D\Theta$ is the directional derivative of Θ along Δp and Δc . To ensure that such η^j exists, we need to choose $\eta > \|\lambda\|_\infty$. We notice that SQP is an infeasible solver that is not guaranteed to return a feasible solution. Specifically, a feasible solution is only returned when the constraint qualifications are satisfied. In practice, we find the constraint qualifications can be violated when the physics constraints are violated. To improve the success rate of SQP, we follow Solodov (2009) to use a feasibility safe-guard. Specifically, instead of using Equation 11 as the only condition of line-search, we add a condition to ensure that $\|C(p + \delta p, c + \Delta c)\|_1 \leq \epsilon_c$. We find that by using a sufficiently small ϵ_c , the SQP solver becomes much more robust and we never observe failure cases in our experiments.

A.3 FAST KKT SYSTEM SOLVE

Directly solving Equation 10 without exploiting the sparsity pattern can take $O(h^3)$. Instead, we show that by utilizing the sparsity pattern, we can solve the KKT-system at a cost of $O(h)$. To see this, we write the Lagrangian multiplier $\lambda = (\lambda_{t-h+1}, \dots, \lambda_t)$. The fast linear system solve can be derived by a permutation of variables as follows:

$$\nu_{t-k} \triangleq \begin{pmatrix} \Delta p_{t-k-1} \\ \Delta c_{t-k} \\ \lambda_{t-k} \end{pmatrix} \quad \begin{pmatrix} \Delta p \\ \Delta c \\ \lambda \end{pmatrix} = P \begin{pmatrix} \nu_{t-h+1} \\ \vdots \\ \nu_t \end{pmatrix},$$

with P being the permutation matrix. The lefthand side of Equation 10 after symmetric permutation reads:

$$P \begin{pmatrix} \frac{\partial^2 J}{\partial p^2} & \frac{\partial C^T}{\partial p} \\ \frac{\partial^2 J}{\partial c^2} & \frac{\partial C^T}{\partial c} \\ \frac{\partial C}{\partial p} & \frac{\partial C}{\partial c} \end{pmatrix} P^T = \begin{pmatrix} A_{t-h+1} & B_{t-h+2}^T & & \\ B_{t-h+2} & A_{t-h+2} & B_{t-h+3}^T & \\ & B_{t-h+3} & A_{t-h+3} & \ddots \\ & & & B_t^T \\ & & & B_t & A_t \end{pmatrix},$$

which takes a block tridiagonal form. Here we define the blocks as follows:

$$A_{t-k} \triangleq \begin{pmatrix} \frac{\partial^2 J}{\partial \Delta p_{t-k-1}^2} & \frac{\partial \Lambda_{t-k}}{\partial \Delta p_{t-k-1}^T} \\ \frac{\partial^2 J}{\partial \Delta c_{t-k}^2} & \frac{\partial \Lambda_{t-k}}{\partial \Delta c_{t-k}^T} \\ \frac{\partial \Lambda_{t-k}}{\partial \Delta p_{t-k-1}} & \frac{\partial \Lambda_{t-k}}{\partial \Delta c_{t-k}} \end{pmatrix} \quad B_{t-k} \triangleq \begin{pmatrix} \frac{\partial \Lambda_{t-k-1}}{\partial \Delta p_{t-k-1}^T} \\ \frac{\partial \Lambda_{t-k}}{\partial \Delta p_{t-k-2}} \end{pmatrix}$$

Therefore, we could use the cyclic reduction algorithm (Gander & Golub, 1998) to solve the linear system with a cost of $O(h)$. Put together, each iteration of our SQP involves a single solve of the KKT-system, so the iterative cost is $O(h)$.

B HYPERPARAMETERS

In Table 2, we report the choice of our method’s hyperparameters.

Table 2: Our method’s hyperparameters. These are the ones used to generate our figures and results. Highlighted in blue indicates hyperparameters introduced by this paper.

Hyperparameter	Value
RL Hyperparameters (DQN & Double DQN)	
Discount factor (γ)	0.8 (Gathering)
	0.9 (Sorting and ArtManip.)
Replay Buffer Capacity	1,500,000
Batch Size	512
Total Interactions / Samples	350,000 (Gathering and Sorting)
	400,000 (ArtManip.)
Networks and Optimization	
Network Shape of Features Extractor (MLP)	[512, 512, 128]
Learning Rate	5e-5
Gradient Steps	1
Train Frequency	4
Network Optimizer	Adam
Environment and Data	
Reward Function	Sparse (+1 on success, 0 otherwise)
Action Repeat	1
Episode Horizon	60
Observation Type	State
RTG	
Push Stride	2.0
Number of discrete actions	48
Offline transitions generated	49144 (Gathering)
	67793 (Sorting)
	89905 (ArtManip.)

C TASK VISUALIZATIONS

In Figure 10, Figure 11, Figure 12, we show visually how each of our proposed task is accomplished.

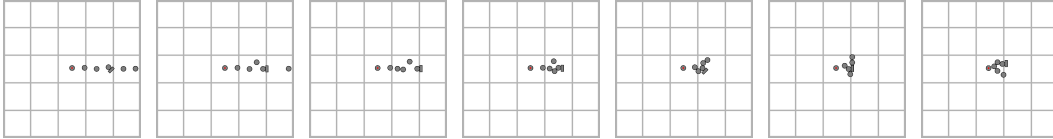


Figure 10: A sample successful trajectory for the task of Gathering.

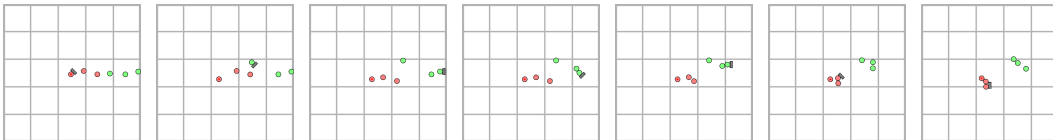


Figure 11: A sample successful trajectory for the task of Sorting.

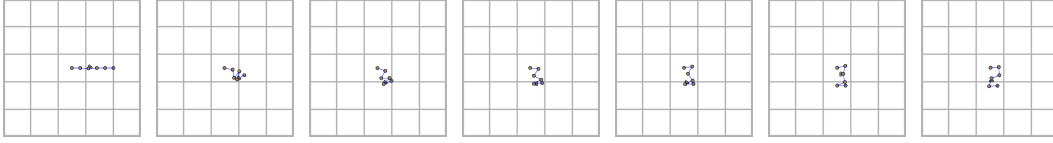
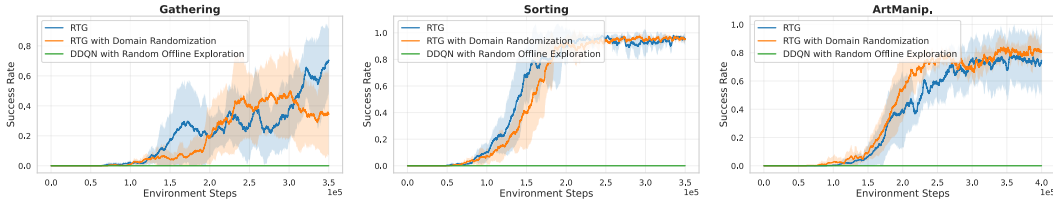
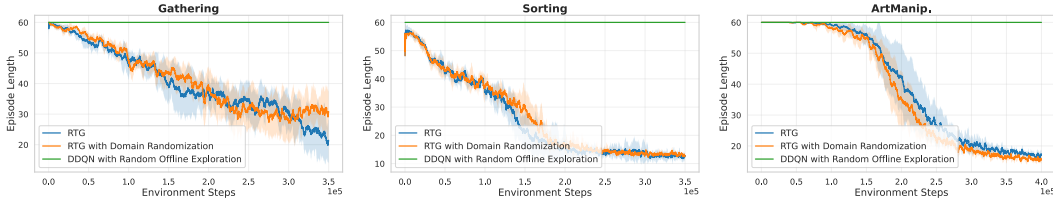
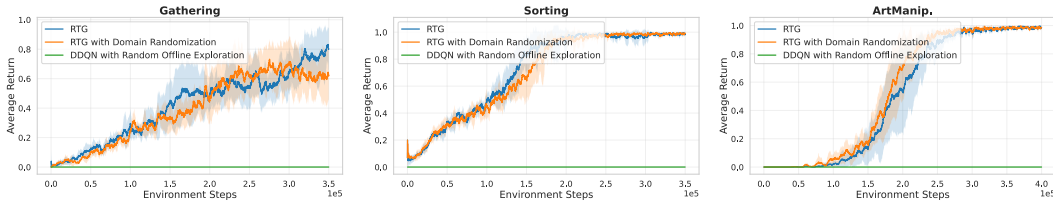


Figure 12: A sample successful trajectory for the task of ArtManip.

D ADDITIONAL RESULTS

Here we report more comparisons with (a) DDQN with Random Offline Exploration and (b) RTG with Domain Randomization, both combined with DDQN. For the former, DDQN with random offline exploration is essentially an ablation to validate the data informativeness of offline transitions from RTG. For the latter, we add domain randomization to the initial state distribution (with each object’s positions perturbed by ± 0.5 , ± 1.0 , ± 0.5 with respect to three tasks) to validate our method RTG’s robustness. The comparisons are shown in Figure 13, Figure 14, Figure 15.

Figure 13: Mean average success rate of algorithms for each task. Results are averaged within each environment. Shaded areas represent ± 1 std. over 5 seeds.Figure 14: Mean episode length of algorithms for each task. Results are averaged within each environment. Shaded areas represent ± 1 std. over 5 seeds.Figure 15: Mean average return of algorithms for each task. Results are averaged within each environment. Shaded areas represent ± 1 std. over 5 seeds.

E RUNTIME ANALYSIS

Runtime of backward and forward simulation. During backward generation, we run our RRBS in quasi-static mode, with maximum solver iteration set to 1000. On an AMD Ryzen 9 5950X CPU (16C/32T, 1 socket, 1 NUMA node), a single parallel backward optimization over 48 candidate trajectories takes 2.45 ± 0.13 s wall-clock time (mean \pm std) for our task of ArtManip (with joints). This corresponds to 51.0 ± 2.6 ms per backward action, where each optimizer solves one per-action

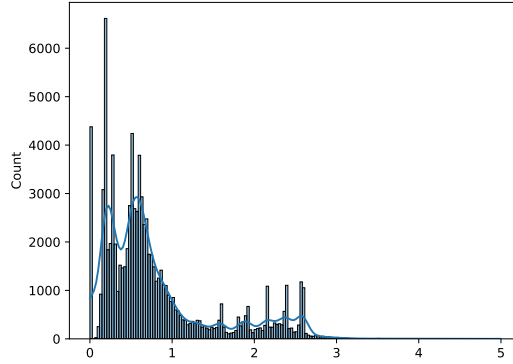


Figure 16: Replay gap distribution between reverse-generated states and the true forward dynamics. The horizontal axis shows the average 2D positional gap per node, measured as the mean Euclidean distance between each node’s position in the backward-optimized state and the corresponding state after Forward Replay. Distances are in simulator units. The distribution is concentrated near zero but exhibits a non-zero tail, indicating a small yet systematic mismatch in reverse physics and thereby justifying our Forward Replay step.

backward simulation as defined in Figure 2(a). For Gathering and Sorting (without joints), the optimization stage over 48 candidates costs 1.90 ± 0.06 s, *i.e.*, 39.6 ± 1.3 ms per action step. For comparison, the forward simulator costs 1.07 ± 0.34 ms per action step, so a single-step backward optimization is approximately one order of magnitude more time-consuming than a forward step.

Time complexity of beam search. For our beam search (Algorithm 1) over backward actions with beam breadth B and horizon depth D , at each search layer we expand at most B nodes, and for each node we run one parallel backward optimization followed by a ranking step. Thus the total number of backward steps scales as $\mathcal{O}(D \times B)$, and the overall time complexity of the beam search is linear in both beam width and horizon depth.

F FORWARD REPLAY GAP ANALYSIS

We quantify the discrepancy between the states generated by our backward simulator and those produced by the true forward dynamics in Figure 16. As shown, the replay gap is small but clearly non-zero, indicating that the reverse physics are not perfectly consistent with the forward dynamics. This systematic mismatch motivates our Forward Replay step, which re-simulates backward-optimized trajectories under forward dynamics before utilizing them.

G USE OF LLMs

We acknowledge the use of large language models (LLMs) as assistive tools in this research. LLMs are used during paper writing, for improving grammar and wording. All outputs from these models were meticulously reviewed, revised, and verified by the authors, who retain full responsibility for all content presented in this paper.