Collective Intelligence for Object Manipulation with Mobile Robots

So Kuroki¹, Tatsuya Matsushima¹, Jumpei Arima², Yutaka Matsuo³, Shixiang Shane Gu³, and Yujin Tang³

Abstract—While natural systems often present collective intelligence that allows them to self-organize and adapt to changes, the equivalent is missing in most artificial systems. We explore the possibility of such a system in the context of cooperative object manipulation using mobile robots. Although conventional works demonstrate potential solutions for the problem in restricted settings, they have computational and learning difficulties. More importantly, these systems do not possess the ability to adapt when facing environmental changes. In this work, we show that by distilling a planner derived from a gradient-based soft-body physics simulator into an attention-based neural network, our multi-robot manipulation system can achieve better performance than baselines. In addition, our system also generalizes to unseen configurations during training and is able to adapt toward task completions when external turbulence and environmental changes are applied.

I. INTRODUCTION

Collective intelligence plays an important role in natural systems, where various life forms often demonstrate self-organized and adaptive behaviors that are indispensable for them to survive the changing environment and achieve what is beyond the capabilities of any individual among the same species. For example, army ants are able to use their bodies to form a bridge that adapts to the width of a gap for the others to pass by [1]. And a school of fish can change its formation during motion quickly to get rid of a predator or to hunt so that their survival rate is maximized [2]. If these collective behaviors have allowed the species to survive the cruelty of evolution, can we build artificial systems that are based on the concept of collective intelligence and benefit from it too? In this work, we explore the possibility in object manipulations by a group of cooperative mobile robots.

As more mobile robots become readily available and move out from factories to diverse real-world settings, we see potential extensions of their applications in daily scenarios such as room cleaning, laundry folding, etc, where object manipulation through pushing is the common fundamental building block of the upper level skills. In addition, recent works have been focusing on achieving object manipulation systems with multiple mobile robots. Compared with a single robot setting, such systems are more valuable for several reasons [3]. For instance, while pushing a single rigid object at a time involves straight-forward kinematics, pushing a deformable object presents unique challenges as dynamics becomes more complex and dexterity demands multi-robot coordination. Moreover, building several simple robots can be more cost effective and fault tolerant than having a single powerful robot. Last but not least, in the process of implementing a cooperative mobile robot system, it is possible to get insights that lead to a better understanding of some fundamental problems in the social and life sciences.

Crucially, a multi-robot manipulation system allows dexterity because we have more flexible grasp wrench spaces by commanding the robots to the appropriate contact points and adjusting their relative distances between each other based on the target object’s state. Here, achieving system level coordination has been the main goal for many research works, and object manipulation using cooperative mobile robots can be viewed as a specific case of multi-robot coordination [4]. Early methods often assign leaders and followers roles to the robots so that the complex task is factored and heuristic solutions can be easily incorporated [5], [6]. In this scheme, although the leaders usually bear more computational burdens, the problem is approached in a structured way and the overall complexity is reduced. With the development of physics simulators [7]–[10] and faster hardware accelerators, we are witnessing better performance brought by trajectory optimization/generation based methods and learning based methods these days. For example, [11] obtained paths for...
mobile manipulators with methods based on RRT [12]. [15] uses reinforcement learning to generate the trajectories for mobile manipulation actions. While these works demonstrate potential solutions for object manipulation with mobile robots, trajectory optimization based methods may require a prohibitive amount of computing at each decision step and learning based methods can have difficulties in learning the task due to the long task horizons. Furthermore, these systems do not present the collective intelligence we mentioned earlier in the natural system that would allow them to adapt when facing changes and failures.

In this paper, we show that by distilling a planner derived from a gradient-based soft-body physics simulator into an attention [14] based neural network, our multi-robot manipulation system can achieve better performance than both trajectory optimization and reinforcement learning methods. We train our multi-robot system in a simulator, and demonstrate that it generalizes well to scenarios with unseen physics configurations and numbers of robots that are different from what is given during the training. After training, we zero-shot transfer the learned policy to the real world without observing significant drops in performance. Finally, our robots are able to self-adjust in the face of environmental changes. For example, we pulled away one of the robots during task execution, and observed a neighbor robot proactively replaced the missing worker’s position to help complete the task (see Figure 1). This cooperative and self-adaptive behavior demonstrates the emergent collective intelligence unseen in conventional systems.

II. RELATED WORK

A. Push Manipulation and Dexterity

In manipulation, it is important to predict the movement of the object with respect to the action, and to obtain the action sequence to move the object to the goal state. For movement prediction, early works studied under the quasi-static assumption [15], [16], which approximates the equations of motion in the horizontal direction, assuming that the vertical scale of the associated object is sufficiently small relative to the horizontal. Recently, we see a surge of data-driven methods applied to push manipulation. For instance, [17] models the stochastic nature of friction by sampling parameters from a set of distributions and validate the model by comparing simulation with the large scale experimental data released by [18]. To avoid explicit estimations of physics parameters, data-efficient forward model construction methods have been proposed to learn the dynamics directly [19], [20]. For action sequence computations, researchers have conducted experiments with RRT and MPPI [21] to obtain continuous trajectories [21]–[23]. Not surprisingly, data-driven methods are also applied in this area. For example, learning the next action with images of the current and the goal states as inputs is proposed in [24]. [25] summarizes push manipulation research. Finally, the latest works also put dexterity at its core [26], [27], and it is reasonable to believe that a multi-robot system is suitable for that purpose because it allows flexible grasp wrench spaces.

B. Collective Intelligence in Machine Learning and Robotics

Various works tried to incorporate collective intelligence concepts into artificial systems and have proved the merits in doing so [28]–[30]. For example, it has been shown that systems with neural cellular automata implementations can not only model complex 2D/3D patterns but also demonstrate strong robustness to external turbulences through regeneration [31], [32]. In multi-agent setups, complex collective behaviors emerge automatically to allow efficient cooperative/competitive strategies [33], coordinated modular locomotion controllers [34] and flexible structure designs [35], [36]. In addition to these simulated environments, roboticists have also reproduced well known collective behaviors found in natural systems in robotic systems. For instance, [37] studies the principle behind the army ants’ bridge construction behavior and identifies the control mechanism emerged from each individual’s decisions. Based on this, [38] then designed self-assemble robots that climb over each other to form amorphous structures. Similarly, fish schooling behaviors are also reproduced in underwater robots that present natural and self-organized flocking states [39], [40]. In the space of cooperative manipulation, multiple works have learned to control swam robotics. In the task of pushing a horizontal box, pushing with multiple robots completed the task more efficiently than with a single robot [41], [42].

C. Differentiable Physics Engines

Differentiable physics engines leverage differentiable programming for physics simulations. Relying on automatic differentiation, differentiable simulators propagate gradients through dynamics and can be utilized for parameter estimation [43], gradient-based trajectory optimization [7], [44], and policy learning [9], [45]. While most differentiable simulators are tailored for rigid-body physics [9], we see recent simulators implementing particle dynamics [7], [46] which, in the context of object manipulation, unlocks the learning of non-rigid object manipulations. For instance, [47] conducted sim2real transfer of dough manipulation with imitation learning with the help of such simulators. Our method is based on PlasticineLab [44], a differentiable simulator benchmark using DiffTaichi [7] that enables gradient-based trajectory optimization through particle dynamics.

D. Multi-Head Self-Attention

An attention module enables a network to develop fast weights that adapt quickly based on the input data. Since its release, we have witnessed successful applications of attention in many areas [14], [48]–[51]. In general, a self-attention module can be mathematically described as:

$$ Y = \text{softmax} \left( \frac{1}{\sqrt{d_{in}}} (XW_q)(XW_k)^\top (XW_v) \right) $$

where $X \in \mathbb{R}^{N \times d_{in}}$ is the input data, $Y \in \mathbb{R}^{N \times d_{out}}$ is the output, $W_q, W_k \in \mathbb{R}^{d_{in} \times d_k}$ and $W_v \in \mathbb{R}^{d_k \times d_v}$ are learnable projection matrices. If there are multiple sets of projection matrices, we can concatenate the outputs and have $Y = [Y_1, \cdots, Y_m]$, we call such a setting multi-head self-attention where $m$ is the number of heads.
III. PROPOSED METHOD

A. Problem Statement

We design and study our system in the settings where multiple mobile robots (e.g., Roomba) are required to push and manipulate a soft body object (e.g., a rope) to a user-defined goal pose. To concentrate on the more important module of motion planning, we assume the availability of a camera of global views in the environment that allows us to extract the pose information of both the robots and the target object. We define \( r(s_t) \) as a measurement of the system’s performance at time step \( t \):

\[
r(s_t) = \max_{1 \leq i,j \leq N_r} \left( \frac{f(s_i, s_g) - f(s_0, s_g)}{\max(1 - f(s_0, s_g), \epsilon)} \right)
\]

where \( s_t \) and \( s_g \) represent the current and the user-defined goal state of the target object respectively, \( f(s_i, s_j) \in [0,1] \) measures the similarity between \( s_i \) and \( s_j \) \((f(s_i, s_j) = 1, \forall i = j)\), and \( \epsilon \) is a small positive number to prevent the zero division error. In this paper, we use intersection over union (IoU) in the global camera’s image space as the similarity function \( f \).

B. Particle System Based Motion Planner

We use PlasticineLab (PL) [44] as our simulator, with which we derive and test a particle-based trajectory planner (see Figure 2 (a) top). Concretely, we model the target object as a particle system and sample particles to form data of shape \( N_p \times d_p \) to represent the current and the goal states of the target object \((s_t, s_g)\). \( N_p \) is the number of particles and \( d_p \) is the information dimension of each particle, in our experiments \( d_p = 6 \) for the positions and the linear velocities in the 3D space. We then ask PL to treat the mobile robots as rigid body manipulators, and represent the states of which with a matrix of shape \( N_r \times d_m \), where \( N_r \) is the number of robots and \( d_m = 5 \) contains each robot’s positions and linear velocities (we ignore \( v_z \) because we assume the linear velocities in the XY plane alone is sufficient to solve the problem, see Figure 2 (a) bottom). PL models the interaction between the target object and the robots with one-way coupling. Thanks to PL’s differentiable physics, we can easily acquire a velocity planner to command each robot to accomplish the task. Although the performance of this planner is better than the baselines (as we will show shortly), querying the planner at every time step is less efficient. Moreover, it is not straightforward for such a planner to adapt to environmental changes (e.g., changes of friction, particle masses, or the number of robots). We therefore propose to distill this planner into an attention-based neural network policy that overcomes these shortcomings while maintaining the performance.

C. Distilling the Planner

Our policy is based on the self-attention mechanism and is depicted in Figure 2 (b). The input to our policy is a matrix of shape \( N_r \times d_{in} \), where the rows describe the sensory information collected by each of the \( N_r \) robots. In our design, each robot’s sensory information is composed of four parts (i.e., \( d_{in} = d_a + d_b + d_c + d_d \)): (a) the positions of the particles of the target object in \( s_t \) in the robot’s frame \((d_a = 3N_p)\); (b) the positions of the particles of the target object in \( s_g \) in the robot’s frame \((d_b = 3N_p)\); (c) the relative position of its nearest neighbor robot \((d_c = 3)\); and (d) the vectors from the object particles in \( s_t \) to those in \( s_g \) \((d_d = 3N_p)\), this last component is common info to all robots. Receiving the input, our policy network transforms the data into features of shape \( N_r \times d_{feat} \) via a fully connected network (MLP), and then several layers of multi-head self-attentions are applied to the features. A visibility mask is used here to make sure each robot has access to only the data from itself and its nearest neighboring robot. Intuitively, this attention module allows each robot to aggregate its own knowledge about the world with those from
policy distillation (i.e., some robots’ sensory data into the situation in the real-world, we also introduced data delay in configurations. In addition, to better cope with the less ideal (e.g., perturbation of friction, rope yield stress, robot size, \( N_i \) poses. When deriving the planner in the simulator, we set 5 of Roombas to push manipulate the rope into user defined

We use a rope as our target object and command a group of Roombas to push manipulate the rope into user defined 

\[
\text{Fig. 3: Illustration of our real-world experimental setup.}
\]

D. Real-world Policy Deployment

Although our policy is only trained with data collected from the simulator, we deploy and test the policy in the real world with various configurations (see Figure 2 (c) for sample configurations) without any fine-tuning. Figure 3 gives our hardware setups. We mount a camera on the ceiling to capture global views, and then analyze the point cloud from the camera to extract the parts corresponding to the target object by matching the colors. Each robot bears a marker, from which its positions in the environment are detected. A central server is responsible for collecting sensory data, executing the policy and distributing the velocity commands to all robots. We wish to point out that, our system design allows distributed policy execution too. As we will show later in the experimental results, our distilled policy is robust to message delays commonly seen in a distributed system. To avoid jerky motions, we smooth the actions by averaging the latest \( H \) policy outputs.

IV. EXPERIMENTS

We answer the following questions via experiments.

- How does the particle system based motion planner compare with other methods?
- Can we distill the planner into a neural network policy and keep the performance?
- Does our distilled policy adapt to environmental changes and present collectively intelligent behaviors?

A. Experimental Setup

We use a rope as our target object and command a group of Roombas to push manipulate the rope into user defined poses. When deriving the planner in the simulator, we set \( N_p = 102 \) and \( N_r = 6 \), and apply domain randomization (e.g., perturbation of friction, rope yield stress, robot size, etc) so that the collected dataset contains various physics configurations. In addition, to better cope with the less ideal situation in the real-world, we also introduced data delay in policy distillation (i.e., some robots’ sensory data into the policy at time \( t \) are those from \( t - k \) where \( k \sim \text{Unif}(0, 10) \)). In policy distillation, we set \( d_{feat} = 128 \) and train on the collected dataset until the error converges. In tests, we use the same \( N_p \) but with \( N_r \in \{3, 4, 5\} \) robots. For each test, we fix the initial state of the robots and the rope but, depending on \( N_r \), sample the goal state from 50 (when test in sim) and 4 (when test in real) predefined configurations. In the real-world deployment, each Roomba has an ArUco marker so that the global camera can detect their positions. All data communication and command distributions are conducted via ROS. The robots’ control frequency is set to 10Hz and the action smoothing window size is \( H = 5 \). Although the sensory data (e.g., positions of the points and the robots, etc) are inevitably noisy in real-world compared with those in the simulator, our policy is still able to accomplish the tasks, demonstrating the robustness of our method.

We introduce several baseline methods to compare the performances. To confirm the advantage of our particle system based planner, we trained a PPO [52] agent with fully connected neural network policy, and a MPPI planner on the same task. We also compare the performance of the distilled policy with the teacher policy (i.e., the particle system based planner) to show that our method is able to maintain the performance. Finally, to demonstrate the capability of generalization, we distilled the planner into a fully connected network policy for comparison.

B. Evaluating the Particle System Based Planner

Figure 4 gives the learning curves of the particle system based planner and the baselines. In all experiments, we used \( N_r = 6 \) robots and the starting configurations of the robots and the ropes are fixed. On the left, we sample and fix a goal configuration and report the performance from five different random seeds, while on the right, we report the results from training and testing with five different goal configurations. The particle system based planner’s performance is clearly better than the two baseline methods, and the result is consistent with what is reported in recent works.

Confirming that our derived planner has better performance in simulation, we also want to investigate its sim2real
The mean reward and standard deviation for each episode of 5 tasks with the same seed. We clamp the reward to be greater than 0 for a better illustration.

**TABLE I:** PL planner performance. In each experiment, we test on four different goal states and report the mean and the standard deviation of $r(s_t)$ (incremental IoU).

| Task          | Performance |
|---------------|-------------|
| Pushing Box   | 0.62 ± 0.22 (sim) |
|               | 0.59 ± 0.28 (real) |
| Shaping Rope  | 0.62 ± 0.02 (sim) |
|               | 0.50 ± 0.15 (real) |

gap’s size so that we know what to expect when we distill the planner and evaluate our policy in the real-world. To this end, we applied the planner directly in the real-world and recorded its performances. In addition to the rope manipulation task, we also added a box pushing task (see Figure 5 for task snapshots). Table I summarizes our results, where for each run, we test the planner on four different goal configurations and we report the mean and the standard deviation of $r(s_t)$. The results show a small sim2real gap and suggest the planner be a good candidate as a teacher policy for our policy distillation.

C. Evaluating the Distilled Policy

After policy distillation, we evaluated the resulting policies in the simulation of the rope manipulation task. In the first set of tests, we used $N_r = 6$ robots and tested all the methods on 50 new goal states $s_{g}$’s. We report the mean and the standard deviation of $r(s_t)$ from each method in Table II, and all the distilled policies have the prefix “BC+” for behavior cloning. We observe no huge performance drops from the distilled policies. With a moderate sized fully connected network policy (the second row, the architecture is a two hidden layer MLP with 64 hidden units each), the score difference from the teacher policy is not statistically significant. However, because an MLP does not allow the system to adjust to environmental changes, we find an attention-based network policy to be more appropriate. Switching to such a network, we observe a slight performance drop (the third row) due to the architectural inductive bias. With the same trained policy, we introduce randomized data communication delays in the test, and we see barely any performance degrade (the fourth row). In fact, the performance is even better. As we mentioned previously, this communication delay is expected in real-world and the result shows that our policy can keep performance in a distributed execution environment. We also include the results from PPO that is trained from scratch in the last row. Although the method has the same MLP network as its policy network, its performance is far lower than any of the distilled policies, suggesting the crucial role played by our proposed method.

**TABLE II:** Performance in the task of rope manipulation. We show the test mean and the standard deviation of $r(s_t)$ with $N_r = 6$ robots on 50 new goal states. The distilled policies have “BC+” as their prefix. For comparison, we also include the performance from a PPO agent.

| Method                        | Performance |
|-------------------------------|-------------|
| PL Planner (teacher)          | 0.74 ± 0.28 |
| BC + MLP                      | 0.64 ± 0.22 |
| BC + Attention (w/o delay)    | 0.54 ± 0.27 |
| BC + Attention (w/ delay)     | 0.59 ± 0.19 |
| PPO + MLP                     | 0.40 ± 0.26 |

**TABLE III:** Generalization test results. We changed several configurations to those unseen during training/data collection and report the test scores. For the experiments in the simulator, we report the mean and standard deviation of $r(s_t)$ on 50 new goal states; for the experiments in the real-world, we report the same metrics from 5 new goal states.

| What’s changed? | BC+Attention | BC+MLP | PPO+MLP |
|-----------------|--------------|--------|---------|
| (Experiments in sim) |              |        |         |
| Friction        | 0.42 ± 0.25  | 0.50 ± 0.27 | 0.32 ± 0.22 |
| Rope yield stress| 0.49 ± 0.23  | 0.58 ± 0.25 | 0.34 ± 0.23 |
| Velocity limit  | 0.50 ± 0.25  | 0.42 ± 0.30 | 0.33 ± 0.23 |
| Robot radius    | 0.45 ± 0.27  | 0.56 ± 0.25 | 0.34 ± 0.24 |
| (Experiments in real) |          |        |         |
| $N_r = 5$       | 0.44 ± 0.10  | N/A    | N/A     |
| $N_r = 4$       | 0.40 ± 0.14  | N/A    | N/A     |
| $N_r = 3$       | 0.54 ± 0.14  | N/A    | N/A     |

**Fig. 4:** Comparison of rewards of a trajectory with PL and baselines in the task of rope manipulation with several robots. For comparison, we fixed the number of the robot at $N_r = 6$. (a) The mean reward and standard deviation for each episode of 5 seeds with the same task. (b) The mean reward and standard deviation for each episode of 5 tasks with the same seed. We clamp the reward to be greater than 0 for a better illustration.

**Fig. 5:** Applying the PL planner directly in the real-world. Robots push each object to the goal state (the blue shaded area).

**Fig. 6:** Test examples. We show the start and end states of the tasks in each column, and we superpose the final $r(s_t)$. 

This table shows the performance of different methods on the task of rope manipulation. The methods tested include PL Planner (teacher), BC + MLP, BC + Attention (w/o delay), BC + Attention (w/ delay), and PPO + MLP. The table also includes experiments in both simulation and real-world settings.
D. Behavior Analysis

In our last experiments, we applied even more aggressive environmental changes to the robots and analyzed the policy network’s attention matrices to get insights into their behaviors. More specifically, we conducted two sets of experiments in the real world. In the first trial, we command a group of \( N_r = 5 \) robots to push the rope into a specified pose (see Figure 7 top rows). And in the second trial, all the settings are kept identical except that we “kidnap” one of the robots in the middle of the task execution and let the rest of robots finish the job (see Figure 7 bottom rows). In the second run, a neighboring robot is able to replace the missing robot and help push the rope to the desired pose. This level of self-adaptiveness is unseen in conventional methods, and demonstrates the concept of collective intelligence in an artificial system. For further analysis, we plotted how the attention matrices changed in both trials and placed them below the corresponding screenshots in Figure 7. Comparing (a.2) and (b.2), the attention weights to the nearest robots suddenly became larger right after kidnapping, proving that the robots are adjusting and relying more on the information from the rest of the robots to accomplish the task.

V. CONCLUSION

In this work, we explore incorporating the idea of collective intelligence in object manipulations by a group of cooperative mobile robots. We find that by distilling a motion planner derived from a particle-based system into an attention-based policy network, our multi-robot system is able to keep the performance of the teacher planner, and at the same time, demonstrate strong generalization capability to both unseen physics properties and the change of the number of robots. The attention matrix in the policy network allows partial interpretation of the robots behaviors, where we observe critical pattern changes at the moment when external turbulence is applied. Most importantly, the system is able to adjust and complete the task in the face of such turbulence, fully demonstrating the self-organized behaviors commonly seen in natural systems. In the future, we wish to introduce more complex tasks to further challenge our system and we expect to see more inspiring collective behaviors.
[28] D. Ha and Y. Tang, “Collective intelligence for deep learning: A survey of recent developments,” *Collective Intelligence*, vol. 1, no. 1, p. 26339137221114874, 2022.

[29] M. Gauci, R. Nagpal, and M. Rubenstein, “Programmable self-disassembly for shape formation in large-scale robot collectives,” in *Distributed Autonomous Robotic Systems*, Springer, 2018, pp. 573–586.

[30] I. Slavkov, D. Carrillo-Zapata, N. Carranza, X. Diego, F. Jansson, J. Kaandorp, S. Hauert, and J. Sharpe, “Morphogenesis in robot swarms,” *Science Robotics*, vol. 3, no. 25, eaau9178, 2018.

[31] A. Mordvintsev, E. Randazzo, E. Niklasson, and M. Levin, “Growing neural cellular automata,” *Distill*, vol. 5, no. 2, e23, 2020.

[32] S. Sudhakaran, D. Grbic, S. Li, A. Katona, E. Najarro, C. Glanois, and S. Risi, “Growing 3d artefacts and functional machines with neural cellular automata,” *arXiv preprint arXiv:2103.08737*, 2021.

[33] L. Zheng, J. Yang, H. Cai, M. Zhou, W. Zhang, J. Wang, and Y. Yu, “Magent: A many-agent reinforcement learning platform for artificial collective intelligence,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[34] W. Huang, I. Mordatch, and D. Pathak, “One policy to control them all: Shared modular policies for agent-agnostic control,” in *International Conference on Machine Learning*, PMLR, 2020, pp. 4455–4464.

[35] D. Pathak, C. Lu, T. Darrell, P. Isola, and A. A. Efros, “Learning to control self-assembling morphologies: A study of generalization via modularity,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[36] E. Gonzalez, L. Houel, R. Nagpal, and M. Malley, “Influencing emergent self-assembled structures in robotic collectives through traffic control,” in *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, 2022, pp. 1601–1603.

[37] H. F. McCreery, G. Gemayel, A. I. Pais, S. Garnier, and R. Nagpal, “Hysteresis stabilizes dynamic control of self-assembled army ant constructions,” *Nature communications*, vol. 13, no. 1, pp. 1–13, 2022.

[38] M. Malley, B. Haghhighat, L. Houel, and R. Nagpal, “Eciton robotica: Design and algorithms for an adaptive self-assembling soft robot collective,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 4565–4571.

[39] M. Gauci, J. Chen, W. Li, T. J. Dodd, and R. Groß, “Self-organized aggregation without computation,” *The International Journal of Robotics Research*, vol. 33, no. 8, pp. 1145–1161, 2014.

[40] F. Berlinger, M. Gauci, and R. Nagpal, “Implicit coordination for 3D underwater collective behaviors in a fish-inspired robot swarm,” *Science Robotics*, vol. 6, no. 50, eabd8668, 2021.

[41] J. Shen, E. Xiao, Y. Liu, and C. Feng, “A deep reinforcement learning environment for particle robot navigation and object manipulation,” *arXiv preprint arXiv:2203.06464*, 2022.

[42] M. J. Mataric, M. Nilsson, and K. T. Simsaric, “Cooperative multi-robot box-pushing,” in *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Human Robot Interaction and Cooperative Robots, IEEE, vol. 3, 1995, pp. 556–561.

[43] P. Ma, T. Du, J. B. Tenenbaum, W. Matusik, and C. Gan, “Risp: Rendering-invariant state predictor with differentiable simulation and rendering for cross-domain parameter estimation,” *arXiv preprint arXiv:2205.05678*, 2022.

[44] Z. Huang, Y. Hu, T. Du, S. Zhou, H. Su, J. B. Tenenbaum, and C. Gan, “Plasticinelab: A soft-body manipulation benchmark with differentiable physics,” *The International Conference on Learning Representations (ICLR)*, 2021.

[45] M. A. Z. Mora, M. Peychev, S. Ha, M. Vechev, and S. Coros, “Pods: Policy optimization via differentiable simulation,” in *International Conference on Machine Learning*, PMLR, 2021, pp. 7805–7817.

[46] M. Macklin, Warp: A high-performance python framework for gpu simulation and graphics, https://github.com/nvidia/warp, NVIDIA GPU Technology Conference (GTC), 2022.

[47] C. Qi, X. Lin, and D. Held, “Learning closed-loop dough manipulation using a differentiable reset module,” *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 9857–9864, 2022.

[48] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh, “Set transformer: A framework for attention-based permutation-invariant neural networks,” in *International conference on machine learning*, PMLR, 2019, pp. 3744–3753.

[49] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, “Swin transformer: Hierarchical vision transformer using shifted windows,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10012–10022.

[50] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, “Decision transformer: Reinforcement learning via sequence modeling,” *Advances in neural information processing systems*, vol. 34, pp. 15084–15097, 2021.

[51] Y. Tang and D. Ha, “The sensory neuron as a transformer: Permutation-invariant neural networks for reinforcement learning,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 22574–22587, 2021.

[52] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.