Supplementary Information

A. Discussion of RL Applications

For the policy enhancement method used in n-link pendulum, we use off-policy training and weigh the relevance of each sample according to the distance to the current action:

$$w = \exp(-\|\mathbf{a} - \mathbf{a}_0\|) \tag{16}$$

where $\mathbf{a}$ is the policy network output, and $\mathbf{a}_0$ is the action sampled from the history pool. For every epoch, we sample from the environment 1,000 times and train the model after each sample. We train both models from the simplest 1-link to the most difficult 7-link setting, each experiment repeated 10 times to obtain stable statistics. In Table 4, we report the statistics of the maximum return for pendulums of different numbers of links. As mentioned in the main text, the performance of MBPO degrades while ours does not. The reason behind it is that, during the policy update, our method does not need to fully depend on the critic function. By making use of the gradients as shown in Eq. 14, the policy network can first optimize the short-term reward ($\partial r/\partial a$) before the critic is up to date. When the critic network is able to reveal the true value of the state-action pair, the policy can then converge as well to the long term goal. On the other hand, traditional RL methods heavily depend on the convergence of the critic network, thus having a hard time searching for the best action in a large space, especially when the links are attached sequentially.

In Fig. 5 that shows our MuJoCo Ant experiment results, we observe an even faster convergence than MBPO, while MBPO already outperforms most of the RL algorithms. This is because by generating nearby samples, the feedback from the environment is no longer a single point but a patch as an approximation of the shape of the state transitioning function. It can greatly help the critic function learn the correct estimation of the values not only for the observation point but also nearby ones, thereby achieving faster convergence on the value estimation and eventually the policy model.

Limitations and future directions. We also observe limitations on these two proposed techniques. We observed that the sample enhancement does not provide better overall rewards in the 'pendulum' test, and the policy enhancement does not provide faster convergence in the 'ant' test. We attribute this to a number of possible causes. Sample enhancement enriches the interaction history with the environment, but it is inherently limited because extra samples can only be generated near the true sample, it may not help the agent jump out of the local minima. Policy enhancement relies heavily on the assumption that the actions drawn from the training batch are the same as or similar to the action that the policy takes at the current moment: it is considered as an on-policy learning algorithm. Therefore, it faces the same limitation, which is poor sample efficiency, as other on-policy algorithms as well. These two techniques cannot be combined together because the generated samples from the sample enhancement do not have accurate first-order gradients at their location, as mentioned in the paper.

While we have shown the two techniques to be a proof-of-concept demonstration that differentiable physics can help improve performance of reinforcement learning using an appropriate method depending on different scenarios, there remain several directions to further explore. We hope that our preliminary investigation on differentiable physics integrated with RL, as presented in this paper, will stimulate more study on the coupling of differentiable physics with reinforcement learning.

B. Ablation Study for Checkpointing Scheme

In this section, we conduct an ablation study to show why we choose to save the checkpoints each time step, instead of having a lower frequency for checkpointing. We vary the intervals between checkpoints and profile the peak memory usage and backpropagation time per step. Our default design is interval=1 such that checkpoints are stored every step.

If interval=k, our method needs to compute the gradients $\overline{x_i}, \overline{x_{i+1}}, .., \overline{x_{i+k-1}}$ given the gradients $\overline{x_{i+k}}$ and the checkpoint x_i. The intermediate variables in step $i+k-1, i+k-2, ..., i$ have to be resumed from x_i, taking $k, i-1, ..., 1$ steps of forward simulations, respectively. Therefore, the total forward steps used to resume variables here are $(k+1)k/2$. If there are n simulation steps in total, the average time of backpropagation per step should be

$$(k+1)/2 \cdot t_{forawrd} + t_{backward}, \tag{17}$$

where $t_{simulation}, t_{adjoint}$ are the time for one step of simulation and adjoint method, respectively. On the other hand, the peak memory usage is

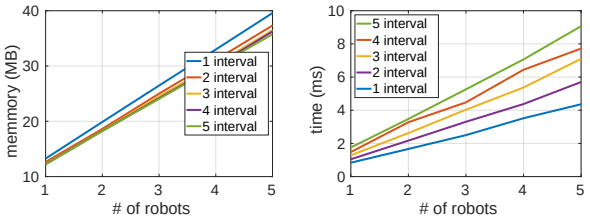
$$n/k \cdot m_{checkpoint} + m_{simulation}, \tag{18}$$

where $m_{checkpoint}$ is the size of one checkpoint; $m_{simulation}$ is the memory usage of one step, which is the size of all intermediate variables. Since $m_{simulation} \gg m_{checkpoint}$, the increase of k would not significantly reduce the memory usage but will slow down the backpropagation linearly.

In Figure 8, the experiments have similar setup as in Section 6.1 where we simulate 10 Laikago robots for 5,000 steps. The memory and the time scale linearly w.r.t. the number of the robots. When the checkpoint intervals get larger, the memory usage decreases gradually to the lower bound of $m_{simulation}$, but the average backpropagation time increases linearly. Since the memory usage is low enough even for interval $k = 1$, we choose to checkpoint every step to have a faster speed and set the interval $k = 1$.

# of links	1	2	3	4	5	6	7
Ours	**0.49± 0.01**	**1.93± 0.08**	**4.28± 0.25**	**7.81± 0.19**	**12.31± 0.16**	**17.78± 0.24**	**24.34± 0.11**
MBPO	**0.49± 0.01**	1.90± 0.02	**4.28± 0.22**	7.50± 0.57	11.13± 0.99	15.02± 1.45	19.97± 2.33
Maximum	0.5	2	4.5	8	12.5	18	24.5

Table 4. Maximum reward on n-link pendulum. Our method has higher reward scores than MBPO, as the system complexity increases.



(a) Memory (b) Time

Figure 8. Ablation study for the checkpointing scheme.

C. The Adjoints of Contact Solver

In the collision solver, we construct a Mixed Linear Complementarity Problem (MLCP) (Stepien, 2013):

$$\begin{aligned} \mathbf{a} &= \mathbf{A}\mathbf{x} + \mathbf{b} \\ \text{s.t.} \quad \mathbf{0} \leq \mathbf{a} &\perp \mathbf{x} \geq \mathbf{0} \quad \text{and} \quad \mathbf{c}_- \leq \mathbf{x} \leq \mathbf{c}_+, \end{aligned} \tag{19}$$

where $\mathbf{x}$ is the new collision-free state, $\mathbf{A}$ is the inertial matrix, $\mathbf{b}$ contains the relative velocities, and $\mathbf{c}_-, \mathbf{c}_+$ are the lower bound and upper bound constraints, respectively. In the forward pass, $\mathbf{A}, \mathbf{b}$ are known variables, and the projected Gauss-Seidel (PGS) method is used to solve for $\mathbf{x}$. In Algorithm 1, we are computing the adjoint of $\mathbf{A}, \mathbf{b}$ given the adjoint $\overline{\mathbf{x}}$. For simplicity, we assume there are no constraints $\mathbf{c}_-, \mathbf{c}_+$ and it is now a pure Gauss-Seidel method which solves $\mathbf{A}\mathbf{x} + \mathbf{b} = \mathbf{0}$. Basically, we store variables in $\mathbf{V}_d, \mathbf{V}_{x1}$, and $\mathbf{V}_{x2}$. In the backward pass, we load the variables and compute the adjoints in a reverse order.

Algorithm 1 Adjoint of Gaussian-Siedel Solver

Input: matrix $\mathbf{A}$, vector$\mathbf{b}$, adjoint $\overline{\mathbf{x}}$, iterations $niter$
Output: adjoints $\overline{\mathbf{A}}$, $\overline{\mathbf{b}}$
Initialize vectors $\mathbf{V}_d(niter, rows(\mathbf{A}))$,
$\mathbf{V}_{x1}(niter, rows(\mathbf{A}))$, $\mathbf{V}_{x2}(niter, rows(\mathbf{A}))$.
# forward pass
for $t = 1$ **to** $niter$ **do**
  $\mathbf{V}_{x1}(t) = \mathbf{x}$
  for $i = 1$ **to** $rows(\mathbf{A})$ **do**
    $d = 0$
    for $j = 1$ **to** $rows(\mathbf{A})$ **do**
      if $j \neq i$ **then**
        $d = d + \mathbf{A}(i,i) * \mathbf{x}(j)$
      end if
    end for
    $\mathbf{V}_d(t,i) = d$
    $\mathbf{x}(i) = (\mathbf{b}(i) - d)/\mathbf{A}(i,i)$
  end for
  $\mathbf{V}_{x2}(t,i) = \mathbf{x}(i)$
end for
# start backward process
$\mathbf{A} = \mathbf{0}, \mathbf{b} = \mathbf{0}$
for $t$niter **to** 1 **do**
  for $i = rows(\mathbf{A})$ **to** 1 **do**
    $\overline{d} = 0$
    $\overline{d} = \overline{d} - \overline{\mathbf{x}}(i)/\mathbf{A}(i,i)$
    $\overline{\mathbf{b}} = \overline{\mathbf{b}} + \overline{\mathbf{x}}(i)/\mathbf{A}(i,i)$
    $\overline{\mathbf{A}}(i,i) = \overline{\mathbf{A}}(i,i) - \overline{x}(i) * [\mathbf{b}(i) - \mathbf{V}_d(t,i)]/\mathbf{A}(i,i)^2$
    $\overline{\mathbf{x}}(i) = 0$
    for $j = rows(\mathbf{A})$ **to** i+1 **do**
      $\overline{\mathbf{A}}(i,j) = \overline{\mathbf{A}}(i,j)d + \overline{d} * \mathbf{V}_{x1}(t,j)$
      $\overline{\mathbf{x}}(j) = \overline{\mathbf{x}}(j) + \overline{d} * \mathbf{A}(i,j)$
    end for
    for $j = i - 1$ **to** 1 **do**
      $\overline{\mathbf{A}}(i,j) = \overline{\mathbf{A}}(i,j)d + \overline{d} * \mathbf{V}_{x2}(t,j)$
      $\overline{\mathbf{x}}(j) = \overline{\mathbf{x}}(j) + \overline{d} * \mathbf{A}(i,j)$
    end for
  end for
end for

D. Model Mismatch/Transplant

Our articulated body simulation has similar functionality to MuJoCo (Todorov et al., 2012) in terms of physical and contact modeling. We first trained the 'MuJoCo Ant' model using our simulation as the environment. Upon convergence, we directly tested our model in MuJoCo, while keeping all robot configuration the same. As shown in Table 5, without any retraining, the trained model is able to obtain non-trivial reward in MuJoCo.

E. Other Limitations and Future Directions

In addition to some possible directions to further explore, as mentioned in Appendix A, our method can benefit from additional enhancement. The current simulator can potentially provide more options for users. For example, we have not provided options to select different contact models. The current LCP formula using iterative solver is fast but sometimes not accurate enough. Implementing extra contact models like exact LCP solvers or convex soft contact model can help the simulator adapt to more applications. Moreover,

Method	Reward in MuJoCo
w/o pretraining	868±77
w/ pretraining (Ours)	**1735±1066**

Table 5. Model transplant results on MuJoCo Ant. After pretraining in our simulation environment, the model can be directly transplanted to a new environment, MuJoCo, with non-trivial reward. This shows that our simulator behaves similarly with MuJoCo.

we assume that each link is a rigid body. It would be natural to extend this work to support deformable objects such as cloth (Todorov et al., 2012; Qiao et al., 2020).

We also plan to build on top of the current simulator to provide more comprehensive tool sets for users. Furthermore, it is worth exploring how differentiable physics can improve the sampling strategies and optimization of deep reinforcement learning models. Lastly, we hope to deploy our algorithms on real-world robots to perform a wide range of complex tasks.

F. Code for Profiling Autodiff Tools

ADF. We post below the code to run the backpropagation of ADF (Leal et al., 2018). As we described in Section 6.1, ADF can be used to run the forward simulation of articulated body dynamics but fails to complete the backpropagation in a reasonable time. We found out that ADF will greatly slow down when the computation graph goes deeper. Here is an example where we repeat a simple operation iteratively. When the iteration number is 28, it already takes a long time to backpropagate. In articulated body simulations, the depth of computation is far larger than 28, so ADF fails to compute the corresponding gradients.

```
#include <autodiff/reverse.hpp>
using namespace autodiff;

// The single-variable function for
// which derivatives are needed
var f(var x)
{
    int num_iter = 28;
    for (int i=0; i<num_iter; i++)
      x = x + 1 / x;
    return x;
}

int main()
{
    var x = 2.0;   // the input variable x
    var u = f(x);  // the output variable u

    // evaluate the derivative of u
    // with respect to x
    auto [ux] = derivatives(u, wrt(x));

}
```

JAX. JAX is based on Python and is hard to integrate into our C++ simulator. Therefore, we write simplified Python code to do the profiling. The input parameters to this code are the number of robots and simulation length. Each robot in the simulation environment of Section 6.1 has 37 DoFs, so we multiply 37 by the number of robots and get the dimension of the states variable. The operations in one simulation loop are far simpler than the articulated body dynamics.

```
from jax import grad
import jax.numpy as jnp
from jax import jit
import numpy as np
import sys

class RunBW(object):
  def __init__(self, dim, n_step):
    self.dim = dim
    self.n_step = n_step

  def test(self, x):
    dim = self.dim
    n_step = self.n_step
    for s in range(n_step):
      var = []
      for k in range(dim):
        var.append(x[k])
      for i in range(dim):
        k = i % (dim -1) + 1
        var[k] = var[k-1] + 1/var[k]
      var = [jnp.expand_dims(v, 0) for
        v in var]
      x = jnp.concatenate(var, 0)
    return jnp.sum(x)

if __name__ == '__main__':
  dim = 37 * int(sys.argv[1])
  n_step = int(sys.argv[2])
  rb = RunBW(dim, n_step)
  ini_state = np.ones([dim])
  grad_test = grad(rb.test)
  result = grad_test(ini_state)
  result = rb.test(ini_state)
```

G. Differentiation of Articulated Body Algorithm

This section describes the list of adjoint operations. We start with basic operators and go over all adjoints used in our implementation. $\overline{(\cdot)}$ is denoted as the adjoint of a variable.
Scalar multiply:

$$\begin{aligned} a &= bc \\ \frac{\partial \phi}{\partial b} &= \frac{\partial \phi}{\partial a}\frac{\partial a}{\partial b} \\ \overline{b} &= \overline{a}c, \overline{c} = b\overline{a} \end{aligned} \tag{20}$$

Matrix vector multiply:

$$\begin{aligned} \mathbf{v} &= \mathbf{m}\mathbf{a} \\ \overline{\mathbf{m}}_{ij} &= \overline{\mathbf{v}}_i \mathbf{a}_j, \overline{\mathbf{a}}_j = \sum \overline{\mathbf{v}}_k \mathbf{m}_{kj} \\ \overline{\mathbf{m}} &= \overline{\mathbf{v}}\mathbf{a}^T \\ \overline{\mathbf{a}} &= \mathbf{m}^T\overline{\mathbf{v}} \end{aligned} \tag{21}$$

Matrix multiply:

$$\begin{aligned} \mathbf{c} &= \mathbf{a}\mathbf{b} \\ \overline{\mathbf{a}} &= \overline{\mathbf{c}}\mathbf{b}^T \\ \overline{\mathbf{b}} &= \mathbf{a}^T\overline{\mathbf{c}} \end{aligned} \tag{22}$$

Matrix $\mathbf{A}^T\mathbf{B}\mathbf{A}$:

$$\begin{aligned} \mathbf{c} &= \mathbf{a}^T\mathbf{b}\mathbf{a} \\ \overline{\mathbf{a}} &= \mathbf{b}^T\mathbf{a}\overline{\mathbf{c}} + \mathbf{b}\mathbf{a}\overline{\mathbf{c}}^T \\ \overline{\mathbf{b}} &= \mathbf{a}\overline{\mathbf{c}}\mathbf{a}^T \end{aligned} \tag{23}$$

Matrix inverse:

$$\begin{aligned} \mathbf{b} &= \mathbf{a}^{-1} \\ \frac{\partial \mathbf{b}}{\partial \mathbf{q}} &= -\mathbf{b}\frac{\partial \mathbf{a}}{\partial \mathbf{q}}\mathbf{b} = -\sum_k \sum_t \mathbf{b}_{ij}\frac{\partial \mathbf{a}_{jk}}{\partial \mathbf{q}}\mathbf{b}_{kt} \\ \overline{\mathbf{a}}_{jk} &= -\sum_i \sum_t \mathbf{b}_{ij}\mathbf{b}_{kt}\overline{\mathbf{b}}_{it} \\ \overline{\mathbf{a}} &= -\mathbf{b}^T\overline{\mathbf{b}}\mathbf{b}^T \end{aligned} \tag{24}$$

Multiply three matrices:

$$\begin{aligned} \mathbf{m} &= \mathbf{a}\mathbf{b}\mathbf{c} \\ \overline{\mathbf{a}} &= \overline{\mathbf{m}}\mathbf{c}^T\mathbf{b}^T \\ \overline{\mathbf{b}} &= \mathbf{a}^T\overline{\mathbf{m}}\mathbf{c}^T \\ \overline{\mathbf{c}} &= \mathbf{b}^T\mathbf{a}^T\overline{\mathbf{m}} \end{aligned} \tag{25}$$

Vector cross product:

$$\begin{aligned} \mathbf{v} = \mathbf{a} \times \mathbf{b} &= [\mathbf{a}_2\mathbf{b}_3 - \mathbf{a}_3\mathbf{b}_2, \mathbf{a}_3\mathbf{b}_1 - \mathbf{a}_1\mathbf{b}_3, \mathbf{a}_1\mathbf{b}_2 - \mathbf{a}_2\mathbf{b}_1] \\ \overline{\mathbf{a}} &= [-\overline{\mathbf{v}}_2\mathbf{b}_3 + \overline{\mathbf{v}}_3\mathbf{b}_2, \overline{\mathbf{v}}_1\mathbf{b}_3 - \overline{\mathbf{v}}_3\mathbf{b}_1, -\overline{\mathbf{v}}_1\mathbf{b}_2 + \overline{\mathbf{v}}_2\mathbf{b}_1] = -\overline{\mathbf{v}} \times \mathbf{b} \\ \overline{\mathbf{b}} &= [\overline{\mathbf{v}}_2\mathbf{a}_3 - \overline{\mathbf{v}}_3\mathbf{a}_2, -\overline{\mathbf{v}}_1\mathbf{a}_3 + \overline{\mathbf{v}}_3\mathbf{a}_1, \overline{\mathbf{v}}_1\mathbf{a}_2 - \overline{\mathbf{v}}_2\mathbf{a}_1] = \overline{\mathbf{v}} \times \mathbf{a} \end{aligned} \tag{26}$$

Vector norm:

$$\begin{aligned} a &= \sqrt{\mathbf{b}\cdot\mathbf{b}} \\ \frac{\partial a}{\partial \mathbf{q}} &= -\frac{1}{2}\frac{1}{\sqrt{\mathbf{b}\cdot\mathbf{b}}}(\frac{\partial \mathbf{b}}{\partial \mathbf{q}}\cdot\mathbf{b} + \mathbf{b}\cdot\frac{\partial \mathbf{b}}{\partial \mathbf{q}}) = \frac{1}{a}\frac{\partial \mathbf{b}}{\partial \mathbf{q}}\cdot\mathbf{b} \\ \overline{\mathbf{b}} &= \frac{\overline{a}}{a}\mathbf{b} \end{aligned} \tag{27}$$

Spatial transform: apply():

$$\begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} = st_{apply}(\begin{bmatrix} \mathbf{E} \\ \mathbf{r} \end{bmatrix}, \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \end{bmatrix}) = \begin{bmatrix} \mathbf{E}\mathbf{a}_1 \\ \mathbf{E}(\mathbf{a}_2 - \mathbf{r}\times\mathbf{a}_1) \end{bmatrix} \tag{28}$$

$$\begin{bmatrix} \overline{\mathbf{E}} \\ \overline{\mathbf{r}} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1\overline{\mathbf{b}}_1^T + (\mathbf{a}_2 - \mathbf{r}\times\mathbf{a}_1)\overline{\mathbf{b}}_2^T \\ \mathbf{E}^T\overline{\mathbf{b}}_2\times\mathbf{a}_1 \end{bmatrix}, \begin{bmatrix} \overline{\mathbf{a}}_1 \\ \overline{\mathbf{a}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{E}^T\overline{\mathbf{b}}_1 - \mathbf{E}^T\overline{\mathbf{b}}_2\times\mathbf{r} \\ \mathbf{E}^T\overline{\mathbf{b}}_2 \end{bmatrix} \tag{29}$$

Spatial transform: apply-inv():

$$\begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} = st_{apply-inv}(\begin{bmatrix} \mathbf{E} \\ \mathbf{r} \end{bmatrix}, \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \end{bmatrix}) = \begin{bmatrix} \mathbf{E}^T\mathbf{a}_1 \\ \mathbf{E}^T\mathbf{a}_2 + \mathbf{r}\times(\mathbf{E}^T\mathbf{a}_1) \end{bmatrix} \tag{30}$$

$$\begin{bmatrix} \overline{\mathbf{E}} \\ \overline{\mathbf{r}} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1(\overline{\mathbf{b}}_1 + \overline{\mathbf{b}}_2\times\mathbf{r})^T + \mathbf{a}_2\overline{\mathbf{b}}_2^T \\ -\overline{\mathbf{b}}_2\times\mathbf{E}^T\mathbf{a}_1 \end{bmatrix}, \begin{bmatrix} \overline{\mathbf{a}}_1 \\ \overline{\mathbf{a}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{E}(\overline{\mathbf{b}}_1 + \overline{\mathbf{b}}_2\times\mathbf{r}) \\ \mathbf{E}\overline{\mathbf{b}}_2 \end{bmatrix} \tag{31}$$

Spatial transform: apply-trans():

$$\begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} = st_{apply-trans}(\begin{bmatrix} \mathbf{E} \\ \mathbf{r} \end{bmatrix}, \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \end{bmatrix}) = \begin{bmatrix} \mathbf{E}^T\mathbf{a}_1 + \mathbf{r}\times(\mathbf{E}^T\mathbf{a}_2) \\ \mathbf{E}^T\mathbf{a}_2 \end{bmatrix} \tag{32}$$

$$\begin{bmatrix} \overline{\mathbf{E}} \\ \overline{\mathbf{r}} \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{b}}_1\mathbf{a}_1^T + (\overline{\mathbf{b}}_1\times\mathbf{r} + \overline{\mathbf{b}}_2)\mathbf{a}_2^T \\ -\overline{\mathbf{b}}_1\times\mathbf{E}^T\mathbf{a}_2 \end{bmatrix}, \begin{bmatrix} \overline{\mathbf{a}}_1 \\ \overline{\mathbf{a}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{E}\overline{\mathbf{b}}_1 \\ \mathbf{E}(\overline{\mathbf{b}}_1\times\mathbf{r} + \overline{\mathbf{b}}_2) \end{bmatrix} \tag{33}$$

Spatial transform: apply-invtrans():

$$\begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} = st_{apply-invtrans}(\begin{bmatrix} \mathbf{E} \\ \mathbf{r} \end{bmatrix}, \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \end{bmatrix}) = \begin{bmatrix} \mathbf{E}^T(\mathbf{a}_1 - \mathbf{r}\times\mathbf{a}_2) \\ \mathbf{E}^T\mathbf{a}_2 \end{bmatrix} \tag{34}$$

$$\begin{bmatrix} \overline{\mathbf{E}} \\ \overline{\mathbf{r}} \end{bmatrix} = \begin{bmatrix} (\mathbf{a}_1 - \mathbf{r}\times\mathbf{a}_2)\overline{\mathbf{b}}_1^T + \mathbf{a}_2\overline{\mathbf{b}}_2^T \\ \mathbf{E}\overline{\mathbf{b}}_1\times\mathbf{a}_2 \end{bmatrix}, \begin{bmatrix} \overline{\mathbf{a}}_1 \\ \overline{\mathbf{a}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{E}\overline{\mathbf{b}}_1 \\ -\mathbf{E}\overline{\mathbf{b}}_1\times\mathbf{r} + \mathbf{E}\overline{\mathbf{b}}_2 \end{bmatrix} \tag{35}$$

Spatial transform: multiply():

$$\begin{bmatrix} \mathbf{E}_0 \\ \mathbf{r}_0 \end{bmatrix} = st_{multiply}(\begin{bmatrix} \mathbf{E}_1 \\ \mathbf{r}_1 \end{bmatrix}, \begin{bmatrix} \mathbf{E}_2 \\ \mathbf{r}_2 \end{bmatrix}) = \begin{bmatrix} \mathbf{E}_1\mathbf{E}_2 \\ \mathbf{r}_2 + \mathbf{E}_2^T\mathbf{r}_1 \end{bmatrix} \tag{36}$$

$$\begin{bmatrix} \overline{\mathbf{E}}_1 \\ \overline{\mathbf{r}}_1 \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{E}}_0 \mathbf{E}_2^T \\ \mathbf{E}_2 \overline{\mathbf{r}}_0 \end{bmatrix}, \begin{bmatrix} \overline{\mathbf{E}}_2 \\ \overline{\mathbf{r}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{E}_1^T \overline{\mathbf{E}}_0 + \mathbf{r}_1 \overline{\mathbf{r}}_0^T \\ \overline{\mathbf{r}}_0 \end{bmatrix} \tag{37}$$

Spatial motion: crossm():

$$\begin{bmatrix} \mathbf{w}_0 \\ \mathbf{v}_0 \end{bmatrix} = sm_{crossm}(\begin{bmatrix} \mathbf{w}_1 \\ \mathbf{v}_1 \end{bmatrix}, \begin{bmatrix} \mathbf{w}_2 \\ \mathbf{v}_2 \end{bmatrix}) = \begin{bmatrix} \mathbf{w}_1 \times \mathbf{w}_2 \\ \mathbf{w}_1 \times \mathbf{v}_2 + \mathbf{v}_1 \times \mathbf{w}_2 \end{bmatrix} \tag{38}$$

$$\begin{bmatrix} \overline{\mathbf{w}}_1 \\ \overline{\mathbf{v}}_1 \end{bmatrix} = \begin{bmatrix} -\overline{\mathbf{w}}_0 \times \mathbf{w}_2 - \overline{\mathbf{v}}_0 \times \mathbf{v}_2 \\ -\overline{\mathbf{v}}_0 \times \mathbf{w}_2 \end{bmatrix}, \begin{bmatrix} \overline{\mathbf{w}}_2 \\ \overline{\mathbf{v}}_2 \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{w}}_0 \times \mathbf{w}_1 + \overline{\mathbf{v}}_0 \times \mathbf{v}_1 \\ \overline{\mathbf{v}}_0 \times \mathbf{w}_1 \end{bmatrix} \tag{39}$$

Spatial motion: crossf():

$$\begin{bmatrix} \mathbf{w}_0 \\ \mathbf{v}_0 \end{bmatrix} = sm_{crossf}(\begin{bmatrix} \mathbf{w}_1 \\ \mathbf{v}_1 \end{bmatrix}, \begin{bmatrix} \mathbf{w}_2 \\ \mathbf{v}_2 \end{bmatrix}) = \begin{bmatrix} \mathbf{w}_1 \times \mathbf{w}_2 + \mathbf{v}_1 \times \mathbf{v}_2 \\ \mathbf{w}_1 \times \mathbf{v}_2 \end{bmatrix} \tag{40}$$

$$\begin{bmatrix} \overline{\mathbf{w}}_1 \\ \overline{\mathbf{v}}_1 \end{bmatrix} = \begin{bmatrix} -\overline{\mathbf{w}}_0 \times \mathbf{w}_2 - \overline{\mathbf{v}}_0 \times \mathbf{v}_2 \\ -\overline{\mathbf{w}}_0 \times \mathbf{v}_2 \end{bmatrix}, \begin{bmatrix} \overline{\mathbf{w}}_2 \\ \overline{\mathbf{v}}_2 \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{w}}_0 \times \mathbf{w}_1 \\ \overline{\mathbf{w}}_0 \times \mathbf{v}_1 + \overline{\mathbf{v}}_0 \times \mathbf{w}_1 \end{bmatrix} \tag{41}$$

Spatial dyad: mul-ori():

$$\begin{bmatrix} \mathbf{w}_0 \\ \mathbf{v}_0 \end{bmatrix} = sd_{mul-inv}(\begin{bmatrix} \mathbf{m}_{11} & \mathbf{m}_{12} \\ \mathbf{m}_{21} & \mathbf{m}_{22} \end{bmatrix}, \begin{bmatrix} \mathbf{w} \\ \mathbf{v} \end{bmatrix}) = \begin{bmatrix} \mathbf{m}_{11}\mathbf{w} + \mathbf{m}_{12}\mathbf{v} \\ \mathbf{m}_{21}\mathbf{w} + \mathbf{m}_{22}\mathbf{v} \end{bmatrix} \tag{42}$$

$$\begin{bmatrix} \overline{\mathbf{m}}_{11} & \overline{\mathbf{m}}_{12} \\ \overline{\mathbf{m}}_{21} & \overline{\mathbf{m}}_{22} \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{w}}_0 \mathbf{w}^T & \overline{\mathbf{w}}_0 \mathbf{v}^T \\ \overline{\mathbf{v}}_0 \mathbf{w}^T & \overline{\mathbf{v}}_0 \mathbf{v}^T \end{bmatrix}, \begin{bmatrix} \overline{\mathbf{w}} \\ \overline{\mathbf{v}} \end{bmatrix} = \begin{bmatrix} \mathbf{m}_{11}^T \overline{\mathbf{w}}_0 + \mathbf{m}_{21}^T \overline{\mathbf{v}}_0 \\ \mathbf{m}_{12}^T \overline{\mathbf{w}}_0 + \mathbf{m}_{22}^T \overline{\mathbf{v}}_0 \end{bmatrix} \tag{43}$$

Spatial dyad: mul-inv():

$$\begin{bmatrix} \mathbf{w}_0 \\ \mathbf{v}_0 \end{bmatrix} = sd_{mul-inv}(\begin{bmatrix} \mathbf{m}_{11} & \mathbf{m}_{12} \\ \mathbf{m}_{21} & \mathbf{m}_{22} \end{bmatrix}, \begin{bmatrix} \mathbf{w} \\ \mathbf{v} \end{bmatrix}) = \begin{bmatrix} \mathbf{m}_{11}^T \mathbf{w} + \mathbf{m}_{12}^T \mathbf{v} \\ \mathbf{m}_{21}^T \mathbf{w} + \mathbf{m}_{22}^T \mathbf{v} \end{bmatrix} \tag{44}$$

$$\begin{bmatrix} \overline{\mathbf{m}}_{11} & \overline{\mathbf{m}}_{12} \\ \overline{\mathbf{m}}_{21} & \overline{\mathbf{m}}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{w}\overline{\mathbf{w}}_0^T & \mathbf{v}\overline{\mathbf{w}}_0^T \\ \mathbf{w}\overline{\mathbf{v}}_0^T & \mathbf{v}\overline{\mathbf{v}}_0^T \end{bmatrix}, \begin{bmatrix} \overline{\mathbf{w}} \\ \overline{\mathbf{v}} \end{bmatrix} = \begin{bmatrix} \mathbf{m}_{11} \overline{\mathbf{w}}_0 + \mathbf{m}_{21} \overline{\mathbf{v}}_0 \\ \mathbf{m}_{12} \overline{\mathbf{w}}_0 + \mathbf{m}_{22} \overline{\mathbf{v}}_0 \end{bmatrix} \tag{45}$$

Spatial dyad: $\mathbf{v}\mathbf{v}^T()$:

$$\begin{bmatrix} \mathbf{m}_{11} & \mathbf{m}_{12} \\ \mathbf{m}_{21} & \mathbf{m}_{22} \end{bmatrix} = sd_{vvT}(\begin{bmatrix} \mathbf{w}_1 \\ \mathbf{v}_1 \end{bmatrix}, \begin{bmatrix} \mathbf{w}_2 \\ \mathbf{v}_2 \end{bmatrix}) = \begin{bmatrix} \mathbf{w}_1 \mathbf{w}_2^T & \mathbf{w}_1 \mathbf{v}_2^T \\ \mathbf{v}_1 \mathbf{w}_2^T & \mathbf{v}_1 \mathbf{v}_2^T \end{bmatrix} \tag{46}$$

$$\begin{bmatrix} \overline{\mathbf{w}}_1 \\ \overline{\mathbf{v}}_1 \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{m}}_{11} \mathbf{w}_2 + \overline{\mathbf{m}}_{12} \mathbf{v}_2 \\ \overline{\mathbf{m}}_{21} \mathbf{w}_2 + \overline{\mathbf{m}}_{22} \mathbf{v}_2 \end{bmatrix}, \begin{bmatrix} \overline{\mathbf{w}}_2 \\ \overline{\mathbf{v}}_2 \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{m}}_{11}^T \mathbf{w}_1 + \overline{\mathbf{m}}_{21}^T \mathbf{v}_1 \\ \overline{\mathbf{m}}_{21}^T \mathbf{w}_1 + \overline{\mathbf{m}}_{22}^T \mathbf{v}_1 \end{bmatrix} \tag{47}$$

Spatial dyad: $\mathbf{v}_\times$:

$$\begin{bmatrix} \mathbf{m}_{00} & \mathbf{m}_{01} & \mathbf{m}_{02} \\ \mathbf{m}_{10} & \mathbf{m}_{11} & \mathbf{m}_{12} \\ \mathbf{m}_{20} & \mathbf{m}_{21} & \mathbf{m}_{22} \end{bmatrix} = ([\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2]^T)_\times = \begin{bmatrix} 0 & -\mathbf{v}_2 & \mathbf{v}_1 \\ \mathbf{v}_2 & 0 & -\mathbf{v}_0 \\ -\mathbf{v}_1 & \mathbf{v}_0 & 0 \end{bmatrix} \tag{48}$$

$$\begin{bmatrix} \overline{\mathbf{v}}_0 \\ \overline{\mathbf{v}}_1 \\ \overline{\mathbf{v}}_2 \end{bmatrix} = \begin{bmatrix} -\overline{\mathbf{m}}_{12} + \overline{\mathbf{m}}_{21} \\ \overline{\mathbf{m}}_{02} - \overline{\mathbf{m}}_{20} \\ -\overline{\mathbf{m}}_{01} + \overline{\mathbf{m}}_{10} \end{bmatrix} \tag{49}$$

Spatial dyad: st2sd():

$$\begin{bmatrix} \mathbf{n}_{11} & \mathbf{n}_{12} \\ \mathbf{n}_{21} & \mathbf{n}_{22} \end{bmatrix} = sd_{st2sd}(\begin{bmatrix} \mathbf{E} \\ \mathbf{r} \end{bmatrix}) = \begin{bmatrix} \mathbf{E} & \mathbf{0} \\ -\mathbf{E}\mathbf{r}_\times & \mathbf{E} \end{bmatrix} \tag{50}$$

$$\begin{bmatrix} \overline{\mathbf{E}} \\ \overline{\mathbf{r}} \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{n}}_{11}^T + \overline{\mathbf{n}}_{22}^T - \mathbf{r}_\times \overline{\mathbf{n}}_{21} \\ -(\mathbf{E}\overline{\mathbf{n}}_{21})_\times \end{bmatrix} \tag{51}$$

Spatial dyad: shift():

$$\mathbf{n} = sd_{shift}(\begin{bmatrix} \mathbf{m}_{11} & \mathbf{m}_{12} \\ \mathbf{m}_{21} & \mathbf{m}_{22} \end{bmatrix}, \begin{bmatrix} \mathbf{E} \\ \mathbf{r} \end{bmatrix}) = \mathbf{a}^T\mathbf{b}\mathbf{a}(st2sd(\begin{bmatrix} \mathbf{E} \\ \mathbf{r} \end{bmatrix}), \begin{bmatrix} \mathbf{m}_{11} & \mathbf{m}_{12} \\ \mathbf{m}_{21} & \mathbf{m}_{22} \end{bmatrix}) \tag{52}$$

Adjoints of shift() can be decomposed into the adjionts of $\mathbf{A}^T\mathbf{B}\mathbf{A}()$ and $st2sd()$.

Quaternion: mul_vec():

$$\begin{bmatrix} \mathbf{q}_1.x \\ \mathbf{q}_1.y \\ \mathbf{q}_1.z \\ \mathbf{q}_1.w \end{bmatrix} = qt_{mul\_vec}(\begin{bmatrix} \mathbf{q}_2.x \\ \mathbf{q}_2.y \\ \mathbf{q}_2.z \\ \mathbf{q}_2.w \end{bmatrix}, \begin{bmatrix} \mathbf{v}.x \\ \mathbf{v}.y \\ \mathbf{v}.z \end{bmatrix}) = \begin{bmatrix} \mathbf{q}_2.w * \mathbf{v}.x + \mathbf{q}_2.y * \mathbf{v}.z - \mathbf{q}_2.z * \mathbf{v}.y \\ \mathbf{q}_2.w * \mathbf{v}.y + \mathbf{q}_2.z * \mathbf{v}.x - \mathbf{q}_2.x * \mathbf{v}.z \\ \mathbf{q}_2.w * \mathbf{v}.z + \mathbf{q}_2.x * \mathbf{v}.y - \mathbf{q}_2.y * \mathbf{v}.x \\ -\mathbf{q}_2.x * \mathbf{v}.x - \mathbf{q}_2.y * \mathbf{v}.y - \mathbf{q}_2.z * \mathbf{v}.z \end{bmatrix} \tag{53}$$

$$\begin{bmatrix} \overline{\mathbf{q}}_2.x \\ \overline{\mathbf{q}}_2.y \\ \overline{\mathbf{q}}_2.z \\ \overline{\mathbf{q}}_2.w \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{q}}_1.x * q.w - \overline{\mathbf{q}}_1.y * q.z + \overline{\mathbf{q}}_1.z * q.y - \overline{\mathbf{q}}_1.w * q.x \\ \overline{\mathbf{q}}_1.x * q.z + \overline{\mathbf{q}}_1.y * q.w - \overline{\mathbf{q}}_1.z * q.x - \overline{\mathbf{q}}_1.w * q.y \\ -\overline{\mathbf{q}}_1.x * q.y + \overline{\mathbf{q}}_1.y * q.x + \overline{\mathbf{q}}_1.z * q.w - \overline{\mathbf{q}}_1.w * q.z \end{bmatrix} \tag{54}$$

$$\begin{bmatrix} \mathbf{v}.x \\ \mathbf{v}.y \\ \mathbf{v}.z \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{q}}_1.x * \mathbf{q}_2.w - \overline{\mathbf{q}}_1.y * \mathbf{q}_2.z + \overline{\mathbf{q}}_1.z * \mathbf{q}_2.y - \overline{\mathbf{q}}_1.w * \mathbf{q}_2.x \\ \overline{\mathbf{q}}_1.x * \mathbf{q}_2.z + \overline{\mathbf{q}}_1.y * \mathbf{q}_2.w - \overline{\mathbf{q}}_1.z * \mathbf{q}_2.x - \overline{\mathbf{q}}_1.w * \mathbf{q}_2.y \\ -\overline{\mathbf{q}}_1.x * \mathbf{q}_2.y + \overline{\mathbf{q}}_1.y * \mathbf{q}_2.x + \overline{\mathbf{q}}_1.z * \mathbf{q}_2.w - \overline{\mathbf{q}}_1.w * \mathbf{q}_2.z \end{bmatrix} \tag{55}$$

Quaternion: mul_qt():

$$\begin{bmatrix} \mathbf{q}.x \\ \mathbf{q}.y \\ \mathbf{q}.z \\ \mathbf{q}.w \end{bmatrix} = qt_{mul\_qt}(\begin{bmatrix} \mathbf{q}_1.x \\ \mathbf{q}_1.y \\ \mathbf{q}_1.z \\ \mathbf{q}_1.w \end{bmatrix}, \begin{bmatrix} \mathbf{q}_2.x \\ \mathbf{q}_2.y \\ \mathbf{q}_2.z \\ \mathbf{q}_2.w \end{bmatrix}) = \begin{bmatrix} \mathbf{q}_1.w * \mathbf{q}_2.x + \mathbf{q}_1.x * \mathbf{q}_2.w + \mathbf{q}_1.y * \mathbf{q}_2.z - \mathbf{q}_1.z * \mathbf{q}_2.y \\ \mathbf{q}_1.w * \mathbf{q}_2.y + \mathbf{q}_1.y * \mathbf{q}_2.w + \mathbf{q}_1.z * \mathbf{q}_2.x - \mathbf{q}_1.x * \mathbf{q}_2.z \\ \mathbf{q}_1.w * \mathbf{q}_2.z + \mathbf{q}_1.z * \mathbf{q}_2.w + \mathbf{q}_1.x * \mathbf{q}_2.y - \mathbf{q}_1.y * \mathbf{q}_2.x \\ \mathbf{q}_1.w * \mathbf{q}_2.w - \mathbf{q}_1.x * \mathbf{q}_2.x - \mathbf{q}_1.y * \mathbf{q}_2.y - \mathbf{q}_1.z * \mathbf{q}_2.z \end{bmatrix} \tag{56}$$

$$\begin{bmatrix} \overline{\mathbf{q}}_1.x \\ \overline{\mathbf{q}}_1.y \\ \overline{\mathbf{q}}_1.z \\ \overline{\mathbf{q}}_1.w \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{q}}.x * \mathbf{q}_2.w - \overline{\mathbf{q}}.y * \mathbf{q}_2.z + \overline{\mathbf{q}}.z * \mathbf{q}_2.y - \overline{\mathbf{q}}.w * \mathbf{q}_2.x \\ \overline{\mathbf{q}}.x * \mathbf{q}_2.z + \overline{\mathbf{q}}.y * \mathbf{q}_2.w - \overline{\mathbf{q}}.z * \mathbf{q}_2.x - \overline{\mathbf{q}}.w * \mathbf{q}_2.y \\ -\overline{\mathbf{q}}.x * \mathbf{q}_2.y + \overline{\mathbf{q}}.y * \mathbf{q}_2.x + \overline{\mathbf{q}}.z * \mathbf{q}_2.w - \overline{\mathbf{q}}.w * \mathbf{q}_2.z \\ \overline{\mathbf{q}}.x * \mathbf{q}_2.x + \overline{\mathbf{q}}.y * \mathbf{q}_2.y + \overline{\mathbf{q}}.z * \mathbf{q}_2.z + \overline{\mathbf{q}}.w * \mathbf{q}_2.w \end{bmatrix} \tag{57}$$

$$\begin{bmatrix} \overline{\mathbf{q}}_2.x \\ \overline{\mathbf{q}}_2.y \\ \overline{\mathbf{q}}_2.z \\ \overline{\mathbf{q}}_2.w \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{q}}.x * \mathbf{q}_1.w + \overline{\mathbf{q}}.y * \mathbf{q}_1.z - \overline{\mathbf{q}}.z * \mathbf{q}_1.y - \overline{\mathbf{q}}.w * \mathbf{q}_1.x \\ -\overline{\mathbf{q}}.x * \mathbf{q}_1.z + \overline{\mathbf{q}}.y * \mathbf{q}_1.w + \overline{\mathbf{q}}.z * \mathbf{q}_1.x - \overline{\mathbf{q}}.w * \mathbf{q}_1.y \\ \overline{\mathbf{q}}.x * \mathbf{q}_1.y - \overline{\mathbf{q}}.y * \mathbf{q}_1.x + \overline{\mathbf{q}}.z * \mathbf{q}_1.w - \overline{\mathbf{q}}.w * \mathbf{q}_1.z \\ \overline{\mathbf{q}}.x * \mathbf{q}_1.x + \overline{\mathbf{q}}.y * \mathbf{q}_1.y + \overline{\mathbf{q}}.z * \mathbf{q}_1.z + \overline{\mathbf{q}}.w * \mathbf{q}_1.w \end{bmatrix} \tag{58}$$

Please refer to our code for more details.