DensePhysNet: Learning Dense Physical Object Representations via Multi-step Dynamic Interactions

Zhenjia Xu[1,2], Jiajun Wu[2], Andy Zeng[3,4], Joshua B. Tenenbaum[2,5], Shuran Song[3,4,6]
[1]Shanghai Jiao Tong University [2]Massachusetts Institute of Technology [3]Princeton University
[4]Google [5]MIT Center for Brains, Minds and Machines [6]Columbia University

***Abstract*—We study the problem of learning physical object representations for robot manipulation. Understanding object physics is critical for successful object manipulation, but also challenging because physical object properties can rarely be inferred from the object's static appearance. In this paper, we propose DensePhysNet, a system that actively executes a sequence of dynamic interactions (e.g., sliding and colliding), and uses a deep predictive model over its visual observations to learn dense, pixel-wise representations that reflect the physical properties of observed objects. Our experiments in both simulation and real settings demonstrate that the learned representations carry rich physical information, and can directly be used to decode physical object properties such as friction and mass. The use of dense representation enables DensePhysNet to generalize well to novel scenes with more objects than in training. With knowledge of object physics, the learned representation also leads to more accurate and efficient manipulation in downstream tasks than the state-of-the-art. Video is available at http://zhenjiaxu.com/DensePhysNet**

I. Introduction

Intelligent manipulation benefits from the ability to distinguish between object materials and infer their physical properties from sight. For example, for tabletop object rearrangement, differentiating heavy and light materials enables better planning of manipulation strategies. Is it possible for robots to self-learn these differences without any explicit supervision?

Although considerable research has been devoted to learning object-centric representations that reflect visual features, they rarely account for latent physical attributes such as mass or friction. Unsupervised learning of physical properties is a less explored problem due to three major challenges:

- Most physical attributes cannot be directly inferred from appearance cues alone in a static environment. For example, while aluminum shares a similar appearance with steel, it is much lighter.
- Most physical attributes are not salient under static or quasi-static interactions: gently pushing a wooden or a metal block results in only subtle differences in their visible motion, despite their different materials and densities.
- Each physical property may only be revealed under specific types of interactions. For example, the sliding distance of an object is determined by both its friction coefficient and mass given its initial momentum; but it is only determined by the object's friction coefficient given its initial velocity. Therefore, without an explicit physics model, the system needs not only to explore different types of interactions, but also to infer and decouple physical properties from multiple action outcomes jointly.

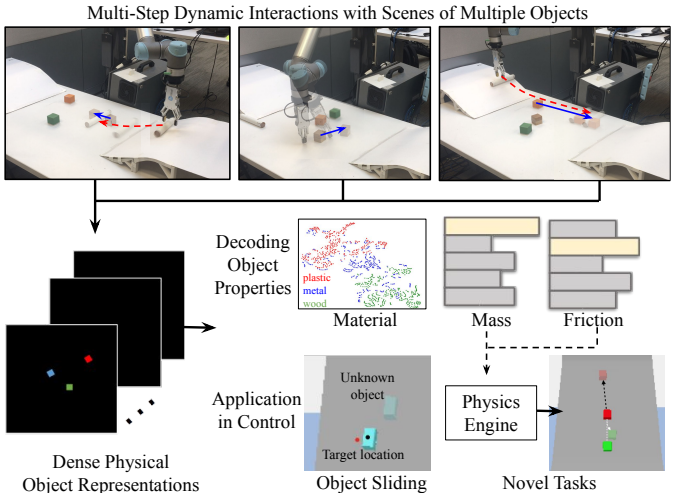


Fig. 1. Our goal is to build a robotic system that learns a dense physical object representation from a few dynamic interactions with objects. The learned representation can then be used to decode object properties such as its material and mass, applied in manipulation tasks such as sliding objects with unknown physics, and combined with a physics engine to tackle novel tasks.

In this work, we propose to discover and learn the physical properties of objects through visual observations of **multi-step, self-supervised, dynamic interactions**. Our system, DensePhysNet, actively executes a sequence of dynamic interactions (e.g., sliding and colliding), and uses its visual observations to learn physical object properties without any explicit supervision (Figure 1).

DensePhysNet takes the current scene and action as input and predicts how objects move after the interaction, represented as pixel-wise optical flow. By learning to predict future object states conditioned on different interactions, DensePhysNet acquires an implicit understanding of their physical properties and how they influence observed motions. We use a recurrent structure to aggregate information from multiple interactions, so that DensePhysNet can better infer and encode objects' physical properties over time. We also design DensePhysNet in a modularized way, so that the learned physical representations can be disentangled from visual representations, which encode objects' visual appearance and actions. This unique design enables it to generalize to new tasks that involve different types of interactions. Because DensePhysNet produces a pixel-wise dense representation, instead of a single feature vector for the entire scene as in many prior methods [19, 16, 1], it also generalizes to complex scenes that contain more objects

than training scenes.

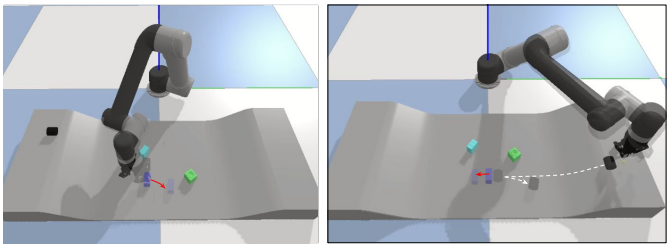
The main contribution of our paper is to demonstrate that deep predictive models over visual observations of multiple dynamic interactions can enable an agent to learn the latent physical attributes of manipulated objects. We execute a series of experiments in both simulation and real settings to evaluate our approach qualitatively and quantitatively. The results show that DensePhysNet is more effective at learning physical object properties than other representation learning methods, and that the learned representations can be leveraged to improve the performance of downstream control tasks such as planar sliding. We also show that when combined with a physics engine, DensePhysNet is capable of leveraging the decoded physical object properties to execute novel control tasks.

II. Related Work

Modeling object physics is a long-standing problem in both robotics and artificial intelligence. Dating back to the 1980s, Atkeson et al. [2] estimated the mass and moments of inertia of a grasped object, using measurements from a wrist force-torque sensor. Yu et al. [30] tackled the same problem by pushing objects using two fingertips equipped with force-torque sensors, and recording the fingertip forces, velocities, and accelerations. Recently, researchers have explored learning to estimate physical properties from their appearance and motion, either in combination with physical simulators [25, 26] or via end-to-end deep learning [10, 13, 28, 24]. These models build upon an explicit physical model, i.e., a model parameterized by physical properties such as mass and force. This enables generalization to new scenarios, but also limits their practical usage: annotations on physical parameters in real-world applications are expensive and challenging to obtain.

An alternative line of work is to learn object representations without explicit modeling of physical properties, but in a 'self-supervised' way through robot interactions. Byravan and Fox proposed to use deep networks to approximate rigid object motion [4]. Pinto et al. [20] proposed to build a 'curious' robot that interacts with objects to learn representations for visual recognition. Denil et al. [8] used reinforcement learning to build object models via physical experiments. Recently, Zheng et al. [33] suggested explicit physical properties can be decoded from the latent representations learned through interactions. A few follow-ups have extended these models for planning and control, including poking [1], pushing via transfer learning [19], and visual predictive learning [11]. There have also been several papers on getting better policies via modeling environment dynamics [29, 34]. Such progress is impressive, though the focus of these papers is still on *visual* representation learning. Without considering the underlying physical processes, they fall short to generalizing to novel objects and tasks.

The work closest to ours is Push-Net, proposed by Li et al. [16], which uses a multi-step model to learn physical object properties for planar pushing. While Push-Net focuses on scenes with a single object and encodes the entire image into a single latent representation, we instead learn dense (pixel-wise) representations from interactions – which enables our model to generalize not only to novel objects, but also to novel scenes with multiple objects. Furthermore, while Push-Net only considers planar quasi-static pushing, we consider two types of dynamic interactions: planar sliding and collision. Our experiments demonstrate that using multiple types of dynamic interactions is key to fully revealing latent physical properties.



(a) Sliding object (b) Colliding with auxiliary object

Fig. 2. **Dynamic Interactions**. We design two types of dynamic interactions (sliding and collision) to reveal physical object properties. To slide an object, the robot approaches it from an angle and executes a push with a high speed, such that the object can slide after the push. The robot releases an auxiliary object on the ramp to make collisions.

We are, of course, not the first to learn dense representations on visual data. Most prior work on this topic revolve around learning correspondences across views in 2D [6, 21] and 3D [31, 23, 22, 3]. Florence et al. [12] proposed dense object nets, learning dense descriptors by multi-view reconstruction and applying the descriptors to manipulation tasks. While their paper primarily focuses on learning object representations that reflect visual appearance, we learn object representations that reflect physical properties and show that they are useful for manipulation tasks that require physical knowledge.

III. Method

The goal of DensePhysNet is to learn latent representations that encode object-centric physical properties (e.g., mass, friction) through self-supervision. To this end, we train a deep predictive model of depth images on a large dataset of observed dynamic robotic interactions. The idea is that in order for DensePhysNet to accurately predict the future states of objects conditioned on different interactions, it needs to acquire an implicit understanding of objects' physical properties and how they influence observed motion. The setting consists of a collection of objects on an inclined ramp laid in front of the robot (Figure 2). When interacting with objects, the robot captures a depth image I_t of the state at time t, executes an action a_t, then captures another depth image I_{t+1} at the next time step. We model DensePhysNet as a neural network that takes as input the visual observation I_t and the executed action a_t, and outputs a prediction of the next state observation I_{t+1} in the form of optical flow $O_{t,t+1}$.

A. Dynamic Interactions

The robotic agent executes two types of dynamic interactions to accentuate the physical properties of objects: sliding and collision. The vast majority of prior methods in representation learning [16, 11, 19, 1] use static or quasi-static manipulations like pushing, grasping, or poking, where it is challenging to observe latent physical object properties from object motion. This is because in (quasi-)static manipulation, changes in object

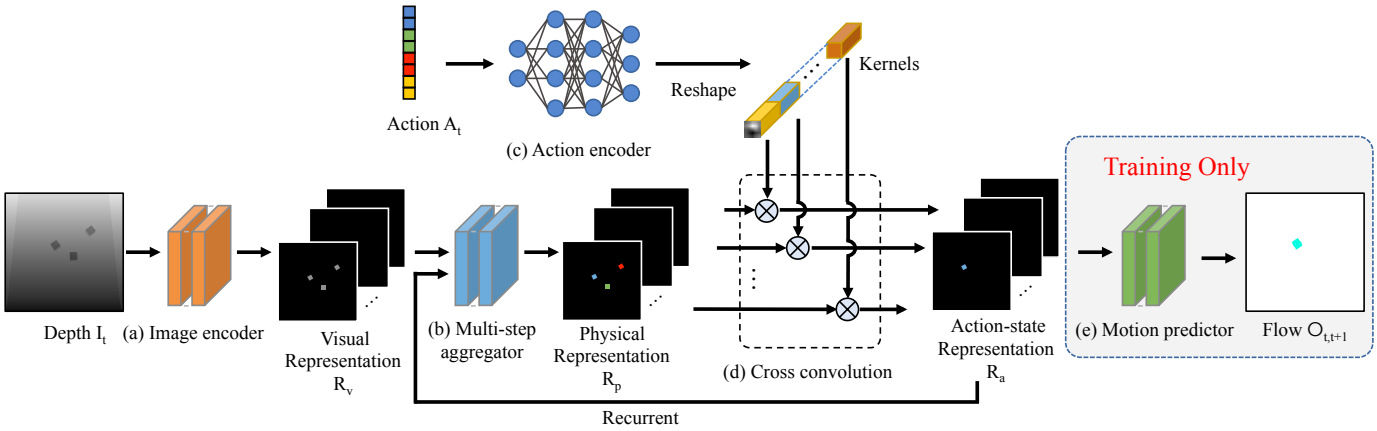


Fig. 3. **DensePhysNet**. DensePhysNet takes in the current state (depth image I_t) and interaction A_t as input and predicts the change of objects' state after the interaction. The change of objects' state is represented as pixel-wise optical flow $O_{t,t+1}$. The network consists of five modules: (a) an image encoder, (b) a multi-step information aggregator, (c) an action encoder, (d) a cross convolutional layer, and (e) a motion predictor. These five modules work jointly to learn three object representations: visual representations R_v that encode visual signals of the object, physical representations R_p that encode physical object properties, and action-state representations R_a that encode objects' states after interaction.

movements are largely influenced by the actions and dynamics of the manipulator, less so from the object itself. For example, during quasi-static pushing, the object is assumed to move only with the end effector and to stop when the end effector stops. In these scenarios, it is naturally more difficult to observe the subtleties in motion due to the physical properties of the object.

Dynamic manipulations, in contrast, can lead to object motions beyond contact with the manipulator that are more likely to reveal its physical properties. For example, when objects are pushed with a high speed, the forces of acceleration can cause objects to slide by themselves for a distance. The differences in their motion after contact, e.g., distance traveled, can serve as a visual cue for differences in surface friction. These cues are relatively more salient than what can be observed in quasi-static manipulations. However, the distance traveled by an object after sliding alone, assuming known initial velocity, can only help derive the object's friction. Therefore, we need other types of dynamic manipulations such as collisions to reveal and distinguish other physical properties.

Sliding. The sliding action is parameterized by direction θ and velocity v with respect to the robot. To slide an object, the robot approaches it from θ and executes a push with a high speed such that the object can slide after the push. To achieve high pushing velocities without exceeding the physical force-torque safety limits of the real-world robot, we constrain the motion planning so that joints closer to the base of the arm (e.g., elbow, shoulder) move slower than joints closer to the end effector (e.g., wrist). The direction θ and velocity v are quantized as discrete variables and we use one-hot vectors to encode them. There are sixteen possible directions and four possible speeds. The sixteen directions are uniformly distributed on $[0, 2\pi]$. Four robot's speeds are $\{0.96, 1.28, 1.44, 1.6\}$rps, representing the robot's joint rotation speed.

Collision. For collisions, we set two inclined ramps, one on each side of the workspace. The robot grasps an auxiliary cylinder and places it on the top of one of the ramps. The cylinder then rolls down the ramp and collides with an object in the middle. The only parameters of the collision action are the cylinder's starting position $[d \times X_s, y, H_s]$, where $d \in \{-1, 1\}$ indicates whether the cylinder is rolling from the left or right ramp, X_s is the distance from the ramp to the center of workspace, y is the same as the y-coordinate of the target object, and H_s is the height of the ramp. The auxiliary cylinder is fixed with radius 2cm, height 4cm, and weight 0.4kg.

Interaction policy. We use a balanced-random interaction policy to ensure interaction diversity and to encourage a full exploration of each action. Specifically, for each step, we choose one type of action to execute via $P(x_{i,j}) = (1/2)^{a_{i,j}} / \sum_{n=1}^{N} \sum_{m=1}^{M} (1/2)^{a_{n,m}}$, where $P(x_{i,j})$ is the probability to apply action i to object j, N is the number of objects in the scene, M is the size of action space, and $a_{i,j}$ is the number of action i applied to object j up to now. We randomly sample the parameters for the chosen action, while enforcing the constraint that the object needs to stay inside the workspace.

B. DensePhysNet

We design DensePhysNet based on three key insights. First, it needs to be modularized in a way that the learned physical representations are disentangled from the representations that encode information about object visual appearance and actions. This is critical for applying the learned physical representations to new tasks, objects, and action types. Second, DensePhysNet has a recurrent structure, so that it can aggregate information from multiple interactions to better infer physical properties. Third, our model produces a dense pixel-wise representation, instead of encoding the entire scene into one single latent representation as in previous papers [16]. This enables handling complex scenes with multiple objects.

DensePhysNet (Figure 3) consists of five modules: an image encoder, a multi-step information aggregator, an action encoder, a cross convolutional layer, and a motion predictor. These five modules work jointly to learn three object representations: visual representations R_v that encode visual signals of the object, physical representations R_p that encode physical object properties, and action-state representations R_a that encode objects' states after interaction.

Each iteration of learning works as follows. First, given a

depth image I_t, the image encoder (Figure 3a) extracts visual signals from the image and outputs the visual presentation R_v. Then, the information aggregator (Figure 3b) learns to integrate the visual representation R_v with the object representation after the last interaction R_a to extract the physical representation R_p. Intuitively, after interactions, objects with different physical properties will end up in different positions and poses, and such signals are now available from the visual input and therefore should lie within the visual representation R_v. The goal of the information aggregator is thus to distill physical knowledge R_p by analyzing R_v and R_a jointly.

In parallel, the action encoder (Figure 3c) encodes action a_t into convolution kernels. The cross convolutional layer (Figure 3d) then applies the encoded action kernels on the physical representation R_p to produce the effect of the action on the objects. Here, the cross convolutional layer can be seen as a learned, latent physical simulator, learning physics to approximate the effect of actions. As shown in Xue et al. [27], this cross convolutional layer better integrates the action and the physical representations compare to tensor concatenation. It outputs the action-state representation R_a.

Finally, the action-state representation R_a is fed into the motion predictor (Figure 3e) to predict the optical flow $O_{t,t+1}$ across two images I_t and I_{t+1}. This is the only supervisory signal we have during training; during testing, the motion predictor is no longer needed.

Network architecture. Here we provide more details on each network module. The image encoder takes as input the depth image with a size of 160×160, and outputs a 32-channel feature map. It has one 11×11, one 5×5, and two 3×3 convolutional layers. There are batch normalization and ReLU layers between adjacent convolutional layers.

The action encoder uses seven fully connected layers with $64, 128, 256, 256, 256, 512$, and 800 hidden units, respectively. It takes in the action vector (a 37-dim vector) an input and outputs 32 action kernels, each with a size of 5×5.

The multi-step aggregator takes the visual representation R_v and the action-state representation R_a in the last step as input (both have a size of $32 \times 160 \times 160$), and combines them with a gate C via $C = S(g(R_v))f(R_v) + [1 - S(g(R_v))]\, R_a$, where $S(\cdot)$ is the sigmoid function and $f(\cdot)$ and $g(\cdot)$ are a 1×1 convolution layer. The gate acts as a filter to block or pass on information based on the calculated weight. Then, fifty residual blocks [14] are applied to the combination C to get the physical representation.

The cross convolutional layer applies the convolutional kernels from the action encoder to the physical representation R_p from the multi-step aggregator to produce the action-state representation R_a. Here, the convolutions are carried out in a channel-wise manner, i.e., each of the 32 layers in the physical representation is convolved with one of the 32 kernels.

The motion predictor takes the action-state representation R_a as input and outputs a pixel-wise optical flow map. This module consists of four 3×3 convolution layers with $32, 32, 32$, and 2 channels, respectively. In between, there are batch normalization and ReLU layers.

Self-supervised training. We use Mean Square Error (MSE) between the predicted optical flow and ground truth as the training loss. Because optical flow can be automatically computed based on visual observations (i.e., images taken before and after the interaction), the whole training process of DensePhysNet can be fully self-supervised without any human annotations. We implement our model in PyTorch [18]. Optimization is carried out using ADAM [15] with $\beta_1 = 0.9$ and $\beta_2 = 0.95$. We use an initial learning rate of 10^{-3} and a learning rate decay of 0.9 after each epoch. The model is trained for 40 epochs with a mini-batch size of 32.

IV. Experiments

We now evaluate the learned physical representation R_p. The goal of the experiments is to understand, through both qualitative analysis and quantitative evaluation, whether the learned physical representations encode information about physical object properties; if so, how accurate the encoding is, and how useful the representation is for object manipulation.

A. Decoding Object Material

We first analyze whether the learned physical representation can be used to distinguish objects of different materials.

Baselines. We compare our model for learning physical representations R_p with two baselines to understand the role of active interaction.

- Visual representations: we first compare the physical representations R_p with the learned visual representations R_v extracted from our model (see Figure 3). The learned visual representations are computed directly from static images, without the knowledge of object motion or actions.
- Our model with sliding: we also compare with our model trained with only sliding, but not collisions, to validate the importance of having multiple types of interactions.

Setup. For experiments in simulation, we use three visually indistinguishable objects, made of one of the three materials:

- Plastic with mass $m \in [0.11, 0.14]$kg and friction coefficient $\mu \in [0.4, 0.6]$.
- Wood with mass $m \in [0.11, 0.14]$kg and friction coefficient $\mu \in [0.8, 1.0]$.
- Metal with mass $m \in [0.17, 0.20]$kg and friction coefficient $\mu \in [0.4, 0.6]$.

Each object's physical properties are uniformly sampled according to its material. Hence, objects of different materials have distinct physical properties, and those made of the same material have similar, but not identical, properties. We use $8{,}000$ sequences for training. We use PyBullet for all our experiments in simulation [7]. For each trial, the robot interacts with the objects 19 times (in test mode without optical flow supervision). We use background subtraction to compute the silhouette of each object; we then extract pixel-wise features for each silhouette as object representations.

We also evaluate DensePhysNet in real-world settings, where we use a UR5 robot arm with an RG2 gripper. Figure 4 shows the setup. We capture RGB-D images using a calibrated Intel RealSense D415, mounted on a fixed tripod overlooking the

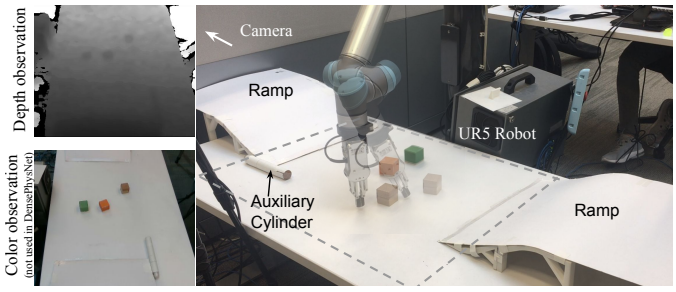


Fig. 4. **Real-world experiment Setup.** The real-world settings includes UR5 robot arm with an RG2 gripper. A flat table with two incline ramp on both side for collision. We capture RGB-D images using a calibrated Intel RealSense D415, mounted on a fixed tripod overlooking the table from the side.

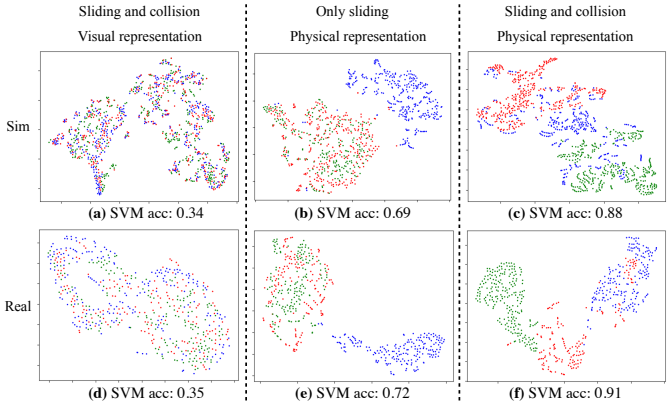


Fig. 5. **t-SNE embeddings of learned features.** The first row shows results in simulation and the second shows results of real-world experiments. We also compare different representations and actions. For each result, we train a 3-way SVM for material classification; its accuracy is presented under the embedding visualization. For simulation results, the visual representation (a) is not informative in distinguishing object material. The representation learned from sliding alone (b) is only informative in distinguishing the difference in friction (wood vs. others), but has trouble telling the difference in mass (plastic vs. metal). Our DensePhysNet (c), designed to learn physical object representations, learns to cluster objects based on their material. Also, in real-world experiments, only the physical representation learned from both interactions (f) can distinguish different materials.

table of objects from the side. The camera is localized using an automatic calibration procedure [32]. The pushing and collision primitives are open-loop, with robot arm motion planning executed using IK solvers [9]. In real-word experiments, we use the model pre-trained in simulation. In total, we test on 20 sequences for three objects of different materials. Each sequence contains fifteen interaction steps. We consider two kinds of interactions policies: (1) sliding only; (2) both sliding and collision. For each sequence, we ensure the object is always within the workspace. In total we collect 10 sequences for each interaction policy.

Results. We randomly sample 1,000 pixels within object silhouettes from 100 test sequences. We then embed the corresponding 1,000 learned feature vectors using 2D t-SNE embeddings [17]. The first row of Figure 5 shows the results, where colors represent different materials. Our model, designed to learn physical object representations, learns to cluster objects based on their material. The representations learned from sliding alone is only informative in distinguishing the difference

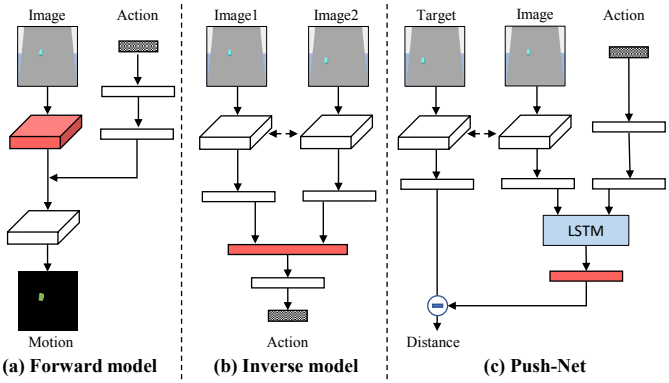


Fig. 6. **Baseline models.** Agrawal et al. [1] proposed a forward and an inverse model, both designed to handle a single-step interaction. The forward model (a) takes the current frame and the action as input, and predict the motion of the object. The inverse model (b) takes the frames before and after the action as input and predicts the action parameters. Push-Net [16] (c) uses an LSTM to capture the history of interactions.

in friction (wood vs. others), but has trouble differentiating different mass (plastic vs. metal). The visual representation is not informative in distinguishing object material.

We further verify each representation's discriminative power by training a 3-way linear-SVM on 1,000 feature vectors (pixels) randomly sampled from 100 training sequences, and testing its material classification accuracy on 100 feature vectors (pixels) randomly sampled from 100 test sequences. In simulation, the representation learned by DensePhysNet achieves a 88% accuracy on material classification, while the model trained only on sliding achieves an accuracy of 69%. The visual representation achieves an accuracy of 34%, close to random guess (33.3%). In the real-world experiment, DensePhysNet achieves a 91% accuracy when using both kinds of interactions. If the model only uses sliding, the accuracy is 72%. The visual representation achieves an accuracy of 35%.

B. Decoding Physical Object Properties

In this experiment, we examine how accurately we can decode physical properties from the learned latent representations.

Baselines. We compare with the following baselines.

- Our model with different interaction types.
- Single-step forward model [1]: We also compare our approach with the the model proposed by Agrawal et al. [1]. Their model consists of a forward and an inverse model, both designed to handle a single-step interaction instead of aggregating the information across multiple interactions. The forward model takes the current frame and the action as input, and predict the motion of the object for one step (Figure 6a).
- Single-step inverse model [1]: The inverse model takes the frames before and after the action as input, and predict the action parameters (Figure 6b).
- Push-Net [16]: For each step, the model takes the current mask M_t, the action and a target mask M_{t+1} as input, and predict the distance between the underlying states of the target M_{t+1} and the mask generated as a result of applying the action to M_t. The original Push-Net also predicts the object's center of mass. For a fair comparison, we omit this

TABLE I
PHYSICAL PROPERTY DECODING ERROR D_{ERR} WITH DIFFERENT MODELS

	Forward [1]	Inverse [1]	Push-Net [16]	Ours
friction	10.00	6.36	4.67	**3.81**
mass	10.00	6.87	6.96	**4.24**

TABLE II
PHYSICAL PROPERTY DECODING ERROR D_{ERR} WITH DIFFERENT ACTIONS

	Slow Push	Sliding	Collision	Slow Push & Collision	Sliding & Collision
friction	8.87	4.07	6.58	5.92	**3.81**
mass	9.76	9.23	7.03	6.75	**4.24**

part since it requires additional annotation (Figure 6c).

Setup. To this end, we train a linear classifier to decode physical properties from the latent representations on an annotated dataset, and test it on a set of novel objects of different shapes. During testing, the robot interacts with the objects to update the latent representations, but no optical flow supervision is used. We conduct this experiment in simulation, where ground truth physical properties are available for training. We only evaluate on scenes of a single object, as both the forward and the inverse models in Agrawal et al. [1] only handle single-object scenarios and predict one feature vector for the whole scene. The shape of each object is randomly sampled from ShapeNet [5], where training and testing objects are of different shapes. Each object's physical properties (friction and mass) are uniformly randomly sampled from 30 discrete values: $\mu \in \{0.4, 0.41, 0.42, \ldots, 0.7\}$ and $m \in \{0.11, 0.113, 0.116, \ldots, 0.2\}$kg. There are 8,000 sequences for training and 2,000 for testing. The robot has 9 interaction steps with the object using three types of policies: sliding only, collision only, and both sliding and collision. We extract the pixel-wise features of the object from the last step using its bounding box (automatically computed via background subtraction), and flatten it into one vector. For the forward and the inverse models, we directly take its hidden layer features (marked red in Figure 6) as the features of the object.

For evaluation, we train both the friction decoder and the mass decoder as a 30-way linear classifier using the cross-entropy loss. The classifiers are trained on 1,000 of the 8,000 training sequences and evaluate it on all 2,000 test sequences. We then calculate the weighted distance error for each piece of data as the evaluation metric: $D_{\text{err}} = \sum_{i=1}^{30} p_i \times |i - y|$, where p_i is the probability of category i predicted by our model and y is the ground truth. We use this metric for many of the following experiments.

Results. Table I shows that DensePhysNet outperforms the baselines, demonstrating the importance of information aggregation across multi-step interactions in modeling physical object properties. The forward model only takes the current image and the action as input. Its representation only contains visual information, which does not help to predict physical properties. Its result is thus the same as random guessing. The inverse model takes images before and after the interaction

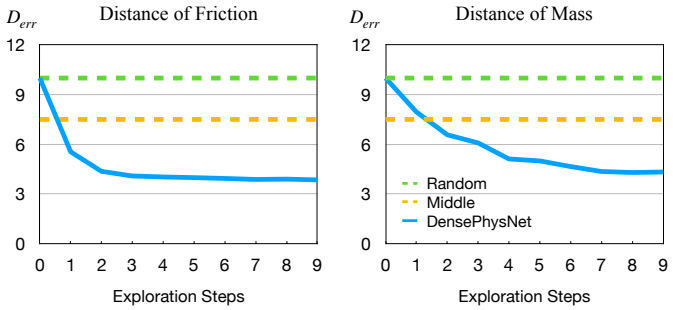


Fig. 7. **Decoding physical object properties.** The plot shows the average distance error on decoding friction coefficient and mass at each step. At the beginning, our model's prediction accuracy is, as expected, the same as random guess; over the course of interactions, the average distance decreases quickly, which means our model gradually accumulates knowledge of object physics.

and learns some information about physical properties from object motion. However, it cannot handle long-range data; its performance is therefore limited.

Table II demonstrates the comparison between different kinds of interactions. Compared with slow push, the performance of sliding is much better. Moreover, sliding helps a lot for the learning of friction, while the mass can only be inferred from the combination of sliding and collisions. In short, dynamic interactions are much more effective than quasi-static interactions and the diversity of action space is of vital importance.

We also conduct an ablation study to understand how the number of interaction steps affects the results. Figure 7 shows DensePhysNet's performance in each step. For calibration, 'Random' shows the performance of random guessing: $D_{\text{err}} = \frac{1}{30} \times \sum_{gt=1}^{30} \sum_{i=1}^{30} \frac{1}{30} \times |i - gt| \approx 10$. 'Middle' shows the performance of predicting the average mass and friction: $D_{\text{err}} = \frac{1}{30} \times \sum_{gt=1}^{30} |15.5 - gt| = 7.5$. At the beginning, our model's prediction is similar to random guessing. Over the course of interactions, the error decreases quickly, which means our model gradually accumulates knowledge of object physics.

C. Application in Sliding Objects with Unknown Physics

The ability to understand physical properties is important for flexible manipulation. In this experiment, we test whether our model can efficiently infer physical properties of unknown objects through multi-step interactions, and further, can make use of its knowledge of object physics to suggest more accurate policies in object manipulation.

Setup. Our task is to push an unknown object so that it slides to the target position. The target position is uniformly randomly chosen in a circle centered on the object and with a radius of 0.25m. The robot needs to propose the parameters of the push, including its direction and initial speed. The sixteen possible directions are uniformly distributed on $[0, 2\pi]$ and the speed can be chosen from $\{0.96, 1.28, 1.44, 1.6\}$rps. After each interaction, the new action-state representation is fed back to the multi-step aggregator to improve the physical representation. Intuitively, if the target object is heavy, the system should be able to infer its mass through interactions and then push it with a higher speed. Figure 8 shows an illustration of the task.

The physical properties of the objects are randomly chosen

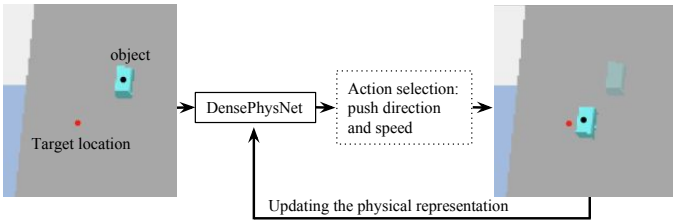


Fig. 8. **Application in object sliding.** The goal for this task is to push an object with certain direction and speed, so that the object will slide to the target position. At each step, DensePhysNet predicts the motion of the object for each possible action in the action space, then selects the one whose outcome is closest to the target. After each interaction, the new action-state representation is fed back to the multi-step aggregator to improve the object's physical representation, which consequentially improves the action predictions in the following interactions.

from two distribution families:

- common objects: $m \in [0.15, 0.16]$kg and $\mu \in [0.6, 0.8]$,
- uncommon objects: $m \in [0.11, 0.13]$kg or $[0.18, 0.2]$kg and $\mu \in [0.4, 0.5]$ or $[0.9, 1.0]$.

We compare our model with Push-Net, and the forward and the inverse models as introduced in Section IV-B. Because our model needs a sequence of interactions to learn physical properties, we also evaluate our model after 0, 3, and 7 interaction steps to understand how interaction helps control. Here, the interaction steps are just for updating the latent representations; no optical flow supervision or finetuning is needed. The policies used by different models are as follows:

- Our model and the forward model: enumerate all possible actions and predict the motion of the object, and then choose the action whose predicted position is closest to the target.
- The inverse model: use the action predicted by the model from the current and the target image.

We test each model 100 times and calculate the mean distance between the object's position after interactions and the target.

Results. Figure 9 shows the results. For objects with common physical properties, all these models have similar performance with a mean distance error of 0.06m. However, if the physical properties are uncommon, our model and Push-Net outperform the other two baselines after a few explorations. This suggests that interactions help to refine the estimation of physical properties and lead to better performance. Moreover, our model achieves similar performance with Push-Net, without requiring direct supervision as Push-Net does.

Real-world experiments. We also evaluate DensePhysNet in real-world settings to push a collection of objects on a flat table. We use the model pre-trained on 8,000 interaction sequences in simulation. In total, we have tested 18 sequences for six different objects with distinct physical properties, appearance, and shape. Each sequence contains five interaction steps. The target location is manually selected for each step.

Figure 10 shows qualitative results for different objects. DensePhysNet not only generalizes from simulation to real-world settings, but also makes use of the exploration steps to improve its action predictions. In particular, for the objects with an unusually large mass, such as the 'heavy object' in Figure 10, the model without exploration (step 0) predicts a

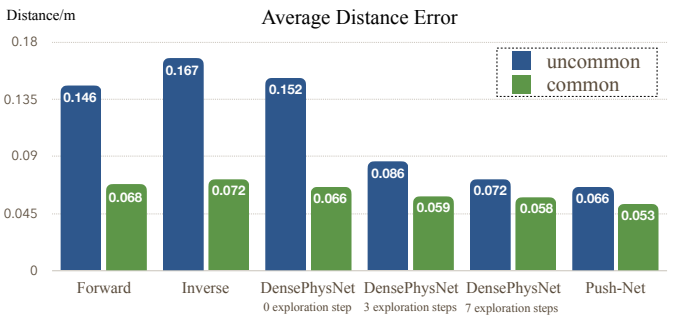


Fig. 9. **Results on object sliding in simulation.** For objects with common physical properties, both our model and the baseline work well. However, for objects with uncommon physical properties, only our model and Push-Net (after a few exploration steps) perform well. This is because the recurrent structure in DensePhysNet and Push-Net learns to infer physical properties via aggregating information from history trajectories.

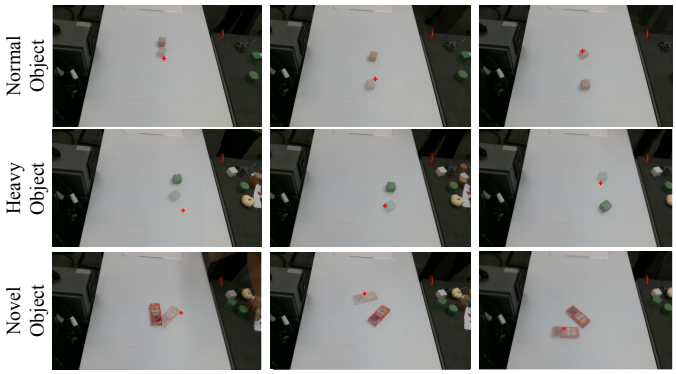


Fig. 10. **Real-world object sliding experiment.** Each row shows the object sliding result for one object after 0, 2, and 5 exploration steps, where the transparent object indicates its position after applying the suggested action. The red cross indicates the target location. The 'normal' object is a wooden block that has similar physical property as the training objects in simulation. The 'heavy' object is a plastic box filled with heavy metal balls. The 'novel' object is a snack box that has a different shape from the training objects.

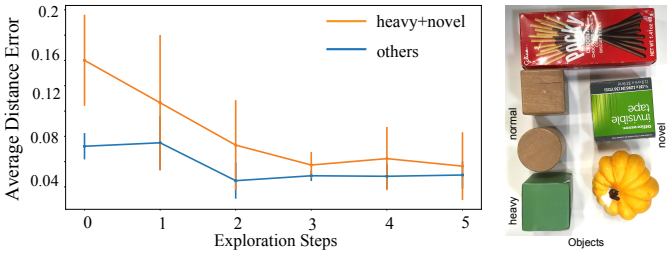


Fig. 11. **Results on object sliding in the real world.** Our algorithm predicts more accurate actions and achieves lower errors after a few explorations steps, especially for heavy and novel objects.

push speed that is too small for the object to slide to the target position. But just after two steps of interaction, the model learns to adjust its prediction according to the estimated physical properties. Figure 11 shows the mean distance error for the 'heavy' and 'novel' objects (objects whose shape is different from those in training), as well as for all other 'normal' objects. On average, the algorithm predicts more accurate actions and achieves lower errors after a few explorations steps, consistent with the experiments in simulation. As expected, the gain of exploration is more significant for heavy and novel objects than normal objects.

D. Generalization

Practical robotic systems need to generalize to complex scenarios. Here we evaluate on two cases: generalizing to scenes with more objects and generalizing to novel tasks.

Generalizing to scenes with more objects. Pixel-wise dense representations work for scenes with multiple objects, and further, generalizes to scenes with more objects than those in training scenes. In this experiment, we train the model using two objects and test it with three objects. Each scene consists of objects with different shapes and physical properties. The robot is allowed to have 19 interactions for both training and testing. Other setups of this experiment is the same as in Section IV-B.

There are 5,000 sequences with two objects for training and 1,500 sequences for testing: 500 with two objects, 500 with three objects, and 500 with four objects. We calculate the average distance between the predicted physical properties and ground truth for each object in each piece of data. Figure 13(a) shows the average error. The small gap between scenes with different numbers objects suggests that our DensePhysNet generalizes to new scenes with more objects; in contrast, baselines [16, 1] do not have an object-wise representation and cannot directly work on scenes with multiple objects.

Generalizing to a novel task. Once we have decoded the physical properties of objects, we can integrate them into a physics engine for planning and control in alternative tasks. As a demonstration, we study a new task, where the goal is to slide an auxiliary cube so that it hits the object to a target position. Here, we need to select the mass and speed of the auxiliary cube. This task is different from object collisions in the training phase: during training, the auxiliary object is a cylinder with a fixed size, mass, initial position and speed.

The robot first interacts with the target object for 7 steps using the pre-trained model in Section IV-B, without finetuning; it then uses the decoder, also trained in Section IV-B, to decode physical properties from the physical representation. We set the target object's initial position to always be at center of the workspace, and its target position to the north of the initial position with a distance uniformly sampled from $[0.1, 0.3]$m. We set the position of the auxiliary cube to be 0.05m south to the center. The mass of the auxiliary cube can be chosen from $\{0.4, 0.43, 0.46 \dots 0.7\}$kg and the speed can be chosen from $\{0.5, 0.53, 0.56, \dots 0.8\}$m/s, and its friction is fixed at 0.2.

For each model, we simulate the collision with the decoded physical properties and each possible pair of mass and speed for the auxiliary object; we choose the pair that gives the best prediction. Figure 12 demonstrates the pipeline of this experiment. We test each model 100 times and calculate the mean distance between the target position and the objects position final position after collision.

Figure 13(b) shows the results. Our model outperforms other baselines significantly. Further, the performance of using only sliding during exploration is not as good as the one using both sliding and collisions. Again, it demonstrates that a diverse set of action types is important to accurate prediction and better performance in control.

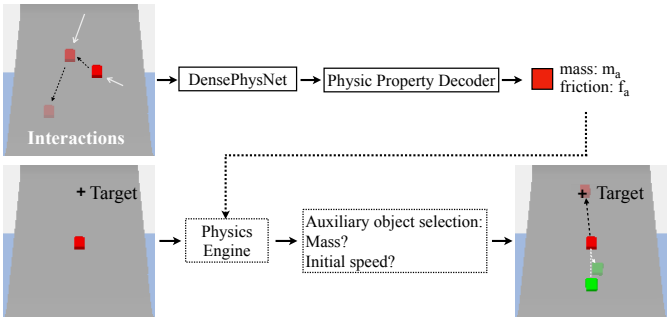
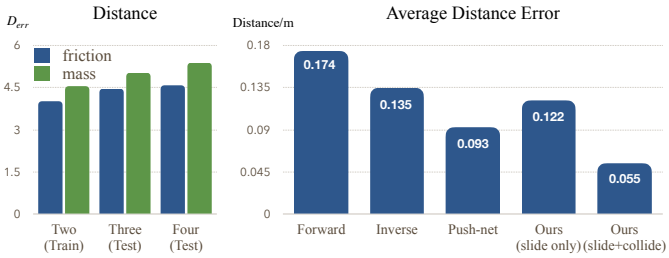


Fig. 12. **Experiments on task generalization.** Here, the robot first interacts with the objects. The learned representation is then used to decode physical properties, which are later used in a physics engine for planning in other tasks.



(a) Generalize to more objects **(b)** Generalize to novel tasks with a physics engine

Fig. 13. **Generalization.** (a) Results of generalizing to more objects. (b) Results of generalizing to a new task, where the algorithm combines the decoded object physical properties with a general physics engine to predict planning and control parameters. In this new task, DensePhysNet outperforms baselines significantly. In particular, using both types of interactions is important to achieve good generalization results.

V. Discussion and Future Work

We have proposed DensePhysNet, a model that learns dense, physical object representations from self-supervised interactions. We have demonstrated that DensePhysNet learns about object materials and physics through both qualitative and quantitative analyses. Further, we have shown that the learned representations can be used in downstream control tasks such as planar sliding to suggest more accurate action policie. Below, we discuss the key features and limitations of our design.

The design of action space is important for learning object physics. We have shown that, with both planar sliding and collisions, the model learns to infer both object mass and friction. Extending DensePhysNet to accommodate a much richer set of interactions would fully demonstrate its potentials.

DensePhysNet learns physical representations from object motion via depth images; our model does not take color images as input. Color signals can however be informative of object geometry and potentially physics, as shown in prior research on visual representation learning [20, 12]; they can also help to scale our model to a richer set of objects. Therefore, a promising research direction is to build models that learn from both color and motion cues for better object representations.

Acknowledgments

This work is in part supported by the Center for Brains, Minds and Machines (NSF STC award CCF-1231216), ONR MURI N00014-16-1-2007, Google, and Facebook.

References

[1] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[2] Christopher G Atkeson, Chae H An, and John M Hollerbach. Estimation of inertial parameters of manipulator loads and links. *The International Journal of Robotics Research (IJRR)*, 5(3):101–119, 1986.

[3] Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, and Carsten Rother. Learning 6d object pose estimation using 3d object coordinates. In *European Conference on Computer Vision (ECCV)*, 2014.

[4] Arunkumar Byravan and Dieter Fox. Se3-nets: Learning rigid body motion using deep neural networks. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017.

[5] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. *arXiv:1512.03012*, 2015.

[6] Christopher B Choy, JunYoung Gwak, Silvio Savarese, and Manmohan Chandraker. Universal correspondence network. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[7] Erwin Coumans. Bullet physics engine. *Open Source Software: http://bulletphysics. org*, 2010.

[8] Misha Denil, Pulkit Agrawal, Tejas D Kulkarni, Tom Erez, Peter Battaglia, and Nando de Freitas. Learning to perform physics experiments via deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

[9] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, 2010.

[10] Sebastien Ehrhardt, Aron Monszpart, Niloy Mitra, and Andrea Vedaldi. Taking visual motion prediction to new heightfields. *arXiv:1712.09448*, 2017.

[11] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE, 2017.

[12] Peter R Florence, Lucas Manuelli, and Russ Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. In *Conference on Robot Learning (CoRL)*, 2018.

[13] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. In *International Conference on Learning Representations (ICLR)*, 2016.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[16] Jue Kun Li, David Hsu, and Wee Sun Lee. Push-net : Deep planar pushing for objects with unknown physical properties. In *Robotics: Science and Systems (RSS)*, 2018.

[17] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research (JMLR)*, 9:2579–2605, 2008.

[18] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *Advances in Neural Information Processing Systems (NeurIPS) Workshops*, 2017.

[19] Lerrel Pinto and Abhinav Gupta. Learning to push by grasping: Using multiple tasks for effective learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[20] Lerrel Pinto, Dhiraj Gandhi, Yuanfeng Han, Yong-Lae Park, and Abhinav Gupta. The curious robot: Learning visual representations via physical interactions. In *European Conference on Computer Vision (ECCV)*, 2016.

[21] Tanner Schmidt, Richard Newcombe, and Dieter Fox. Self-supervised visual descriptor learning for dense correspondence. *IEEE Robotics and Automation Letters (RA-L)*, 2(2):420–427, 2017.

[22] Jamie Shotton, Ben Glocker, Christopher Zach, Shahram Izadi, Antonio Criminisi, and Andrew Fitzgibbon. Scene coordinate regression forests for camera relocalization in rgb-d images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[23] James Thewlis, Hakan Bilen, and Andrea Vedaldi. Unsupervised learning of object frames by dense equivariant image labelling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[24] Nicholas Watters, Andrea Tacchetti, Theophane Weber, Razvan Pascanu, Peter Battaglia, and Daniel Zoran. Visual interaction networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[25] Jiajun Wu, Ilker Yildirim, Joseph J Lim, William T Freeman, and Joshua B Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[26] Jiajun Wu, Erika Lu, Pushmeet Kohli, William T Freeman, and Joshua B Tenenbaum. Learning to see physics via visual de-animation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[27] Tianfan Xue, Jiajun Wu, Katherine Bouman, and William T Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[28] Tian Ye, Xiaolong Wang, James Davidson, and Abhinav

Gupta. Interpretable intuitive physics model. In *European Conference on Computer Vision (ECCV)*, 2018.

[29] Wenhao Yu, Jie Tan, C Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. *arXiv preprint arXiv:1702.02453*, 2017.

[30] Yong Yu, Tetsu Arima, and Showzow Tsujio. Estimation of object inertia parameters on robot pushing operation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[31] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, and Jianxiong Xiao. 3dmatch: Learning the matching of local 3d geometry in range scans. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[32] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[33] David Zheng, Vinson Luo, Jiajun Wu, and Joshua B Tenenbaum. Unsupervised learning of latent physical properties using perception-prediction networks. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.

[34] Wenxuan Zhou, Lerrel Pinto, and Abhinav Gupta. Environment probing interaction policies. In *International Conference on Learning Representations (ICLR)*, 2019.