Estimating Mass Distribution of Articulated Objects through Non-prehensile Manipulation

K. Niranjan Kumar
Georgia Institute of Technology

Irfan Essa
Georgia Institute of Technology

C. Karen Liu
Stanford University

***Abstract*— We explore the problem of estimating the mass distribution of an articulated object by an interactive robotic agent. Our method predicts the mass distribution of an object by using limited sensing and actuating capabilities of a robotic agent during an interaction with the object. Inspired by the role of exploratory play in human infants, we take the combined approach of supervised and reinforcement learning to train an agent such that it learns to strategically interact with the object for estimating its mass distribution. Our method consists of two neural networks: (i) the policy network which decides how to interact with the object, and (ii) the predictor network that estimates the mass distribution given a history of observations and interactions. Using our method, we train a robotic arm to estimate the mass distribution of an object with moving parts (e.g. an articulated rigid body system) by pushing it on a surface with unknown friction properties. We show the robustness of our method across different physics simulators and robotic platforms. We further test our method on a real robot platform with 3D printed articulated chains with varying mass distributions. We present results that demonstrate that our method significantly outperforms the baseline agent that uses random pushes to interact with the object.**

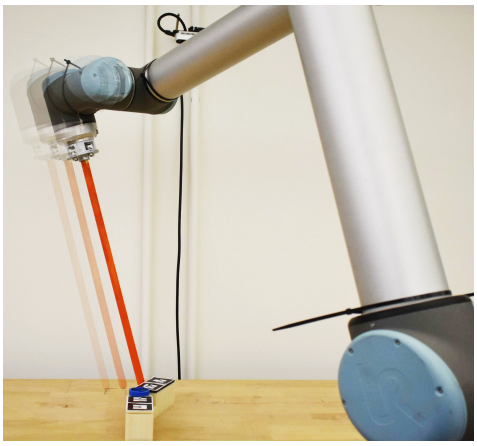

Fig. 1. Example of a push (in two frame time-lapse) that can be executed by the robot on a 2-link chain (Figure 3). The chain loses contact with the end-effector (painted orange in the picture) and continues sliding on the surface briefly before coming to rest.

I. INTRODUCTION

Humans continuously make physical inferences while perceiving and interacting with the world around them [1]. Relying on our physical intuition, we can predict with a high degree of certainty if a block tower would destabilize and fall [2] by simply looking at it. While visual observation is sufficient to predict a rough outcome of the future, having the ability to physically interact with the surroundings provides more critical information to reason about the world. Infants, for example, engage in exploratory play to discover non-obvious physical properties of objects [3], and develop a core knowledge in long-term cognition about the physical world [4]. Can we teach the robot to engage in exploratory play with a new object in order to learn more about it?

In this work, we develop an interactive agent capable of estimating the mass distribution of an articulated object by pushing it around on a surface. We design a simulated world, shown in Figure 3, and a real robot setup, shown in Figure 1, where a robotic arm can interact with an articulated chain by pushing it at any point on any segment. While estimating mass properties through physical interaction has been previously studied, our problem is uniquely challenging in three aspects. First, to make our method applicable to the real-world hardware, we do not expect the robot to be able to push or track the state of a moving object. Instead, our method is only allowed to use the information acquired when the object comes to a complete stop. Second, unlike estimating the mass of a single rigid body, the Euclidean distance between two equilibrium states of an articulated object is insufficient to inference the mass of each moving segment. Compounding with the discontinuous behaviors due to self-collisions among segments, estimating the mass distribution of an articulated object accurately requires the robot to push the object at the right spots, with the right force, and make the right conclusions about the outcomes of its interactions. Third, we do not require the agent to grasp the object during any part of the interaction and allow only non-prehensile pushing of the object.

We propose a dual network architecture that consists of (i) a policy network and (ii) a predictor network working in tandem towards the goal. The predictor network observes the reaction of the object to pushes imparted by the robot, and attempts to predict the mass distribution of the object at the end of the episode. The policy network learns how to push the object so that useful information can be extracted from each push. The predictor network learns from a set of state-action trajectories simulated using articulated objects with various mass distributions and friction coefficients. Once learned, the combination of the policy and the predictor networks can reliably estimate the mass distribution of an unseen object with arbitrary segment shapes and friction coefficient under moderate sensing and actuating errors,

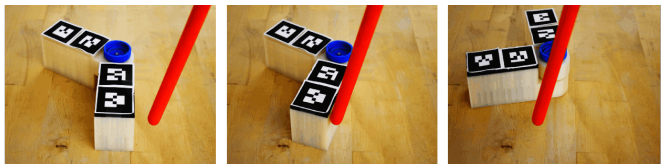

Fig. 2. Sequence of pictures showing dynamic manipulation of a two-link chain by the robot end-effector

using only a few pushes.

The main contribution of our paper is to demonstrate that embodied learning of physical parameters of a system benefits from an intelligent interaction policy. We provide evidence to show that not all actions are created equal when an agent interacts with the intention to discover the physical properties of an object. By comparing to a baseline random interaction policy, we show that our agent learns to exploit the subset of informative action to significantly improve the accuracy of estimation.

II. RELATED WORK

Humans utilize intuitive physics model to reason about the world, and predict its change in the near future [5], [6]. Building such a physical model has traditionally been approached as a system identification problem [7], which has the potential to significantly improve the control policies operated in the real world. Recent advances in deep learning methods advocate a new approach towards predicting the future state of the world directly from the recent observation history [8], [9], [10], [11], [12], [13], [14], [15]. By training a mapping between past states and the future state using simulated sequential data, one can predict whether a block tower is going to fall [16], [17], [18], and the outcome of collision between two rigid bodies [11]. The predictive model can also be conditioned on external actions that could influence the next state of the system. This model can be used to find the optimal action such that it results in a state preferential to the task, such as training an agent to play billiards [19] or pushing objects to their goals [20], [21].

While directly predicting the next state is a powerful tool, some applications focus on identifying specific physical parameters useful for control algorithm analysis, or developing environment-conditioned control policies [22], [23], [24]. For example, the degrees of freedom and the range of motion of an articulated rigid body system can be identified from videos of motion [25], [26], or from a robot actively interacting with the object [27], [28], [29]. Predicting the mass of an object has been of particular interest in the subarea of robotic manipulation. [30] used a force sensor on two finger tips to estimate the mass and the center of mass of blocks. [31] estimated the mass and friction coefficients from videos. [32] learned latent representations of physical properties that correlate with mass and friction. Our work is closely related to [33], in which a robotic arm is used to push blocks for estimating their mass and friction coefficients. However, our work goes one step further by predicting the mass distribution of an object consisting

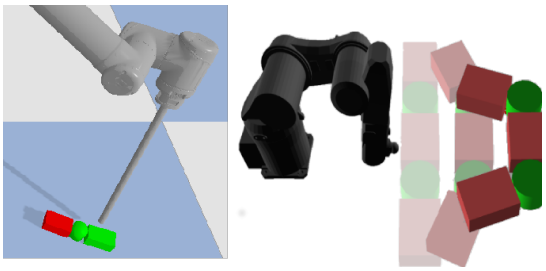

Fig. 3. We run simulations where the agent can interact with articulated objects across two different robot platforms

multiple moving parts. Using a random probing policy as proposed by the previous work, like [33] typically results in poor estimation of the mass distribution. Instead, we train a policy that strategically pushes the object to extract maximum information for estimating its mass distribution. While pushing is the primary means by which an agent gathers information about the object in our work, pushing has also been utilized in methods that extract visual information. For example, [34] introduced a method to learn useful visual representations by pushing objects . Similarly, [35] learned to push objects around to accomplish a segmentation task. For robotic applications, [36] taught a robot to de-clutter a scene using optimal pushes, while [37] focused on finding optimal pushes to move objects in a constrained setting.

III. METHOD

We introduce a method to enable a robot to estimate the mass distribution of an articulated object by strategically pushing it around on a flat surface. While this problem can be formulated in a variety of ways, our proposed method considers the practical limitations inherent to the robot's sensing and actuation capabilities–it only depends on the information the robot can reliably observe, and the commands the robot can reliably execute. To this end, our method only requires the robot to be able to observe and push the object when it is at an equilibrium state (i.e. the object comes to a complete stop), as opposed to transient states where neither accurate state estimation nor precise pushes can be easily achieved. This requirement increases the difficulty of the problem, but makes the deployment of our method to real-world hardware more feasible.

A conventional approach to this problem is to conduct system identification such that the observed data is consistent with the equations of motion. However, since we only observe the object and apply the actions at equilibrium states, we cannot utilize dynamic equations to infer masses in the absence of derivative information (e.g. velocity, acceleration). Further, effective system identification also requires analysis on persistent excitation to ensure that the observations span a wide range of system behaviors. Although the object in our problem is a passive multibody system, the mass parameters can only be uniquely identified by a subset of observation and action trajectories (see Appendix A). This problem is further exacerbated by practical scenarios

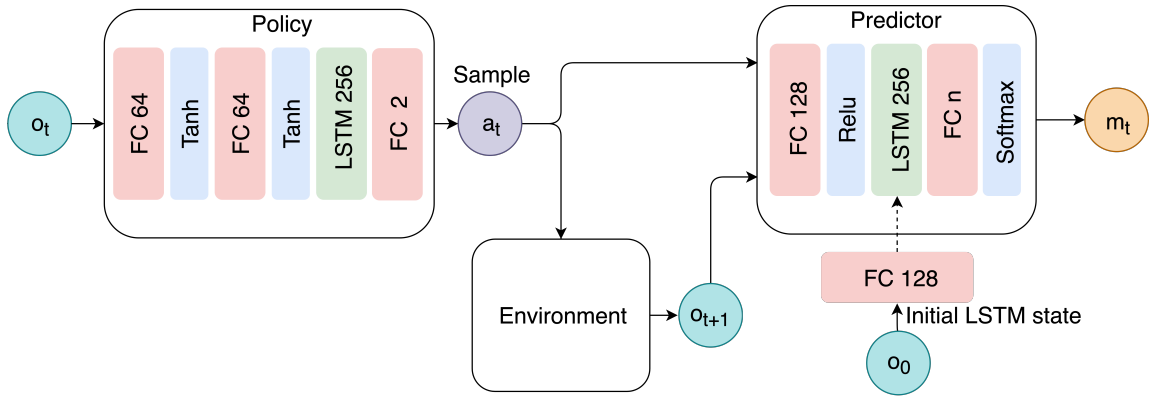

Fig. 4. Architecture of or proposed method. The Policy and Predictor networks are trained alternately. During testing, we use the Policy network to suggest actions that are likely to provide novel information about the mass distribution of the object.

in which the robot is only expected to interact with the object a few times.

We take a learning approach to circumvent the above challenges. Our method consists of two learned models: a predictor that estimates the mass distribution from observation and action history, as well as a policy that determines the appropriate push actions to induce useful information for predicting mass distributions. It should be noted that we predict the mass distribution of the object, instead of the actual mass of each segment in the object. Since the kinetic friction force applied on each segment is proportional to the mass of the segment when it lies flat on the surface, the ratio of masses among segments is unaffected by the friction coefficient. Therefore, if we only predict the mass distribution and assume the total mass is known, our method is agnostic to the surface materials of the object.

A. Predicting Mass Distribution from Interaction

Consider an articulated chain with n rigid bodies of mass connected by revolute joints. Let $\boldsymbol{o}$ be the observed configuration of the object, and $\boldsymbol{a}$ represent the push the robot exerts on the object. Starting from the initial observation $\boldsymbol{o}_0$, the robot applies a push $\boldsymbol{a}_0$ and observes the object when it comes to a complete stop, $\boldsymbol{o}_1$. After repeating this process K times, the robot infers the mass distribution $\boldsymbol{m} \in \mathbf{R}^n$ of the object, where each element of $\boldsymbol{m}$ indicates the mass of a rigid body normalized by the total mass of the object.

With this problem setting, we learn a predictor f_μ using supervised learning. The predictor takes as input the observation and action history $\boldsymbol{h}_t = (\boldsymbol{o}_0, \boldsymbol{a}_0, \cdots, \boldsymbol{a}_{t-1}, \boldsymbol{o}_t)$, the current action $\boldsymbol{a}_t$, and the next observation at the equilibrium $\boldsymbol{o}_{t+1}$, to predicts the mass distribution $\boldsymbol{m}_t$ at the current step t:

$$\boldsymbol{m}_t = f_\mu(\boldsymbol{h}_t, \boldsymbol{a}_t, \boldsymbol{o}_{t+1}). \tag{1}$$

In practice, we represent f_μ as a Long Short-Term Memory(LSTM) network (Figure 4) parameterized by $\boldsymbol{\mu}$. We define a loss function as follows:

$$\mathcal{L}(\theta) = \frac{1}{N}\frac{1}{K}\sum_{i=1}^{N}\sum_{t=0}^{K} \|\boldsymbol{m}^i - f_\mu(\boldsymbol{h}_t, \boldsymbol{a}_t^i, \boldsymbol{o}_{t+1}^i)\|_2, \tag{2}$$

where superscript i is the index of the N training samples. Theoretically, we only need the last prediction to be accurate, i.e. , $f_\mu(\boldsymbol{h}_K, \boldsymbol{a}_K^i, \boldsymbol{o}_{K+1}^i) = \boldsymbol{m}^i$. In practice, however, encouraging the earlier prediction ($t < K$) to also be accurate provides a richer reward signal for the reinforcement learning algorithm during policy training.

Minimizing Equation 2 results in a predictor network that achieves high accuracy when the input trajectory contains sufficient information to identify the mass distribution, i.e. there is a unique mass distribution consistent with the observations induced by given actions. However, as discussed earlier, some input trajectories in Equation 1 cannot uniquely identify the mass distribution due to the singularity in the dynamic system, and sparse observations and actions. As a result, the predictor performs poorly on predicting the mass distributions for those regions in the input space.

B. Learning How to Interact

Given a supervised model that predicts the mass given observation and action trajectories, what should those trajectories be to get an accurate mass distribution estimate? More specifically, we need to decide what actions to take at each time step so that the input trajectory to the predictor lies in the region of high accuracy. We model this problem as a standard Markov Decision Process (MDP) with two modifications. First, a state contains all the historical observations and actions up to the current step, not just the current observation. This is because, intuitively, the nature of exploration requires memorizing what has been done and observed in the past. Second, the reward function evaluates the actions indirectly through the predictor network, such that the policy learns to only produce trajectories that the predictor is able to estimate accurately.

We define the fully observed state to be $\boldsymbol{s}_t = \{\boldsymbol{o}_{0:t}, \boldsymbol{a}_{0:t-1}\}$, which contains the observation history and the actions that produce it. The reward function can then be defined as a function of the current action $\boldsymbol{a}_t$ and the next state $\boldsymbol{s}_{t+1}$:

$$r(\boldsymbol{a}_t, \boldsymbol{s}_{t+1}) = 1 - \frac{2}{(M_{\max} - M_{\min})}\|\bar{\boldsymbol{m}} - f_\mu(\boldsymbol{h}_t, \boldsymbol{a}_t, \boldsymbol{o}_{t+1})\|^2, \tag{3}$$

where $\bar{\boldsymbol{m}}$ is the ground truth mass distribution, and $[M_{min}$,$M_{max}]$ represents the domain from which the mass vectors in the training set are sampled. We design the reward function to be inversely proportional to the error in mass estimated by the predictor network, and scale its output to be between $[-1, 1]$. Intuitively, learning a policy can be seen as adapting the input data distribution to operate on low error regions of the predictor. We use a policy gradient method, Proximal Policy Optimization [38] to solve for the policy $\pi_\theta(\boldsymbol{a}|\boldsymbol{s})$, where π is represented as a LSTM neural network parameterized by θ.

C. Alternating Training Procedure

In practice, we find that alternating between training the predictor and the policy a few times results in predicting the mass distribution more accurately. We begin by training the predictor on a uniform distribution of action and observation trajectories. Subsequently, we freeze the predictor and train the policy network until convergence. Next, we retrain the predictor with the input trajectories provided by the current policy network. The alternating training procedure terminates when the accuracy of the predictor reaches a target threshold. The training process is terminated with the training of the predictor as the data distribution imposed on the predictor by the policy might have changed after the previous policy updates. Effectively, the alternating scheme gradually refines the training distribution for the predictor from a uniform distribution to a task-relevant one induced by the policy.

D. Executing Actions

We assume that the robot is capable of observing the configuration of the object directly. This assumption can be relaxed by using image input and adding convolution neural networks to the predictor and the policy, without modifying the core of the proposed framework. We further assume that the robot is able to command torques at each actuator, and that its base is able to move and rotate on the surface where the object lies. Based on these assumptions, we define the observation vector as the configuration of the object, $\boldsymbol{o} = \{x, y, \alpha, \boldsymbol{\theta}\}$, where x, y, α are the global translation and yaw angle of the root link, while $\boldsymbol{\theta}$ corresponds to the joint angles of the rest of the chain. The action vector is defined as $\boldsymbol{a} = \{a_1, a_2\}$, where $a_1 \in \mathbf{R}$ represents the target velocity of the end-effector when the robot delivers the push, and $a_2 \in \mathbf{R}$ is used to select the point of application of the push. Depending on the value of a_2 and sign of a_1, different points on different chain segments are chosen. We only allow pushes at points that lie on the same horizontal plane as the center of the chain segment and partition the 2 dimensional action vector to represent all permissible pushes. See Figure 6 for more details.

We use the operational space control [39] formalism to generate appropriate joint torques that achieve a desired linear and angular acceleration $\ddot{\boldsymbol{x}} \in \mathbf{R}^6$ at the end-effector:

$$\boldsymbol{\tau} = \boldsymbol{J}_x^T \boldsymbol{M}_x \ddot{\boldsymbol{x}} + \boldsymbol{g}(\boldsymbol{q}), \tag{4}$$

where $\boldsymbol{q}$ is the current configuration of the robot in generalized coordinates, and $\boldsymbol{J}_x = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{q}}$ is the Jacobian evaluated at the end-effector. Since the push occurs at an equilibrium state, the effect of the Coriolis force is ignored, but the effect of gravity is included as $\boldsymbol{g}(\boldsymbol{q})$. The operational space mass matrix is defined as:

$$\boldsymbol{M}_x = (\boldsymbol{J}_x \boldsymbol{M}^{-1} \boldsymbol{J}_x^T)^{-1}, \tag{5}$$

where $\boldsymbol{M}$ is the original mass matrix in the generalized coordinates.

We compute the desired acceleration $\ddot{\boldsymbol{x}}$ using a simple PD feedback rule to match the desired linear position $\boldsymbol{x}_T^{des} \in \mathbf{R}^3$. We also command the robot to orient its end-effector to align with $\boldsymbol{x}_R^{des} \in \mathbf{R}^3$. For the derivative term, the end-effector is expected to match both the desired linear and angular velocities, $\dot{\boldsymbol{x}}^{des} \in \mathbf{R}^6$. Thus, the desired end-effector acceleration can be computed as:

$$\ddot{\boldsymbol{x}} = k_p \begin{pmatrix} \boldsymbol{x}_T^{des} - \boldsymbol{x}_T \\ \boldsymbol{x}_R^{des} \ominus \boldsymbol{x}_R \end{pmatrix} + k_d(\dot{\boldsymbol{x}}^{des} - \dot{\boldsymbol{x}}), \tag{6}$$

where $\boldsymbol{x} = [\boldsymbol{x}_T, \boldsymbol{x}_R] \in \mathbf{R}^6$ denotes the end-effector position $\boldsymbol{x}_T$ and orientation $\boldsymbol{x}_R$ in the world frame. The operator $\ominus$ indicates the difference between the two 3D orientations. The stiffness k_p and damping coefficient k_d are chosen manually ($k_d = 152$ and $k_p = 3000$ in our experiments).

During testing, we first query π_θ to obtain the optimal action (a_1, a_2) for the current state. Then, we set the desired linear position in Equation 6 to $\boldsymbol{x}_T^{des} = \boldsymbol{T}\boldsymbol{p}$, where $\boldsymbol{T}$ is the transformation that transforms a vector in the object frame to the world frame, and $\boldsymbol{p} \in \mathbf{R}^6$ corresponds to the coordinates in object frame of the point represented by $\boldsymbol{a}$. The desired orientation $\boldsymbol{x}_R^{des}$ is defined such that, when $\boldsymbol{x}_R = \boldsymbol{x}_R^{des}$, the end-effector is perpendicular to the surface $\boldsymbol{p}$ lies on.

Before the robot is in contact with the object, the desired velocity in Equation 6 is set to zero, $\dot{\boldsymbol{x}}_T^{des} = \boldsymbol{0}$. When the end-effector is sufficiently close to the object and perpendicular to the surface of $\boldsymbol{p}$, we set the desired velocity to $\dot{\boldsymbol{x}}_T^{des} = \dot{\boldsymbol{p}}$ to exert the pushing force. We maintain this desired velocity for a fixed amount of time (10 time steps in our experiments). The object moves with the momentum imparted by the robot and eventually comes to rest due to the friction between the object and the surface. We execute actions in an open loop control framework and enforce safety limits on the robot torques.

IV. EXPERIMENTS

We evaluated our results on a simulated KUKA robotic arm using the physics engine, DART [40], as well as on a UR10 robot hardware in the real world.

A. Network architecture and training

The predictor network consists of fully connected and LSTM layers as shown in Figure 4. The final softmax layer transforms the output of the last fully connected layer into the output mass distribution. Every fully connected layer except the last one is followed by the ReLU non-linearity. Similarly, we model the policy with two fully connected hidden layers

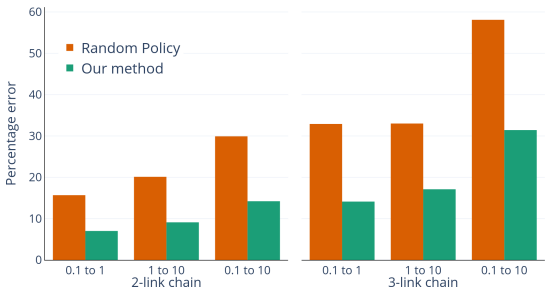

Fig. 5. Percentage error of our method compared to the random predictor baseline.

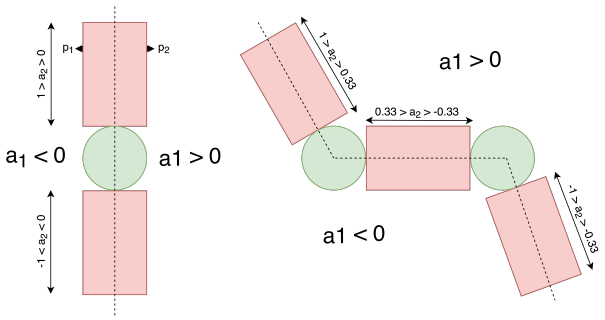

Fig. 6. We partition a two dimensional action space to represent all permissible pushes that can be executed by the robotic arm. The sign of a_1 represents which side of the segment is going to be pushed while a_2 spans the points on the selected side. For example all pushes that can be applied to the points p_1, p_2 and p_3 are represented by $\boldsymbol{a} = \{-k, 0.75\}$, $\boldsymbol{a} = \{k, 0.75\}$ and $\boldsymbol{a} = \{k, -0.75\}$ respectively, where k is a positive real number.

containing 64 neurons followed by an LSTM layer with 256 units (Figure 4).

For the initial training of the predictor, we collect $16k$ episodes for the 2-link chain, and $32k$ for the 3-link chain by running a random policy that outputs a uniformly sampled action at each step until the end of the episode. The rollouts can be collected efficiently by running 16 agents as parallel threads. We train the predictor for a total of $500k$ steps with a batch size of 16, using stochastic gradient descent. The learning rate is initially set to 0.1, and is halved every $166k$ iterations. This trained predictor is used as a part of the reward function for the subsequent training of the policy, and provides a baseline to compare our method against.

The policy is trained using Proximal Policy Optimization (PPO) [38], [41], for $31k$ iterations with a learning rate of 10^{-4}. Once we train the policy that maximizes this reward function, we freeze the policy network weights and fine-tune the predictor network for another $500k$ iterations using a fresh dataset of rollouts. It is essential to use a fresh dataset because the data distribution imposed by the neural network policy is different from the one imposed by the initial random policy. This process can be iterated multiple times. In our experiments, we find that six iterations of policy training and predictor fine-tuning are sufficient to obtain good performance converge. As the learning progresses, the improvement from each learning meta-iteration decreases and eventually becomes unnoticeable.

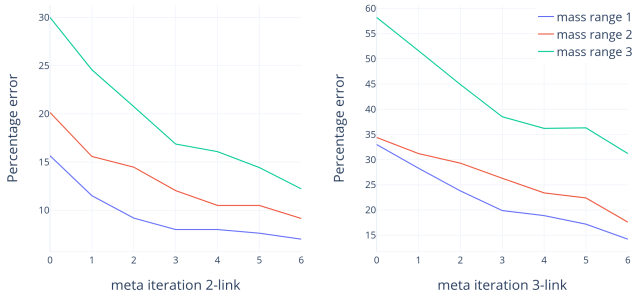

Fig. 7. Plot of percentage error vs meta iteration number. Each meta iteration represents one policy training and predictor finetuning . Mass ranges 1, 2 and 3 represent $[0.1, 1]$, $[1, 10]$ and $[0.1, 10]$ respectively

B. Evaluation in simulation

We import a model of the KUKA KR 5 Sixx R650 Robotic arm in DART and an articulated chain of rigid cuboid segments connected by revolute joints. We experiment with two types of objects: 2-link chains and 3-link chains, with the link masses uniformly sampled from three different mass ranges: $[0.1, 1]$, $[1, 10]$, and $[0.1, 10]$. A wider mass range is more challenging because the range of informative pushing forces also needs to be wider. When the robot uses a large force to push a light object, the object might rapidly rotate over many cycles. This information is lost in the training set because the observation only records the equilibrium state of the object, rather than the entire trajectory. Consequently, the training set contains examples that result in many-to-one mapping between input and output spaces.

In addition to varying the mass, we also randomize the friction coefficient from a range of $[0.5, 1]$, and the length for each segment from $[0.1, 0.15]$. We start the episode with a random configuration of the articulated object, and add Gaussian noise with zero mean and 0.01 standard deviation to the observations and actions in the environment to test the robustness of our networks.

To demonstrate the advantage of our pushing policy, we create a baseline that uses a predictor trained with actions sampled from a uniform distribution. We call this baseline "Random Policy". This approach is similar to [33] where the action taken is independent of the state of the system. Figure 5 shows the percentage error in the mass distribution predicted by the different model architectures.

We find that having a policy that strategically pushes with the intention of minimizing prediction error outperforms the random policy for every case we consider. We observe that the improvement our policy offers for wider and more difficult mass ranges is more significant than for smaller ranges. We also note that alternating between supervised training and reinforcement learning is quite stable and leads to a consistent drop in percentage error as show in Figure 7. This implies that the predictor retains its accuracy when the trajectory distribution changes abruptly by the updated policy, and does not need to be trained from scratch in each meta-iteration.

C. Evaluation on real hardware

Unlike simulation, moving the base of the real robot around to follow the object can be unrealistic. With a stationary robot, the mass ranges that can be explored become limited as the object can be pushed out of the robot workspace easily with one wrong action. We conduct experiments with a UR10 robot which has a significantly larger workspace than the KUKA robot. We make the robot interact with articulated 3D printed chains of different mass distributions. The 3D printed links have a designated "mass chamber", shown in Figure 8 (Right) where different weights can be added to vary the mass distribution of the object as necessary. We connect these links with ball-bearings to create an near friction-free revolute joint. A Realsense D435 depth camera is utilized to capture RGBD images of the scene consisting of the table, the robot and the articulated object. We calibrate the entire setup by using QR codes placed on various spots on the table with a ROS wrapper for the alvar AR tag tracking library. Further, we place QR codes on each link and track the pose of the articulated chain after every robot interaction.

To manipulate the chain, the robot has to be able to hit it at precise locations as instructed by the policy. We design an end-effector with a wooden dowel and attach it to the robot with a 3D printed mount in our experiments. To ensure that the contact with the object is at a precise point on the object, we tilt the dowel by an angle of approximately $30°$. After every interaction, the robot resets to its rest state so that the camera can get an occlusion free image of the object. The agent registers the new location of the object and performs the next interaction step as instructed by the policy.

The real-world experiments are conducted using a two-link chain with a joint range of $[-\pi/2, \pi/2]$. We first use the uniform policy to collect 30 episodes each (150 in total) of the robot interacting with 5 different mass distributions. The two-link objects are visuallly identical and can be one of the following weight pairs which span the range of mass distributions possible: $[0.16, 0.16]$, $[0.064, 0.192]$, $[0.064, 0.256]$, $[0.192, 0.064]$ and $[0.256, 0.064]$. We simulate the robot in PyBullet [42] and train a predictor on the mass range $[0.01, 0.3]$. The features learnt by the predictor are fed to a small two-layer fully connected network $[64 - Relu - 32 - Relu - 5]$, trained to classify the objects into one of the 5 classes by using a cross-entropy loss. We find that the random predictor reaches an accuracy of 40.74%, with random chance being 20%. We then collect another 150 episodes with our learned policy and train the classification network using the features from the predictor of the last meta iteration. Our method outperforms the baseline by a large margin achieving an accuracy of 81.4% as shown in Figure 8. We notice that the learned policy tries to push the object such that there is movement about the revolute joint so that inference about individual links is possible. Intuitively, this makes sense as in cases where the joint is locked down, the body behaves like a rigid object and it becomes hard to decouple properties about individual links.

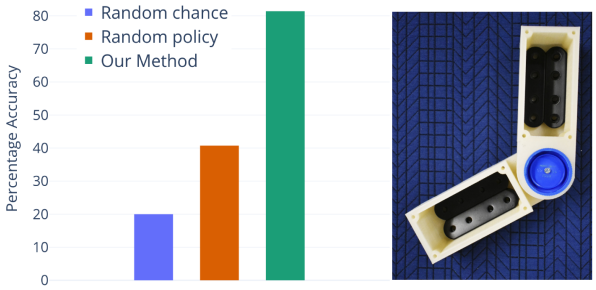

Fig. 8. Accuracy on classification of two-link chains with 5 different mass distributions (Left). We vary the effective mass of each link by adding iron braces to the mass chamber (Right)

V. DISCUSSION

In this paper we present an agent that learns to predict the mass distribution of articulated chains by strategically selecting few pushes that extract maximum information. We propose a training procedure that alternates between training a predictor with supervised learning, and a policy with reinforcement learning. We test our hypothesis in simulation by pushing 2-link and 3-link articulated chains with a robotic arm. Our experiments show that our approach consistently outperforms the baseline method across problems and various levels of difficulty and different robotic platforms. Throughout our work we made design choices that make it easier to eventually apply our algorithm on a real world robot. To this end we use a UR10 robot to test out our algorithm on a 2-link chain in the real world.

As a future direction, we intend to utilize convolutional neural networks to learn directly from RGBD images instead of using a tracker based approach. We would also like to explore other joint structures and estimate more properties like joint friction and kinematic structure of objects. We also hope to use our predictive models to plan non-prehensile manipulation of articulated objects into specific goal states with limited interaction.

REFERENCES

[1] E. S. Spelke and K. D. Kinzler, "Core knowledge," *Developmental Science*, vol. 10, no. 1, pp. 89–96, jan 2007.

[2] J. Hamrick, P. Battaglia, and J. B. Tenenbaum, "Internal physics models guide probabilistic judgments about object dynamics."

[3] D. A. Baldwin, E. M. Markman, and R. L. Melartin, "Infants' ability to draw inferences about nonobvious object properties: Evidence from exploratory play," *Child Development*, vol. 64, no. 3, pp. 711–728, 1993.

[4] P. Muentener, E. Herrig, and L. Schulz, "The efficiency of infants' exploratory play is related to longer-term cognitive development," *Frontiers in Psychology*, vol. 9, p. 635, 2018.

[5] J. R. Kubricht, K. J. Holyoak, and H. Lu, "Intuitive physics: Current research and controversies," *Trends in cognitive sciences*, vol. 21, no. 10, pp. 749–759, 2017.

[6] J. B. Hamrick, P. W. Battaglia, T. L. Griffiths, and J. B. Tenenbaum, "Inferring mass in complex scenes by mental simulation," *Cognition*, vol. 157, pp. 61–76, 2016.

[7] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive processing*, vol. 12, no. 4, pp. 319–340, 2011.

[8] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, *et al.*, "Interaction networks for learning about objects, relations and physics," in *Neural Information Processing Systems*, 2016, pp. 4509–4517.

[9] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum, "A compositional object-based approach to learning physical dynamics," *arXiv preprint arXiv:1612.00341*, 2016.

[10] S. Purushwalkam, A. Gupta, D. Kaufman, and B. Russell, "Bounce and learn: Modeling scene dynamics with real-world bounces," in *International Conference on Learning Representations*, 2019.

[11] T. Ye, X. Wang, J. Davidson, and A. Gupta, "Interpretable intuitive physics model," in *ECCV*, 2018.

[12] A. Ajay, J. Wu, N. Fazeli, M. Bauza, L. P. Kaelbling, J. B. Tenenbaum, and A. Rodriguez, "Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing," *2018 IEEE/RSJ IROS*, Oct 2018.

[13] J. Wu, E. Lu, P. Kohli, B. Freeman, and J. Tenenbaum, "Learning to see physics via visual de-animation," in *Advances in Neural Information Processing Systems*, 2017, pp. 153–164.

[14] S. Ehrhardt, A. Monszpart, N. J. Mitra, and A. Vedaldi, "Taking visual motion prediction to new heightfields," *Computer Vision and Image Understanding*, vol. 181, pp. 14–25, 2019.

[15] D. Mrowca, C. Zhuang, E. Wang, N. Haber, L. F. Fei-Fei, J. Tenenbaum, and D. L. Yamins, "Flexible neural representation for physics prediction," in *Advances in Neural Information Processing Systems*, 2018, pp. 8799–8810.

[16] P. W. Battaglia, J. B. Hamrick, and J. B. Tenenbaum, "Simulation as an engine of physical scene understanding," *Proceedings of the National Academy of Sciences*, vol. 110, no. 45, pp. 18 327–18 332, 2013.

[17] W. Li, A. Leonardis, J. Bohg, and M. Fritz, "Learning manipulation under physics constraints with visual perception," *arXiv preprint arXiv:1904.09860*, 2019.

[18] W. Li, A. Leonardis, and M. Fritz, "Visual stability prediction and its application to manipulation," in *2017 AAAI Spring Symposium Series*, 2017.

[19] K. Fragkiadaki, P. Agrawal, S. Levine, and J. Malik, "Learning visual predictive models of physics for playing billiards," *arXiv preprint arXiv:1511.07404*, 2015.

[20] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 2786–2793.

[21] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," in *Advances in Neural Information Processing Systems*, 2016, pp. 5074–5082.

[22] W. Yu, J. Tan, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," *arXiv preprint arXiv:1702.02453*, 2017.

[23] W. Zhou, L. Pinto, and A. Gupta, "Environment probing interaction policies," in *ICLR*, 2019.

[24] J. Yang, B. Petersen, H. Zha, and D. Faissol, "Single episode policy transfer in reinforcement learning," *arXiv preprint arXiv:1910.07719*, 2019.

[25] J. Sturm, V. Pradeep, C. Stachniss, C. Plagemann, K. Konolige, and W. Burgard, "Learning kinematic models for articulated objects," in *International Joint Conference on Artificial Intelligence*, 2009.

[26] A. Venkataraman, B. Griffin, and J. J. Corso, "Learning kinematic descriptions using spare: Simulated and physical articulated extendable dataset," *arXiv preprint arXiv:1803.11147*, 2018.

[27] O. Brock, J. Trinkle, and F. Ramos, *Learning to Manipulate Articulated Objects in Unstructured Environments Using a Grounded Relational Representation*. MITP, 2009.

[28] D. Katz, M. Kazemi, J. A. Bagnell, and A. Stentz, "Interactive segmentation, tracking, and kinematic modeling of unknown 3d articulated objects," *2013 IEEE International Conference on Robotics and Automation*, pp. 5003–5010, 2013.

[29] D. Katz, A. Orthey, and O. Brock, "Interactive perception of articulated objects," in *Experimental Robotics*. Springer, 2014, pp. 301–315.

[30] Yong Yu, T. Arima, and S. Tsujio, "Estimation of object inertia parameters on robot pushing operation," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, April 2005, pp. 1657–1662.

[31] J. Wu, I. Yildirim, J. J. Lim, B. Freeman, and J. Tenenbaum, "Galileo: Perceiving physical object properties by integrating a physics engine with deep learning," in *Advances in Neural Information Processing Systems 28*, 2015, pp. 127–135.

[32] D. Zheng, V. Luo, J. Wu, and J. B. Tenenbaum, "Unsupervised learning of latent physical properties using perception-prediction networks," *arXiv preprint arXiv:1807.09244*, 2018.

[33] Z. Xu, J. Wu, A. Zeng, J. B. Tenenbaum, and S. Song, "Densephysnet: Learning dense physical object representations via multi-step dynamic interactions," in *Robotics: Science and Systems (RSS)*, 2019.

[34] L. Pinto, D. Gandhi, Y. Han, Y.-L. Park, and A. Gupta, "The curious robot: Learning visual representations via physical interactions," in *European Conference on Computer Vision*. Springer, 2016, pp. 3–18.

[35] H. Van Hoof, O. Kroemer, H. B. Amor, and J. Peters, "Maximally informative interaction learning for scene exploration," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5152–5158.

[36] A. Eitel, N. Hauff, and W. Burgard, "Learning to singulate objects using a push proposal network," 07 2017.

[37] D. Katz, A. Venkatraman, M. Kazemi, J. A. Bagnell, and A. Stentz, "Perceiving, learning, and exploiting object affordances for autonomous pile manipulation," *Autonomous Robots*, vol. 37, no. 4, pp. 369–382, 2014.

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

[39] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.

[40] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, "DART: Dynamic animation and robotics toolkit," *The Journal of Open Source Software*, vol. 3, no. 22, p. 500, Feb 2018. [Online]. Available: https://doi.org/10.21105/joss.00500

[41] A. Hill, A. Raffin, M. Ernestus, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[42] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

APPENDIX

Consider an arbitrary articulated object that has a mass matrix $\mathcal{M}(q)$. We use $q \in \mathbb{R}^{\mathrm{N}}$ to represent the configuration of the object in generalized coordinates.

$$\mathcal{M}(q)\ddot{q} + C(q, \dot{q}) = Q,$$

where C is the coriolis force and Q is the external force both expressed in the generalized coordinates.

Let $J_k \in \mathbb{R}^{6\times \mathrm{N}}$ be the Jacobian and $I_k \in \mathbb{R}^{3\times 3}$ be the inertia matrix of the rigid body k. J_k can further be split into the linear and angular constituent matrices $J_v \in \mathbb{R}^{3\times \mathrm{N}}$ and $J_\omega \in \mathbb{R}^{3\times \mathrm{N}}$ respectively i.e. $J_k = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix}$. By definition,

$$\begin{aligned}\mathcal{M}(q) &= \sum_k J_k^T(q) \begin{bmatrix} m_k I & 0 \\ 0 & m_k I_k \end{bmatrix} J_k(q) \\ &= \sum_k m_k (J_{v_k}^T J_{v_k} + J_{\omega_k}^T I_k J \omega_k).\end{aligned}$$

Assuming $\dot{q} = 0 \implies C(q, \dot{q}) = 0$, the equation of motion can be expressed as

$$\sum_k m_k A_k \ddot{q} = Q,$$

where $A_k = J_{v_k}^T J_{v_k} + J_{\omega_k}^T I_k J \omega_k$.

If $N > 6$, $A_k \in \mathbb{R}^{\mathrm{N}\times \mathrm{N}}$ is not full-rank by definition. Thus there exists a null space for each A_k. If $\ddot{q}$ lies in the null-space of any A_k we cannot uniquely determine the value of m_k. If $N \leq 6$, we still cannot guarantee A_k to be full-rank because if the object is in a kinematic singularity state, $J_k(q)$ will lose rank.