SoftGym: Reinforcement Learning in Deformable Object Environments

Arjun Teh, Joshua Zhanson, Lance Zhou

***Abstract*—Most reinforcement learning algorithms that use simulations have simple configurations so that the state space can be tractable. However, tasks involving manipulation of deformable objects do not admit easy state representations besides arbitrary RGB scene input. We propose two variations on a pushing task involving deformable objects based on the NVIDIA FleX particle simulation environment and introduce several state representations for reinforcement learning with deformable objects: a center-of-masses representation, a sampling point-cloud representation, and a raycastint representation. to facilitate learning in these new environments. Finally, we apply Proximal Policy Optimization to these environments and show successful learning on both deformable object tasks.**

I. Introduction

Many objects and environments in the real world contain deformable parts, such as cloth, rope, wire, and paper. Robotic systems will need to be adaptable to these soft objects in order to succeed in the real world. Interaction with these soft objects has been significantly less investigated due to the difficulty of modeling complex object dynamics, accurately predicting object behavior, and the extremely large configuration space of the object. While guiding real-world manipulation and navigation with physical simulations is appealing, the computational cost of such simulations often makes real-time control guided by simulation infeasible. Also, due to a large and ill-defined configuration space of deformable objects, approaches such as dividing the task into a sequence of simpler small tasks tend to fail.

Recently, reinforcement learning has been applied to many control problems as a largely model-free data-driven method to learn control with minimal human supervision. Such learning, however, is not without its own shortcomings. Phrasing problems as Markov Decision Processes is rarely a simple task. Defining appropriate state and action spaces, especially state spaces that conform to the Markovian assumption (that is, each following state is dependent only on the immediate previous state) can be complex. Designing an appropriate yet realistic reward function poses an arguably even harder open problem. If a given reward function is too sparse, not enough learning signal is provided to the agent. Reward shaping can alleviate this problem, but requires a significant degree of hand-tailoring. It is not even easy to define or detect success in many real-world manipulation and control tasks. Also, the poor sample complexity of reinforcement learning algorithms often make directly applying reinforcement learning in the real world infeasible. Training in simulation, however, raises the question of transferring learned policies from simulation to the real world. However, despite these shortcomings, reinforcement learning remains an appealing approach to learning robotic control in a variety of domains.

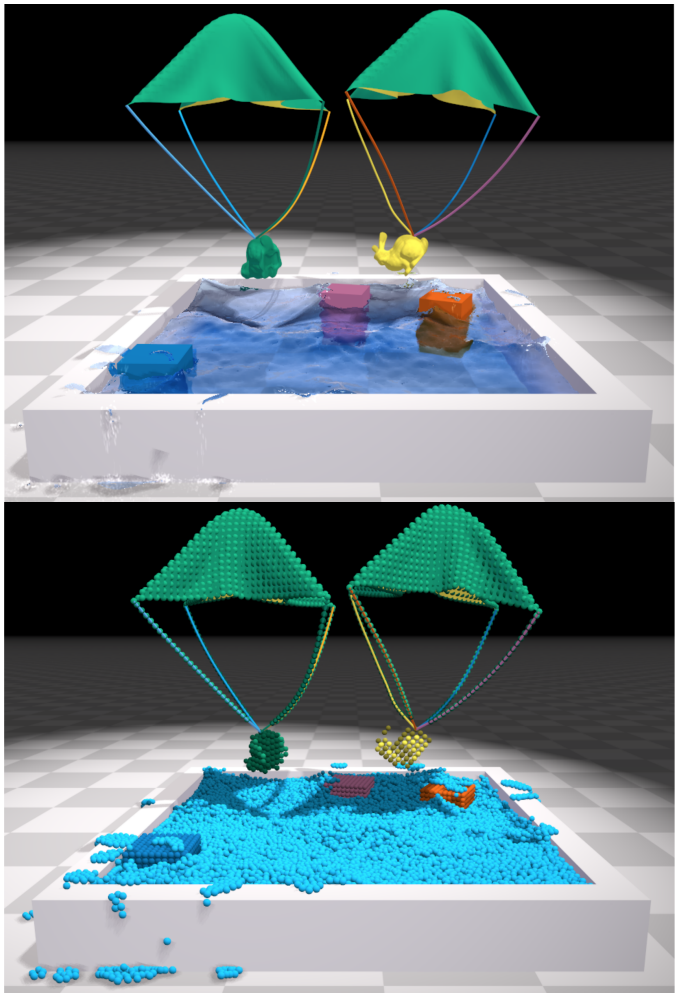

Fig. 1. The NVIDIA FleX physics simulation environment [12] [1] models fluids, cloth, and solids as collections of particles, using efficient position-based constraint solvers to simulate collisions and interactions in a unified way.

With the advent of powerful, fast simulation environments for fluids and other deformable objects such as the particle-based NVIDIA FleX [12] [1], it becomes feasible to consider applying reinforcement learning to solve soft-body manipulation problems in simulation. In this work, we

- construct two deformable object environments in the NVIDIA FleX physics simulator
- propose a variety of state space formulations in order to

make the particle-based environment tractable
- evaluate the learning performance of modern reinforcement learning algorithms using these state space formulations on our deformable object environments.

A. Background

Reinforcement learning is typically formulated as a Markov Decision Process (MDP), represented by the tuple (S, A, P, R). At each time step t, the agent acquires an observation of the current state $s_t \in S$ and chooses an action $a_t \in A$ following a policy $\pi(a_t|s_t)$. The environment then moves onto the new state s_{t+1} sampled from the transition distribution $P(s_{t+1}|s_t, a_t)$ and provides a scalar reward $r(a_t|s_t)$ to the agent. The goal of a reinforcement learning agent is to learn an optimal policy that maximizes the expected cumulative discounted reward $R_t = \sum_t \gamma^t r_t$. States can be coordinates, RGB camera inputs or screen images, or vectors of joint velocities or angles i.e. the current pose of a robot arm. Actions can be desired driving directions, button presses or inputs, or desired motor torques.

A policy π in RL is a mapping from states to actions. An intuitive way to optimize policies is to use gradient descent along with Policy Gradient (PG). Classical PG methods uses likelihood ratio method by sampling returns from interactions with environment. Actor Critic methods learns both policy (Actor) and state value function (Critic) at the same time. The critic updates the value function and the actor improves the policy in the direction suggested by the critic.

Trust region methods like Trust Region Policy Optimization (TRPO) constrain the magnitude of each policy update with a constraint on the KL divergence, thus allowing monotonic improvement in expected reward and preventing catastrophic gradient steps due to the often high-variance nature of reinforcement learning environments. Proximal Policy Optimization (PPO) combines a likelihood ratio (in order to enable multiple rounds of policy optimization on a single batch of sampled transitions via importance sampling) with loss clipping to prevent large changes in the policy and approximate a KL divergence constraint.

Currently, we use PPO as our optimization algorithm of choice due to its strong performance on a variety of reinforcement learning environments as well as its common usage in the modern deep reinforcement learning paradigm.

II. Related Works

There has been work tackling various challenges of robotic control in deformable environments both in and out of simulation.

A. Modeling deformable objects

Modeling deformable objects in the real world to improve manipulation and navigation is one promising approach. Work has been done in improving shape estimation by jointly tracking a deformable object and updating its model [8]. While physical simulations of deformable objects are expensive to evaluate online, path planning with a preprocessing approximation has been shown to reduce computational cost [4]. Likewise, these approaches for estimation with preprocessing simulations can be applied to motion planning for manipulation [7]. Furthermore, a probabilistic approach has been shown to be effective in accounting for the interactions of such deformable objects with a robot's sensors in a navigation environment [5]. Interaction with deformable objects using a manipulator to estimate its elasticity and other physical properties with a point-cloud approach can also assist such motion-planning [6]. Finally, optimal control methods coupled with an uncertainty-based motion planner based on a generic physical simulator have been applied in extremely deformable environments (such as in a medical context) [15].

Multi-armed bandits have been used to choose a model of the deformable object with which to compute a grasp and switch between models on-the-fly [13]. Furthermore, work has been done in working with deformable objects without any physics simulation, such as visually tracking deformable objects [3], modeling deformable object and gripper interactions with improved stretching constraints [18], representing deformable objects with voxel grids with added physical properties such as deformability and sensitivity [16], and leveraging particular physical properties of deformable objects as well as other constraints to prevent over-stretching [2].

In our work, we focus exclusively on simulated deformable objects. We assume a model of a given deformable object in simulation and explore representations of such a model, rather than the problem of modeling a deformable object in the real world.

B. Deformable objects in simulation

Work has been done for developing differentiable simulations to aid in training in simulation. Differentiable simulations are designed to enable gradient based optimization for optimal control and direct motion planning. Hu *et al.* develop a differentiable material point method simulation for modeling soft body robots [9]. Schenck *et al.* differentiate SPH fluid simulation for use in deep learning [19]. However, these methods have overhead for maintaining state over the course of a whole trajectory which can be intensive in both memory and computation. By the nature of the direct differentiation of the simulation, the environments are limited to the specific cases that were tested in the papers.

Mrowca *et al.* [14] apply hierarchical neural networks to predict the future physical state of deformable objects in simulation by decomposing objects into particle-based graph representations. Furthermore, Li *et al.* show that such a learned particle-based simulator can provide a helpful inductive bias in learning deformable object manipulation [11].

Despite these recent advances in differentiable physics simulators and neural methods for modeling deformable objects, the problem of simulating deformable objects is far from solved. We intend to test modern RL algorithms in a gradient free simulation environment in order to leverage the breadth of environments that can be produced in simulation. Furthermore,

we propose that restructuring of the state space will provide benefits to learning algorithms without the need for gradient optimization on the physics simulation environment. In other words, our approach considers a fixed, given simulation environment and aims to improve learning by proposing better representations for deformable objects rather than optimizing the realism of a simulator or using a learned simulator to assist task learning.

C. Reinforcement learning

Reinforcement learning has been applied to manipulating soft objects in simulation environments. Matas *et al.* [10] trained a robotic arm agent in Pybullet using multiple RL algorithms to perform 3 cloth manipulation tasks: draping a small towel over a hanger diagonal folding of cloth, and folding a towel to a tape mark. The environment exposes both RGB observation of the scene and a low-dimensional state and actor input such as joint angles and gripper position and velocity. The agent is able to achieve a high success rate in simulation and a somewhat lower rate in real world experiments.

Very recently, Wu *et al.* [20] have shown that separating policies for deformable object manipulation into a "pick" phase and a "place" phase can significantly speed up policy learning. By conditioning the "place" policy on the final state of the "pick" policy, the problem of training a single policy can be separated into that of training two policies. Moreover, by training the "place" policy using random initial "pick" locations and choosing "pick" locations with the highest chance of success under the learned "place" policy during evaluation, Wu *et al.* simplify the problem even further. With the addition of a discretization of the target deformable objects to facilitate the "pick" phase and domain randomization, Wu *et al.* learn cloth and rope manipulation policies directly from RGB camera input.

Though we do not use high-dimensional RGB camera input or novel policy network architectures, we propose several intermediate state representations for reinforcement learning in deformable object environments which provide a stronger inductive bias for learning than simple scene input and demonstrate their learning performance on two deformable object tasks.

III. Method

We propose two training environments that are soft body variations of classical environments in order to isolate the effects of the deformable object state space. Since NVIDIA FleX represents every object as a collection of points with a mass and radius, the whole state space for any FleX environment can therefore be defined as follows:

$$\mathcal{S} \in \mathbb{R}^{N \times p}$$

where N is the number of particles in the environment and p is the number of components to particle. Each particle contains information about position, velocity and mass.

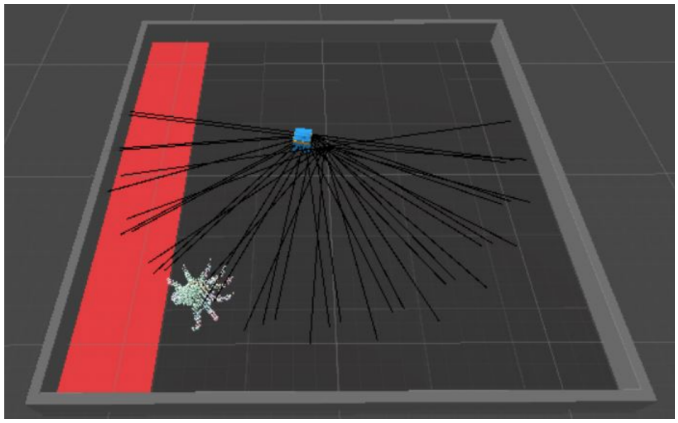

Fig. 2. Raycasting state representation. The observation is composed of the distance to the n closest objects as well as their object id along n predefined directions.

Thus, this state space is extremely high-dimensional and requires methods for significantly reducing the dimensionality of the state space. One way to do this is to use the average of the positions for each larger entity of the scene.

$$\mathcal{O} = \frac{1}{N}\sum_{i=1}^{N} p_i, \quad p_i \in \mathbb{R}^{3+3+1}$$

where $\mathcal{O}$ is the observation space for the agent. Assuming we are operating in $\mathbb{R}^3$, we have that position and velocity are both 3 dimensions and mass adds one more. Perhaps the best method for reduction, this provides the simplest computation as well. However, this also reduces the state space to the point that very little information of the shape or configuration of deformable objects or fluids is being passed to the agent.

A middle ground between representing a scene as the concatenation of all the relevant particles and representing a scene as the mean of particle positions, velocities, etc. within each object is to sample a number of particles at the beginning of an episode and track them throughout the episode, reporting the positions and velocity of those particles only. This has the benefit of largely capturing the state of a scene without the exploding dimensionality of considering all the particles in any given scene. However, a great deal of nuance in the scene is still lost, as a small number of sampled particles is unlikely to capture anything more than the general positions and velocities of key objects in the scene.

Finally, we introduce an agent-centric state representation suitable for tasks involving an embodied agent interacting with a deformable environment. Instead of randomly sampling points from the scene, we employ a raycasting approach where we report the distance and nature of to the closest object at several predetermined directions from the agent (see Figure 2).

This state representation has the advantage that it captures information more relevant to the agent's immediate surroundings than randomly sampling particles in the scene. Additionally, we can approximate this representation with

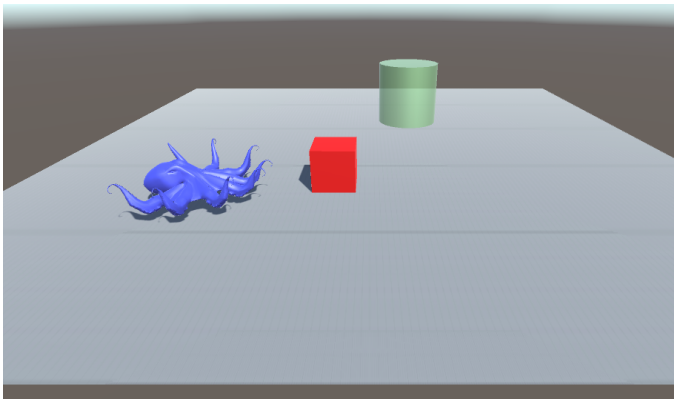

Fig. 3. Simple box pushing example where the target is replaced with a elastic octopus that deforms on contact. The objective of the environment is for the agent (red box) to push the octopus to the goal location (green cylinder). The difficulty in this environment stems from the malleability and complex dynamics of the jelly-like octopus model.

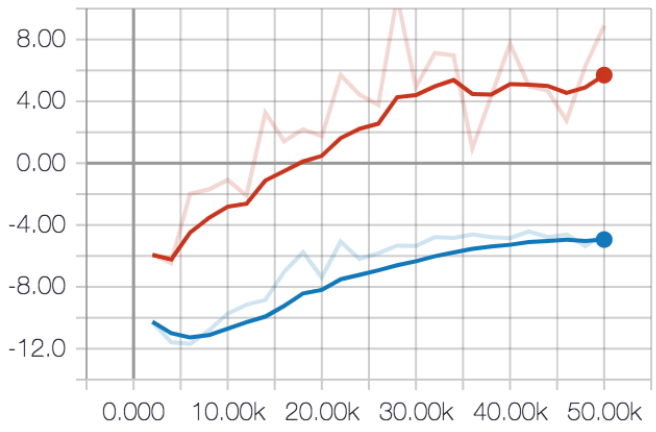

Fig. 4. Smoothed PPO training curves on sparse (dark blue) and dense (red) versions of octopus box pushing environment. The x-axis is the number of optimizer steps and the y-axis is the average cumulative reward per episode over the past 20 training episodes. Unsmoothed graphs are also shown.

common depth sensors in the real world, such as a Kinect or a LIDAR sensor, without needing any oracle information such as the precise position or velocity of particles in the scene given by the simulation environment. However, the agent must now deal with the partial observability of the environment, as this state representation captures a limited window of information, relative to the agent's current position.

IV. Results

A. OctoPush

In our first task, the goal is for an agent (a box) to push a soft octopus model to a target location. Figure 3 shows one possible configuration of the test environment.

In this environment, the state space is a 70-dimensional vector consisting of (1) the (x, y, z) positions and masses of the 8 agent cube particles, (2) the (x, y, z) position and mass of the octopus, and (3) the (x, y) position of the goal location. The positions of the agent cube and the octopus are computed by taking the mean of all the constituent particles of each object. The action is a 2-dimensional vector consisting of an (x, y) impulse to apply to the agent box at the current time step. The reward function is defined as +10 for reaching the goal, -5 for either pushing the octopus out-of-bounds or the agent cube going out-of-bounds, and -0.01 reward every timestep otherwise.

Beyond testing the validity of our setup, this environment allows us to test the ability of an agent to handle basic deformable object physics. For example, the octopus absorbs some of the force applied by the agent cube, requiring repeated impacts between the cube and the octopus to push the octopus. In contrast, rigid box-pushing environments have much simpler dynamics where the agent can easily push the box to the goal with a minimum of resistance or unpredictability.

However, we find that in this environment the agent does not learn to push the octopus to the goal but rather simply learns to go out-of-bounds as quickly as possible and achieve the -5 out-of-bounds reward (see dark blue curve in 4).

To fix this, we add a new version of this environment, which we refer to the "dense" version (versus the "sparse" version described above). In the "dense" version, we also apply a rudimentary form of reward shaping in this environment by giving the agent a positive reward proportional to the velocity of the octopus towards the goal, i.e.

$$\max(\langle goal\_position - target\_position, target\_velocity \rangle, 0)$$

where $\langle u, v \rangle$ denotes the standard $\mathbb{R}^2$ dot product.

We find that the dense version of the environment is significantly easier to solve than the sparse version. Although the cumulative reward of the dense version of the environment will be higher than that of the sparse version due to the added reward to guide learning, we find that the agent is able to successfully learn to push the octopus to the goal in the dense environment. This contrast is likely due to the low amount of learning signal provided to the agent in the sparse environment, as this makes it difficult for the agent to discover positive reward by random (or effectively random) exploration.

We then experiment with state space formulations enabled by the unified particle-based properties of the FleX simulator in the above deformable octopus pushing environment. To begin with, we found that we were unable to achieve learning performance using a point-cloud state representation where the state is the positions of each particle in the goal and target objects in the rigid box pushing environment. In this environment, the state was very high-dimensional and contained a large amount of repeated or irrelevant information, making learning of a mapping from state to action more rather than less difficult. In order to ameliorate this issue, we instead sample a few particles from each object at the beginning of the episode and construct the state from the positions of each sampled particle throughout the episode. In experiments involving this random sampled state, we find that our agent is still capable of learning and solving the environment.

Fig. 5. Ablation test of size of observation on learning performance. Top: Training reward curves, one set of hyperparameters per curve. Bottom: Max reward obtained by models as a function of state space dimension, log scale.

Furthermore, we test the effect of the size of the observation space on agent learning by constructing an ablation test where we randomly sample different number of particles and repeat training with the same set of hyperparameters. The results of this test are shown in Figure 5.

We find that performance is relatively similar up until a certain point ($n = 64$ particles is the last number of particles able to show learning performance). Visualizations of the learned policy show that policies that perform well nudge the octopus gently and repeatedly, likely to reap more of the dense reward, while policies that perform poorly simply push the cube out of bounds as quickly as possible. A smaller and more tractable state space is therefore able to engender learning policies that are able to meaningfully interact with the environment. On the other hand, state spaces that are too large overwhelm the agent and its policy network with too much information, a great deal of which is redundant in this relatively simple environment. All else being equal, a smaller state space helps learning in this simple environment.

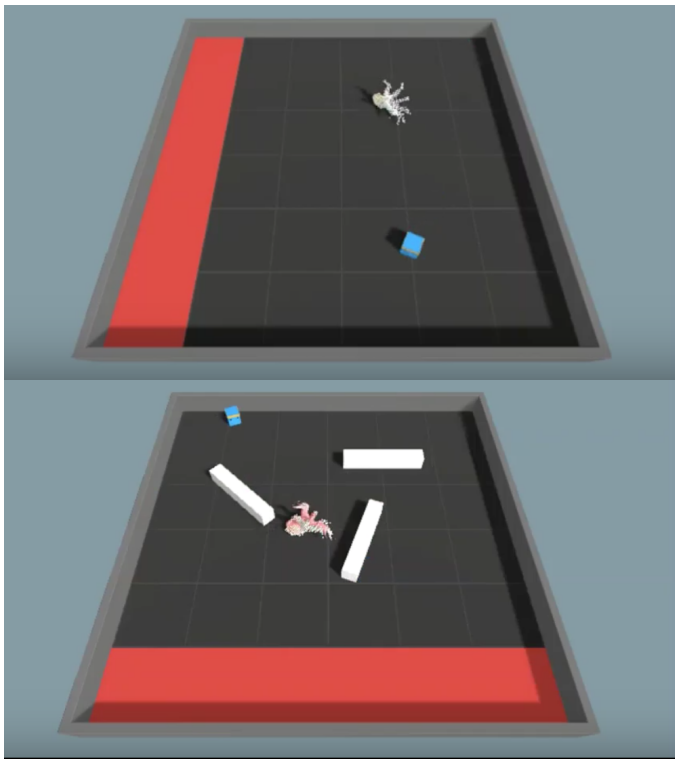

Fig. 6. PushBlock environment, with and without fixed obstacles. The agent (blue block) must push the octopus to the goal, indicated by a red boundary of the field.

To summarize, we suspect that it is indeed the high dimensionality of state representations that makes training difficult on our octopus pushing environment. Additionally, we find that the sparsity of the reward plays a large part in whether the agent is able to solve the environment.

B. *PushBlock*

We then go on to explore the possibility of a more structured observation in deformable object environments by introducing a similar octopus pushing task. In this environment, the observation is a 65-dimensional vector corresponding to the (x, y, z) position of the closest particle in 13 different predefined directions, the distance to that particle, and the id of the object containing that particle. The action space in this environment is 6-dimensional: apply a fixed impulse in one of 4 directions, or rotate the agent left or right to re-position the raycasting sensors. The reward structure is similar to before: +5 for pushing the octopus to a goal (a randomly chosen boundary of the environment) and -0.001 reward otherwise.

We first test this environment with both a rigid box target and a deformable octopus mesh to be pushed to the goal position. We then add complexity by adding fixed walls to the environment. These obstacles require the agent to navigate between them in order to push the octopus to the goal and make it impossible for the agent to directly go to the octopus and push it into the goal. Additionally, these obstacles block line-of-sight of the agent block to the octopus, requiring the agent to first identify where the octopus is in order to

Fig. 7. Training curves for four PushBlock environment configurations: rigid block, octopus without obstacles, octopus with one obstacle, and octopus with three obstacles.

successfully push it into the goal. Figure 6 shows the octopus pushing setup and the octopus pushing setup with 3 obstacles.

In this environment, we find that agents trained using our raycasting observation are able to achieve learning performance in four different configurations of the environment. Figure 7 shows learning curves for each configuration. Visualizations of the trained policies show that in environments without obstacles, the agent moves directly towards the target octopus and applies a strong impulse to throw it into the goal. In environments with obstacles, the agent adopts a "searching" policy exemplified by circling the playing field in a clockwise pattern several times, before identifying the target octopus and quickly pushing the octopus into the goal. Notably, the agent rotates itself as it circles around, generally facing in the direction of the octopus.

The agent adapted itself to the nature of the raycast observation in this environment and learned to use its behavior to gather more relevant information about the current state of the environment. This indicates that structured state representations that have a strong correspondence between agent actions and changing observations can encourage an agent to learn policies that implicitly gather requisite information about deformable object enviornments. Thus, rather than seeking to completely represent a soft-body environment, a partial representation of the state to facilitate the agent itself exploring and learning more about the current state of the environment could be a promising approach to solving the problem of representing deformable object environments.

V. Ethics

The primary ethical considerations of our work with soft-body manipulation is that the techniques we develop will eventually be applied to interact with humans, animals, and other living things. Better representations for soft objects such as cloth, fluids, and deformable meshes could be applied to facilitate physical interactions with humans, such as in routine daily life tasks like shaking hands or giving hugs. However, our work could also be used to assist the elderly or infirm by representing the state of a human body in the environment. Most importantly, work in robotic control in deformable environments can be applied to medical and surgical robotics. For example, our representations could be used to capture the state of a key organ or body tissue during surgery. In these settings, it is of the utmost importance that accountability and regularity are established. Additionally, when working with people, we aim to remove the possibility of harm or at least minimize the potential for harm. Therefore, we must consider methods to ensure that our learned control policies behave as expected or within a certain tolerance of error. There should be a way to ensure that behavior policies work as expected and account for bias introduced by the simulator. It is impossible to completely avoid harm, but what level of risk is acceptable?

There is no guarantee that the soft-body physics in simulation will match real-world soft-body physics, especially with the additional variables and unpredictability introduced when working with living things. As we discovered in building learning environments, physics simulators are far from perfect. Specifically, the regularity of physics simulators is often lacking, leading to numerous artifacts due to numerical instability and other known and unknown factors. When using simulators in a learning pipeline to learn real-world control policies, such irregularities must be accounted for. Note that sim-to-real transfer techniques like domain randomization do not solve this problem, since those techniques deal with bridging the gap between the dynamics of the simulator and the real world rather than the occasional failure of the simulated dynamics. Also, online models of deformable objects could still fail, leading to failure or unpredictable behavior.

Furthermore, living things are not static fixtures of the environment. Humans and animals are independent agents with their own hidden state, intents, and actions. To further complicate things, the actions and goals of humans are affected by the actions of the robotic agent. Work from the fields of multi-agent systems and human-robot interaction involving modeling and accounting for the actions of other agents to cooperatively carry out various tasks and behaviors could be applied to allow robots to intelligently interact with humans.

Finally, deep methods have notoriously poor accountability and are very opaque. It is difficult to reason about the output of high-dimensional neural networks, making it difficult to create guarantees regarding deep control policies. Before such control policies can be deployed in the real world, especially in high-stakes tasks like assistive robotics and medical robotics, a method to provide guarantees regarding the behavior of these robots and the well-being of the humans they interact with must be established. There has been work in provable neural networks, interpretability in deep learning, and more broadly safety in robotics and cyber-physical systems which could be used to ensure that humans do not come to harm at the hands of deep networks.

VI. Conclusion and Future Work

Particle based physics simulation environments enable the exploration of particle-based state representations for deep reinforcement learning algorithms. Such particle-based state representations have analogs in the real world, such as point clouds and depth maps given by different types of sensors. However, these representations are very high-dimensional,

making it impossible to apply deep reinforcement learning algorithms directly to particle-based input. Furthermore, while deformable objects can be modeled as part of a unified particle-based physics simulation environment, still do not admit an easy representation both in and out of simulation.

We propose and examine several intermediate state representations for these particle-based settings including the average position of all particles comprising an object, a randomly sampled subset of particles, and an agent-centric depth-sensitive representation in addition to the naive representation including all points. By testing these representations on two deformable object environments involving pushing a deformable target mesh object to a goal position, we show that these representations are suitable candidates to allow application of reinforcement learning algorithms to deformable object environments. Furthermore, these representations enable learning policies that meaningfully interact with deformable objects and implicitly encourage exploration and navigation of partially observable environments.

Future work may include applying these representations to more complex tasks in simulation, such as robotic manipulation of soft-body objects like meshes, cloth, or rope. However, the instability and occasional unpredictability of the simulation environment must be overcome, as the particle-based nature of the NVIDIA FleX environment leads to numerical issues when meshes are compressed by a gripper. Also, representing fluids is still an open problem due to the extremely complex configuration space and complex dynamics, though particle-based simulation environments offer one route to do so. However, in the real world, there is significant difficulty in even detecting the presence and state of fluids, let alone representing fluids. Finally, it would be interesting to explore neural network architectures specifically adapted to point clouds. For example, PointNet [17] can be adapted to several tasks on point clouds, such as classification, segmentation, and scene understnading, and may also improve performance in robotic control settings.

Acknowledgments

Thanks to David Held and Xingyu Lin for originally proposing the idea of a suite of reinforcement learning environments that could conform to the OpenAI Gym API, Brendan Miller for helpful discussions on PointNet, and Adithya Murali for feedback on the midterm report.

References

[1] Jan Bender Bender, Matthias Müller, and Miles Macklin. Position-Based Simulation Methods in Computer Graphics. *Eurographics Tutorials*, 2015.

[2] Dmitry Berenson. Manipulation of Deformable Objects Without Modeling and Simulating Deformation . *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[3] Cheng Chi and Dmitry Berenson. Occlusion-robust Deformable Object Tracking without Physics Simulation . *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019.

[4] Barbara Frank, Markus Becker, Cyrill Stachniss, Wolfram Burgard Burgard, and Matthias Teschner. Efficient path planning for mobile robots in environments with deformable objects. *IEEE International Conference on Robotics and Automation*, 2008.

[5] Barbara Frank, Cyrill Stachniss, Rudiger Schmedding, Matthias Teschner, and Wolfram Burgard. Real-world robot navigation amongst deformable obstacles. *IEEE International Conference on Robotics and Automation*, 2009.

[6] Barbara Frank, Rüdiger Schmedding Schmedding, Cyrill Stachniss, Matthias Teschner, and Wolfram Burgard. Learning the elasticity parameters of deformable objects with a manipulation robot. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[7] Barbara Frank, Cyrill Stachniss, Nichola Abdo, and Wolfram Burgard. Efficient motion planning for manipulation robots in environments with deformable objects. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

[8] Tao Han, Xuan Zhao, Peigen Sun, and Jia Pan. Robust shape estimation for 3d deformable object manipulation. *arXiv preprint arXiv:1809.09802*, 2018.

[9] Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6265–6271. IEEE, 2019.

[10] Matas Jan, James Stephen, and J. Davison Andrew. Sim-to-real reinforcement learning for deformable object manipulation. In *CoRL*, 2018.

[11] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B. Tenenbaum, and Antonio Torralba. Learning Particle Dynamics for Manipulating Rigid Bodies, Deformable Objects, and Fluids. *International Conference on Learning Representations*, 2019.

[12] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified Particle Physics for Real-Time Applications. *ACM Transactions on Graphics*, 2014.

[13] Dale McConachie and Dmitry Berenson. Estimating Model Utility for Deformable Object Manipulation Using Multiarmed Bandit Methods. *IEEE Transactions on Automation Science and Engineering*, 2018.

[14] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li Fei-Fei, Joshua B. Tenenbaum, and Daniel L.K. Yamins. Flexible Neural Representation for Physics Prediction. *Neural Information Processing Systems*, 2018.

[15] Sachin Patil, Jur van den, and Berg Ron Alterovitz. Motion planning under uncertainty in highly deformable environments. *Robotics: Science and Systems*, 2011.

[16] Calder Phillips-Grafin and Dmitry Berenson. A Representation Of Deformable Objects For Motion Planning With No Physical Simulation . *IEEE International Conference on Robotics and Automation*, 2014.

[17] Charles Qi, Hao Su, Kaichun Mo, and Leonidas Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CVPR*, 2017.

[18] Mengyao Ruan, Dale McConachie, and Dmitry Berenson. Accounting for Directional Rigidity and Constraints in Control for Manipulation of Deformable Objects without Physical Simulation . *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018.

[19] Connor Schenck and Dieter Fox. Spnets: Differentiable fluid dynamics for deep neural networks. *arXiv preprint arXiv:1806.06094*, 2018.

[20] Yilin Wu, Wilson Yan, Thanard Kurutach, Lerrel Pinto, and Pieter Abbeel. Learning to manipulate deformable objects without demonstrations. 2019.