APPEARED IN
Proc. Robotics: Science & Systems, 2018

Push-Net: Deep Planar Pushing for Objects with Unknown Physical Properties

Jue Kun Li, David Hsu, Wee Sun Lee
School of Computing, National University of Singapore, 117417 Singapore

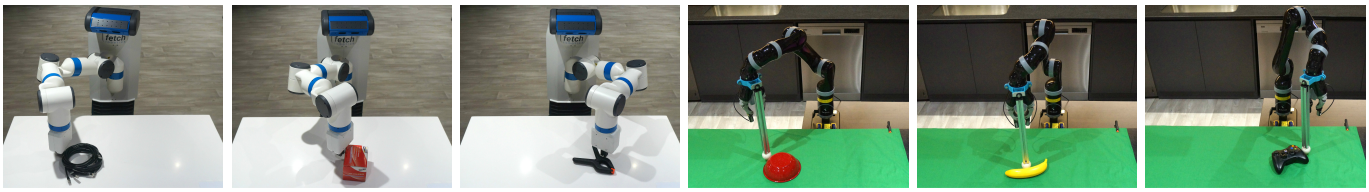


Fig. 1: Push-Net enables both a Fetch arm and a Kinova MICO arm to robustly and efficiently push novel objects of unknown physical properties for re-positioning and re-orientation on a 2D plane.

***Abstract*—This paper introduces Push-Net, a deep recurrent neural network model, which enables a robot to push objects of unknown physical properties for re-positioning and re-orientation, using only visual camera images as input. The unknown physical properties is a major challenge for pushing. Push-Net overcomes the challenge by tracking a history of push interactions with an LSTM module and training an auxiliary objective function that estimates an object's center of mass. We trained Push-Net entirely in simulation and tested it extensively on many different objects in both simulation and on two real robots, a Fetch arm and a Kinova MICO arm. Experiments suggest that Push-Net is robust and efficient. It achieved over 97% success rate in simulation on average and succeeded in all real robot experiments with a small number of pushes.**

I. Introduction

Pushing is a simple yet powerful action that can be used to complement grasping when doing manipulation. Pushing can be used to re-position objects that may not be easily graspable. For example, pushing a pile of wires to another location on the table (leftmost in Fig. 1) may be more effective than moving it via grasping. Pushing can also be used to re-orient an object when its initial orientation fails to afford a grasping action. In this work, we examine whether pushing for re-positioning and/or for re-orientation can be done with high reliability for arbitrary convex objects with unknown physical properties, such as friction and mass distribution.

Specifically, we consider the problem of planar quasi-static pushing with a single contact. Modeling the physical process of pushing is difficult as it involves the effects of the object's geometry, friction properties of the surfaces, mass distribution of the object and the pushing forces exerted. These contribute to making the effect of pushing difficult to determine [16]. Various attempts have been made to come up with force-motion models for pushing an object [23, 8], but these models typically require strong assumptions.

In this paper, we argue that it is unnecessary to accurately know the effects of pushing in order to reliably re-position and/or re-orient an object. Furthermore, a small number of observations on the outcomes of pushing is sufficient to learn how to re-position and/or re-orient a novel object. This is similar to how a person performs pushes. The person would push a novel object a few times, and observe the effects on the object. This enables the person to adjust the actions to effectively push the object to the desired configuration. While humans cannot figure out a precise model of pushing dynamics, the history of push interactions gives us a sense of how an object may translate and/or rotate given a push action, which is sufficient for re-positioning and/or re-orientation.

While humans appear to have the ability to predict the outcome of a push on a novel object from observing the outcomes of just few pushes, this is not the entire story. We hypothesize that humans are able to predict the behaviour of a novel object from a small number of pushes only because of our long experience observing the behaviour of other objects. We argue that the same approach will work with robot pushing. We train a recurrent neural network using the behaviour on a number of objects to learn the outcome of pushes. Then the learned network selects actions to re-position and/or re-orient a novel object using a relatively small number of pushes.

On the physical side, insights on why the approach is viable can be obtained from the Voting Theorem [16] which summarizes how an object translates and/or rotates under an applied force. The Voting Theorem says that with the knowledge of centroid of pressure distribution or center of mass (COM), the rotation direction of an object is determined by a vote of three rays I_P, I_L and I_R. I_P is the direction of the applied push. I_L and I_R are the two edges of the friction cone. Conversely, if we know three rays and observe how an object translates and/or rotates over a sequence of pushes, it can give us some hints on where the COM may lie in the body of an object. With a better estimate of the COM, we can approximately apply the Voting Theorem to select actions which push an object towards achieving the goal, and this is usually sufficient for completing the task reliably.

We consider single-contact quasi-static pushing, where inertia effects are negligible. We assume uniform coefficient of friction between the supporting points and the underlying surfaces. However, we do not assume any other knowledge about physical properties of objects, such as center of mass and

pressure distribution. Based on these assumptions, we propose a novel deep recurrent neural network model, Push-Net, which can push novel objects of unknown physical properties for the purpose of re-positioning and/or re-orientation in a 2D plane.

Push-Net is able to select the next push action while explicitly handling history of push interactions. Briefly, inputs to the network are sampled actions, and binary masks of an object in its current state and its target state. The network outputs the scores of all sampled actions. This score measures the similarity between an object's target state and its actual state as a result of applying an action on its current state. Convolutional Neural Network (CNN) layers are used to extract visual representations of masks. We use Long Short-Term Memory (LSTM) model to capture history of push interactions. To select actions for more efficient pushing, we embed physics into the network as an auxiliary learning objective. The network is encouraged to predict the location of the COM of an object on the image plane.

We train the network entirely using simulation data, and perform extensive experiments. Push-Net was able to achieve over 97% average success rate in simulation and succeed in all real robotic experiments with a small number of pushes. The simulation results show the robustness and the efficiency of Push-Net compared to handcrafted and simpler neural network baselines. Real robotic experiments demonstrate the capability of Push-Net to push real objects with unknown physical properties and verify the necessity of push history for robust pushing. Test objects include a subset of YCB dataset [3] and common household objects.

The remainder of the paper is as follows. After discussing related work, we analyze the problem of planar pushing to gain more insights in Section III. We introduce Push-Net in Section IV, followed by experimental evaluation and discussion.

II. Related Work

The study of pushing mechanics aims to predict how the state of an object changes under a push. The methodologies fall into two paradigms: model-based and data-driven. The model-based approaches propose analytic and deterministic models which explain the dynamics of pushing. In order to achieve tractable representation of the dynamics, assumptions and approximations are often adopted in these works [16, 14, 4, 8]. Such simplification inevitably introduces modelling errors which lead to poor generalizability, i.e. the ability to handle objects of different shapes and/or varying physical properties. Both [4, 21] use physics-based simulations to predict the effect of pushing, but this requires them to assume that the simulation environments have similar physical properties as the real ones. In [24], the authors utilize a physics engine to learn physical parameters through black-box Bayesian optimization, but object models need to be known.

To alleviate the effect of modelling errors and increase generalizability, data-driven methods come into handy with big data and increasing computational power. Recent works [11, 13, 22, 20] show promising results in understanding intuitive physics from visual cues using deep learning, such as estimating objects' properties, and predicting stability of block towers. In the realm of manipulation, data driven approaches also help gain generalizability by utilizing large-scale dataset. Both [12, 19] present self-supervise frameworks to collect real robotic grasping data. Their learned policies are able to generalize to grasp novel objects.

The success of deep learning in grasping stimulates research in learning better dynamic model for non-prehensile manipulations. In [6], the authors develop an action-conditioned video prediction model for push interaction. Byravan et al. [2] proposes a SE3-Net to predict 3D rigid motions for constructing future depth images. The next step is to demonstrate the capability of such learned predictive models for control and planning. In [1], they jointly learned forward and inverse models of dynamics using over 400 hours of real robot experience. The inverse model was then used to predict actions given the current and the goal images. [5] collected over 50k unlabeled real robotic pushing attempts using 10 7-Dof arms, and learned a predictive neural network to plan push actions. The experiments show that their method can push novel objects. However, there are two drawbacks in the current data-driven approaches that exploit real robotic data. First, the sample complexity can be arbitrarily large. Second, these approaches pay more attention to visual appearances of objects rather than other critical physical properties, such as pressure distribution and COM. As aforementioned, pushing is an intricate physical process. Considering visual cues only is not sufficient to achieve generality.

To overcome both difficulties, Zhou et al. [23] combined model-based and data-driven approaches for modelling planar friction. They formulated the structural properties of physics principles as constraints for the optimization procedure. The experiments show that their model identification procedure is statistically efficient. However, for every novel object in a new environment, its force-motion model needs to be relearned. Our model is only learned once, and is able to apply to novel objects with unknown physical properties. This is attributed to embedding of history of push interactions in Push-Net. In essence, Push-Net is able to roughly estimate underlying physical properties online. This enables the network to push objects with unknown physical properties. We follow data-driven paradigm, but we train the network with only simulation data. The simulation data contain essential visual cues which are shared by real sensory inputs. This enables the successful transfer of the learned model to real world settings.

III. Problem Analysis

We consider the problem of single-contact quasi-static planar pushing, where inertia effects are negligible. Motion of an object is determined by physical properties of the object, its environment and external push forces. Some physical properties, such as pressure distributions of objects, are statistically indeterminate [8]. This renders pushing a hard problem in robotics. Nonetheless, we human are able to quickly adapt to push novel objects on a daily basis. This could be partially due to our possession of an internal model of physics or

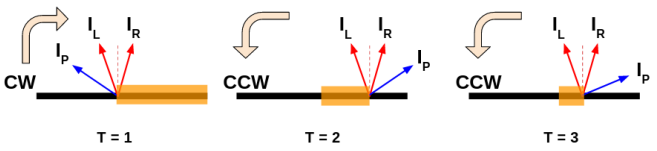


Fig. 2: A toy example: a sequence of push interactions helps determine the COM. I_P is the direction of pushing on the rod (in black); I_L and I_R are the edges of the friction cone. At $T = 1$, the rod is observed to rotate clockwise (CW). By the Voting theorem, the shaded orange region represents possible locations of the COM. At $T = 2$, the rod is observed to rotate counter-clockwise (CCW). The shaded region is hence narrowed by the Voting theorem. At $T = 3$, the possible locations of the COM are further reduced.

intuitive physics [17]. Intuitive physics allow us to reason about physical properties of objects via push interactions. As a result, we are able to push objects more efficiently. This coarse internal model of physics gives us a sense of how an object will translate and/or rotate under an applied force.

The sense of rotation under an applied force has been well summarized by the Voting theorem [16], which says the rays I_P, I_L, and I_R vote on the direction of rotation, with ties resulting in a pure translation. Each vote is determined by the relation of the ray to the object's COM. Given an applied force, the knowledge of COM is essential to decide the sense of rotation. Hence, we are wondering if we observe how an object moves under an applied push, can we roughly estimate where COM is? Better estimate of COM will in turn guide one to push objects more purposefully. Here is a toy scenario supporting the possibility of estimating COM via a sequence of push interactions (Fig. 2). Consider a 1D rod. At time $T = 1$, a push is applied on the center of the rod, with I_P indicating the direction of pushing, and I_L, I_R being the edges of the friction cone. The rod is observed to rotate clockwise. By the Voting theorem, the COM must lie on the right half of the rod (shaded in orange). At $T = 2$, a push is applied at three-quarters along the rod. The rod is observed to rotate counter-clockwise. Again by the Voting theorem, it can be inferred that the COM lies between half and three-quarters along the rod. Similarly, at $T = 3$, we can further narrow down the possible locations of the COM. In short, this toy example shows that some physical properties of objects, such as COM, can be roughly estimated by examining history of push interactions. It is this insight that leads to the design of Push-Net.

IV. Learning to Push Objects with Unknown Physical Properties

We consider the problem of single-contact quasi-static planar pushing for the purpose of re-positioning and/or re-orientation. Formally, let $s_t = \{x_t, y_t, \theta_t\}$ denote the state of an object at time t, where x_t and y_t are the planar position, and θ_t is the orientation. The goal is to push an object from its initial state s_0 to its goal state s_g. At the step t, we generate the next target state s_{t+1} by linear interpolation between s_t and s_g with a fixed step size. The problem now becomes how to push an object from s_t to s_{t+1}. The push transition dynamics is governed by the following equation:

$$s_{t+1} = F(s_t, a_t|P) \tag{1}$$

where $a = \{p_s, p_e\}$ denotes that a pusher moves from a start position to an end position in a 2D plane, and P represents all unknown physical properties that affect push dynamics and remain static with respect to the object of interest. In this work, we focus on one of the static physical properties, COM, which stays the same with respect to a rigid body's origin. As aforementioned, history of push interactions provides information to estimate physical properties, that's $P \approx G(s_0, a_0, ..., s_t)$. The goal is to select the best action that can push an object from s_t closest to s_{t+1}:

$$\begin{aligned} {a_t}^* = \arg\min_{a \in A} & ||s_{t+1} - s_t|| \\ \text{subject to } & COM \text{ is static and} \\ & P \approx G(s_0, a_0, ..., s_t) \end{aligned} \tag{2}$$

where A is a set of all possible actions. To re-position an object, the difference in states is the translational difference (x and y). To re-orient an object, the difference in states is the difference in orientation (θ). To simultaneously re-position and re-orient an object, the difference in states is the pose difference (x, y and θ). Next, we show how to translate this optimization problem with constraints into a neural network.

A. Push-Net Architecture

Push-Net is designed to encode Equation (2). It consists of four main components, which correspond to 1) state representation (s_t and s_{t+1}); 2) actions selection (${a_t}^*$); 3) incorporating history of push interaction ($P \approx G(s_0, a_0, ..., s_t)$); 4) adding physical properties as auxiliary learning objectives (COM is static). The architecture is shown in Fig. 3.

1) State Representation: To represent the current state s_t, we capture an object's pose and shape in an image. Specifically, given an object on a 2D plane, we capture the top-down view of the scene (Fig. 4a). To make state representation invariant to appearances of environment, we perform background subtraction. This gives us a binary mask M_t of size 128×106 containing only object's spatial information and geometry (Fig. 4b). Since the target state s_g may not be achievable with a single push, we generate intermediate s_{t+1} by linear interpolation between s_t and s_g. Concretely, to re-position an object by Δx and Δy meters, the goal mask M_g is first generated by translating the initial mask M_0 by $C\Delta x$ and $C\Delta y$ in the image space, where C is the pixels-per-metric ratio. C can be obtained by calibration with a reference object of known size. At each step, we calculate the direction vector $\vec{d_t}$ between the centers of the object in M_t and M_g. The next mask M_{t+1} is generated by translating M_t in the direction of $\vec{d_t}$ by a fixed amount of $2.5cm$. Similarly, to re-orient an object, M_g is generated by rotating M_0 by $\Delta\theta$. The desired direction of rotation is determined by rotational difference between M_t and M_g. Rotating M_t in the desired direction by 5° gives M_{t+1}. Both M_t and M_{t+1} are parts of inputs to Push-Net. Four layers of CNN are used to extract the latent features f_t and f_{t+1} corresponding to M_t and M_{t+1} respectively.

2) Action Selection: A push action is specified by the tuple (p_s, p_e). A pusher (a sphere of 4cm in diameter in simulation) moves from the start pixel p_s to the end pixel p_e on the

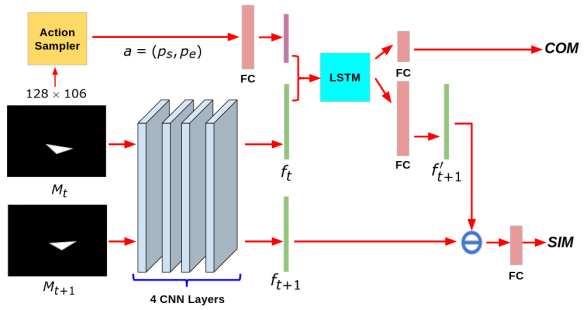


Fig. 3: Push-Net architecture. The inputs to Push-Net are the current mask M_t, the target mask M_{t+1} and the sampled action $a = (p_s, p_e)$. The outputs of Push-Net are predicted COM and similarity score (SIM) which measures the Euclidean distance between the underlying states of the target M_{t+1} and the mask generated as a result of applying a to M_t. Push-Net selects the best action which has the smallest SIM among all sampled actions. All four CNN layers have the filter size of 3×3. Each CNN layer is followed by a max pooling layer (omitted in the figure) of size 3×3. The number of channels in four CNN layers are 16, 16, 32, 32 respectively. The weights of CNN layers are shared by M_t and M_{t+1}. The outputs of the last CNN layer is flattened to form f_t and f_{t+1}. The input size and the hidden size of LSTM are both 80. LSTM has only one layer. FC represents Fully Connected layer. The blue minus sign returns the absolute element-wise difference between f'_{t+1} and f_{t+1}.

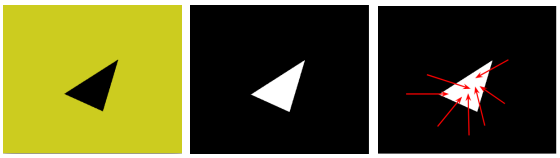


Fig. 4: (a) The camera view in DART simulator. (b) Segmented mask image. (c) The red arrows indicate sampled actions.

image plane. Clearly, the action space is humongous (over 10^8). Hence, rather than learning a network which directly predicts the best action as the output [1] (a harder problem), we design Push-Net to select the best action among a set of candidates (inputs to Push-Net)) based on certain scores. Similar to [15] where they select actions based on predicted grasp quality, we select actions based on how close an action can push an object from its s_t to s_{t+1}. To sample a candidate action, we first uniformly sample a pixel p_e among all pixels belonging to the object in M_t. We then randomly sample a line segment l_e with one end being p_e. If the other end of l_e falls in the background of M_t, then it deems as p_s. Otherwise, we repeat sampling another line segment until p_s is obtained. For each action candidate (Fig. 4c), Push-Net outputs a score (SIM), a vector of size three, which measures the Euclidean distance between the desired s_{t+1} and the actual next state s'_{t+1} as a result of applying an action on s_t. Specifically, SIM = $(sim_x, sim_y, sim_\theta)$, which corresponds to the distance measures in x, y and θ dimension. For re-positioning tasks, only sim_x and sim_y are used to select the best action. For re-orientation tasks, only sim_θ is used. For simultaneous re-positioning and re-orientation, all three dimensions are employed to select the best action.

3) History of Push Interactions: In Fig. 3, $P \approx G(s_0, a_0, ..., s_t)$ is captured by the cell state in LSTM. At each step, the input to LSTM is a feature vector of size 80, which encodes the current mask M_t and an action candidate a. Over time, the cell state accumulates history of push interactions. The output of LSTM is decoded to f'_{t+1}, which is the predicted next internal state representation. Finally, we take the difference between f'_{t+1} and f_{t+1} to obtain predicted SIM score for a given M_t and M_{t+1}. In training phase, we keep the maximum push sequence length to be 10. During testing, there is no restriction on the length of a sequence.

4) Static Physical Properties as Auxiliary Learning Objectives: As illustrated in Section III, history of push interactions contains critical information about an object's physical properties. In turn, better understanding of those physical properties can help select actions to push objects more purposefully. Here, we consider one of the static properties of objects, i.e. COM. A rigid body's COM is always fixed relative to its own origin. We exploit this prior knowledge as an auxiliary learning objective in Push-Net. Particularly, the output of LSTM is decoded to predict where COM will be on the image plane upon applying the action a to M_t. The ground truth labels of COMs are obtained from the simulator. This auxiliary learning objective forces LSTM to be able to discover COM, and hopefully can help select actions to push objects more purposefully. As we show the results in Section V-B, it is indeed the case.

B. Data Preparation

We use DART [10] as the simulator which provides realistic physics simulation. We use blender to randomly generate objects' meshes, which can be imported to DART. The training set contains 60 objects, among which 30 of them are convex prisms with n-side polygonal base ($n \in \{3, 4, 5\}$), and the rest are prisms with ellipse base. The test set contains 12 objects, of which 6 are convex prisms with n-side polygonal base ($n \in \{3, 4, 5, 6, 7\}$), and the other 6 are prisms with ellipse base. Some example of objects' meshes are shown in Fig. 5.

We set up a simulated camera in DART to capture the motion of objects under the effect of applied pushes. The camera is facing perpendicular to the supporting surface. We adjust the camera's field of view so that it produces an image of size 512×424. For each object in the training set, we uniformly sample 20 different COMs within the object's geometry. Under each COM, we position the object at the center of the image plane, and we uniformly sample 20 initial orientations. For each initial orientation, we perform 20 sequences of pushes. Each sequence has 10 steps. At each step, a push is generated in the same way as described in Section IV-A2. For each step, we include the following information in training data: 1) current mask M_t; 2) current pose of the object s_t; 3) push action taken (p_s, p_e); 4) resultant mask M_{t+1}; 5) resultant pose of the object s_{t+1}; 6) COMs of the object before and after the push. All images are downsized to 128×106 for training. In total, we collected 4.8×10^5 sequences of pushes.

As can be noticed, when we treat M_{t+1} as the target mask for M_t, SIM is always a zero vector, because s_{t+1} is exactly the resultant state transited from s_t upon an action (p_s, p_e). To create examples where SIM is non-zero, we augment the

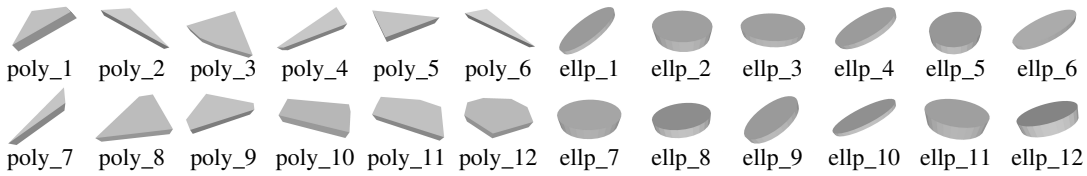


Fig. 5: First row: a subset of the training object meshes; Second row: the testing object meshes.

data by randomly pairing M_t (current mask) with M_k (target mask) for each recorded sequence, where $1 \leq t, k \leq 10$. Such random pairings are performed four times for each sequence. Hence, the augmented dataset contains 2.4×10^6 sequences of pushes with a 80:20 train/test split.

V. Experiments and Discussion

For experimental evaluation, three pushing tasks are considered: re-position an object; re-orient an object; simultaneously re-position and re-orient an object. For re-positioning, a goal is specified by how much an object needs to be translated in x and y axes respectively, i.e. $(\Delta x, \Delta y)$. For re-orientation, a goal is specified by how much an object needs to be rotated around z axis, i.e. $\Delta\theta$. For concurrent re-positioning and re-orientation, a goal is specified in Δx, Δy and $\Delta\theta$.

We aim to answer the following questions: 1) Does the use of history help push objects with unknown physical properties robustly to its goal? 2) Does adding COM as an auxiliary learning objective help Push-Net learn a better push policy? 3) Can we use Push-Net to push novel objects with unknown physical properties? To answer these questions, we conduct quantitative experiments and qualitative analysis in both simulation and real world.

To answer question 1), we compare Push-Net to two baselines. One is a greedy reactive policy (GRP). GRP assumes an object's COM coincides with its geometric center at (x_c, y_c). For re-positioning, at each step, GRP performs a fixed length $(2.5cm)$ push through an object's geometric center towards the goal position. For re-orientation, GRP aspires to push an object to rotate in the desired direction for 5° per step. With GRP, we want to verify that pushing objects with unknown physical properties is not a trivial problem, and handcraft policies are not robust. The other baseline is a version of Push-Net without memory (Push-Net-nomem). All other parts being the same, Push-Net-nomem does not possess a LSTM module to keep track of history. It also omits prediction of COM, and only outputs SIM score. With Push-Net-nomem, we want to show that history of push interactions is critical.

To answer question 2), we compare Push-Net to Push-Net-sim, a version of Push-Net without being regularized by COM prediction. Push-Net-sim outputs SIM score only. This is to verify the necessity of incorporating COM as an auxiliary learning objective to facilitate training.

To answer question 3), we extensively test Push-Net on a set of novel objects with unknown physical properties in both simulation and real robot experiments.

A. Training

We implemented three networks (Push-Net, Push-Net-sim, Push-Net-nomem) using the deep learning framework PyTorch [18]. The loss functions for Push-Net-sim and Push-Net-nomem are Mean Squared Error (MSE) between predicted SIM scores and true SIM scores, $L_{SIM} = MSE(SIM', SIM)$. For Push-Net, we add MSE between predicted COM and true COM as an auxiliary loss function for weight regularization, $L_{COM} = MSE(COM', COM)$. We craft the loss function of the following form:

$$L = C_1 L_{COM} + \frac{C_2 L_{SIM}}{1 + C_3 L_{COM}} \tag{3}$$

where C_1, C_2 and C_3 are constants. The first term minimizes prediction error in COM, while the second term minimizes prediction error in SIM. The denominator in the second term forces the loss in SIM to be small when the loss in COM is already small. The rationale is that a better estimate of COM helps to predict SIM more accurately. We trained our networks using ADAM optimization method [9] with default hyper-parameters. We initialize weights of CNN layers using Xavier initialization [7]. Cell states and hidden states in LSTM are initialized to zeros. We trained each network for 200 epochs with batch size of 16. The training roughly takes half a day on an NVIDIA GTX 1080 Ti GPU.

B. Simulation Results

A simulation experiment is carried out as follows. For an object, we first uniformly sample a point inside the object, and set this point to be its COM in DART simulator. We initialize the object to be in the center of the image plane with a random orientation. The simulated camera captures the scene, from which the current mask image M_0 is obtained by background subtraction. For the task of re-positioning, we sample a goal, both Δx and Δy, from $U(-0.3m, 0.3m)$. For the task of re-orientation, $\Delta\theta$ is sampled from $U(-90^\circ, 90^\circ)$ to be a goal. Given a goal, we obtain the goal mask M_g and the next target mask M_1 in the same way as described in Section IV-A1. From M_0, we sample 1000 action candidates. Together with M_0 and M_1, action candidates are fed into Push-Net to output a list of scores for all action candidates. We pick the action a^* with the best score and execute this action in DART. To update the cell state in LSTM, we feed M_0, M_1 and the selected action a^* again to Push-Net. This process repeats until success or terminal condition is met. The experiment is deemed as successful if an object can be pushed within its goal region in 25 steps, otherwise failure. The goal region is $\pm 5cm$ for re-positioning and $\pm 10^\circ$ for re-orientation. We report two evaluation metrics: success rate to measure robustness and average number of steps to measure efficiency.

We perform evaluation on a random subset of the training set (12 objects), and 12 objects in the testing set (Fig. 5). For

TABLE I: Results of re-positioning tasks: average success rate / average number of steps

(a) Prisms with polygonal base.

objects	GRP	Push-Net-nomem	Push-Net-sim	Push-Net
poly_1	0.95±0.00 / 11.00±0.00	0.99±0.02 / 13.14±0.35	0.96±0.04 / 12.33±0.47	**0.99±0.02 / 9.67±1.11**
poly_2	0.75±0.00 / 11.00±0.00	0.75±0.05 / 16.43±0.49	1.00±0.00 / 10.00±0.00	**1.00±0.00 / 8.00±1.41**
poly_3	1.00±0.00 / 11.00±0.00	0.99±0.02 / 12.57±0.49	0.92±0.09 / 10.17±1.67	**1.00±0.00 / 9.33±0.47**
poly_4	0.79±0.00 / 14.00±0.00	0.90±0.04 / 15.86±0.35	0.88±0.06 / 13.67±0.94	**1.00±0.00 / 9.67±0.47**
poly_5	**1.00±0.00 / 8.00±0.00**	1.00±0.00 / 11.57±0.49	1.00±0.00 / 9.67±0.47	0.98±0.04 / 9.83±0.37
poly_6	0.80±0.00 / 12.00±0.00	0.89±0.07 / 15.86±0.83	0.97±0.04 / 11.00±0.82	**1.00±0.00 / 9.67±0.75**
poly_7	0.70±0.00 / 14.00±0.00	0.87±0.04 / 15.00±0.53	0.92±0.04 / 12.67±0.75	**0.96±0.06 / 9.50±1.80**
poly_8	**1.00±0.00 / 8.00±0.00**	1.00±0.00 / 12.00±0.00	0.99±0.02 / 8.83±0.69	0.98±0.02 / 9.33±1.37
poly_9	1.00±0.00 / 12.00±0.00	1.00±0.00 / 11.86±0.64	1.00±0.00 / 10.33±0.47	**1.00±0.00 / 9.50±0.50**
poly_10	0.95±0.00 / 10.00±0.00	1.00±0.00 / 10.57±0.49	1.00±0.00 / 9.17±0.69	**1.00±0.00 / 8.67±0.47**
poly_11	0.95±0.00 / 12.00±0.00	1.00±0.00 / 14.00±0.00	1.00±0.00 / 10.50±0.50	**1.00±0.00 / 9.70±0.62**
poly_12	1.00±0.00 / 10.00±0.00	1.00±0.00 / 11.57±0.49	1.00±0.00 / 10.33±0.47	**1.00±0.00 / 8.83±1.21**

(b) Prisms with ellipse base.

objects	GRP	Push-Net-nomem	Push-Net-sim	Push-Net
ellp_1	**1.00±0.00 / 8.00±0.00**	1.00±0.00 / 10.57±0.49	1.00±0.00 / 9.67±0.47	1.00±0.00 / 9.17±0.37
ellp_2	**1.00±0.00 / 7.00±0.00**	0.98±0.02 / 12.43±0.49	1.00±0.00 / 9.67±0.47	1.00±0.00 / 8.67±1.37
ellp_3	**1.00±0.00 / 7.00±0.00**	1.00±0.00 / 11.14±0.35	1.00±0.00 / 9.00±0.00	1.00±0.00 / 8.50±0.50
ellp_4	**1.00±0.00 / 7.00±0.00**	1.00±0.00 / 9.86±0.35	1.00±0.00 / 8.83±0.69	1.00±0.00 / 8.00±0.00
ellp_5	**1.00±0.00 / 7.00±0.00**	1.00±0.00 / 10.00±0.00	1.00±0.00 / 9.00±0.00	1.00±0.00 / 7.00±1.15
ellp_6	**1.00±0.00 / 8.00±0.00**	1.00±0.00 / 12.00±0.00	1.00±0.00 / 9.00±0.00	1.00±0.00 / 9.17±0.37
ellp_7	**1.00±0.00 / 8.00±0.00**	0.93±0.03 / 12.71±0.45	1.00±0.00 / 9.00±0.00	0.99±0.02 / 9.50±0.50
ellp_8	**1.00±0.00 / 8.00±0.00**	0.96±0.02 / 13.00±0.00	1.00±0.00 / 9.83±0.37	1.00±0.00 / 9.00±0.00
ellp_9	**1.00±0.00 / 7.00±0.00**	1.00±0.00 / 10.00±0.00	1.00±0.00 / 9.00±0.00	1.00±0.00 / 8.50±0.96
ellp_10	**1.00±0.00 / 9.00±0.00**	0.97±0.04 / 14.57±0.73	1.00±0.00 / 11.00±0.00	1.00±0.00 / 10.00±0.82
ellp_11	**1.00±0.00 / 7.00±0.00**	1.00±0.00 / 11.57±0.49	1.00±0.00 / 9.17±0.37	1.00±0.00 / 9.17±0.37
ellp_12	**1.00±0.00 / 6.00±0.00**	1.00±0.00 / 10.29±0.45	1.00±0.00 / 8.83±0.37	1.00±0.00 / 8.33±0.47

TABLE II: Results of re-orientation tasks: average success rate / average number of steps

(a) Prisms with polygonal base.

objects	GRP	Push-Net-nomem	Push-Net-sim	Push-Net
poly_1	0.83±0.02 / 7.00±0.00	0.54±0.07 / 12.86±0.83	**1.00±0.00 / 7.60±0.49**	0.97±0.02 / 8.67±0.47
poly_2	1.00±0.00 / 4.00±0.00	0.70±0.10 / 12.86±1.25	1.00±0.00 / 7.20±0.75	**1.00±0.00 / 7.00±0.00**
poly_3	0.77±0.02 / 9.33±0.47	0.81±0.07 / 10.00±0.93	0.93±0.03 / 10.67±0.94	**0.98±0.02 / 10.60±0.49**
poly_4	0.65±0.00 / 11.00±0.00	0.64±0.09 / 13.00±1.07	0.97±0.02 / 9.80±0.40	**1.00±0.00 / 9.67±0.47**
poly_5	0.43±0.02 / 15.67±0.47	0.78±0.06 / 11.86±1.55	0.92±0.08 / 11.00±0.82	**0.98±0.02 / 9.80±0.75**
poly_6	0.98±0.02 / 7.00±0.00	0.40±0.07 / 18.00±0.76	1.00±0.00 / 9.00±0.63	**1.00±0.00 / 8.33±0.47**
poly_7	**0.98±0.02 / 6.67±0.47**	0.50±0.09 / 16.00±1.07	0.92±0.02 / 10.67±0.47	0.96±0.04 / 9.40±0.49
poly_8	0.80±0.05 / 9.33±0.47	0.49±0.06 / 14.86±0.64	0.93±0.05 / 10.33±1.70	**0.96±0.04 / 9.20±0.98**
poly_9	0.77±0.02 / 10.00±0.00	0.67±0.04 / 11.00±0.76	**0.95±0.03 / 8.00±0.63**	0.95±0.04 / 8.67±0.47
poly_10	0.70±0.04 / 10.00±0.00	0.67±0.05 / 11.43±1.29	0.92±0.02 / 8.33±0.47	**1.00±0.00 / 7.60±0.49**
poly_11	0.67±0.02 / 12.00±0.00	0.66±0.06 / 13.86±0.99	0.95±0.04 / 11.00±0.82	**0.98±0.04 / 10.00±0.63**
poly_12	0.50±0.00 / 13.33±0.47	0.75±0.08 / 9.43±1.50	0.95±0.03 / 9.40±0.49	**0.97±0.02 / 8.33±0.47**

(b) Prisms with ellipse base.

objects	GRP	Push-Net-nomem	Push-Net-sim	Push-Net
ellp_1	**1.00±0.00 / 6.33±0.47**	0.81±0.04 / 10.43±0.73	1.00±0.00 / 9.20±0.98	1.00±0.00 / 8.67±0.47
ellp_2	0.49±0.04 / 13.33±0.47	0.83±0.03 / 8.57±0.49	1.00±0.00 / 7.80±0.40	**1.00±0.00 / 7.00±0.00**
ellp_3	0.63±0.06 / 12.67±0.47	0.89±0.08 / 7.86±0.99	1.00±0.00 / 9.60±0.49	**1.00±0.00 / 8.00±0.00**
ellp_4	0.92±0.02 / 7.00±0.82	0.85±0.07 / 9.00±1.20	**1.00±0.00 / 9.00±0.00**	**1.00±0.00 / 9.00±0.00**
ellp_5	0.63±0.02 / 11.33±0.47	0.92±0.05 / 5.86±0.99	1.00±0.00 / 7.80±0.40	**1.00±0.00 / 6.00±0.00**
ellp_6	0.85±0.04 / 10.67±0.47	0.81±0.08 / 13.57±1.84	1.00±0.00 / 9.80±0.40	**1.00±0.00 / 9.33±0.47**
ellp_7	0.67±0.02 / 11.00±0.00	0.85±0.06 / 11.14±0.35	0.99±0.02 / 10.20±0.40	**1.00±0.00 / 9.33±0.47**
ellp_8	0.45±0.12 / 15.67±1.25	0.80±0.08 / 9.57±1.18	**0.99±0.02 / 9.00±0.63**	0.98±0.02 / 9.67±0.94
ellp_9	0.58±0.02 / 13.00±0.00	0.81±0.07 / 10.29±1.83	1.00±0.00 / 8.20±0.40	**1.00±0.00 / 8.00±0.00**
ellp_10	0.40±0.07 / 17.33±0.47	0.86±0.07 / 10.43±1.29	1.00±0.00 / 10.00±0.00	**1.00±0.00 / 8.67±1.25**
ellp_11	0.33±0.02 / 17.33±0.47	0.69±0.08 / 12.29±1.03	0.90±0.04 / 11.67±0.47	**0.96±0.02 / 10.40±0.32**
ellp_12	0.69±0.05 / 10.67±0.47	0.74±0.07 / 10.43±1.18	0.99±0.02 / 8.60±0.49	**1.00±0.00 / 8.00±0.00**

TABLE III: Results of re-positioning and re-orientation tasks: average success rate / average number of steps

(a) Prisms with polygonal base.

objects	GRP	Push-Net-nomem	Push-Net-sim	Push-Net
poly_1	0.47±0.02 / 17.01±1.24	0.45±0.04 / 13.01±0.53	0.90±0.04 / 11.63±0.36	**0.95±0.00 / 10.61±0.04**
poly_2	0.45±0.04 / 14.51±0.51	0.52±0.08 / 16.37±0.73	0.97±0.05 / 10.23±0.94	**0.97±0.05 / 9.53±0.50**
poly_3	0.37±0.02 / 11.99±0.81	0.35±0.04 / 10.22±0.63	0.97±0.02 / 9.49±0.23	**0.98±0.02 / 9.15±0.72**
poly_4	0.32±0.02 / 15.13±0.46	0.43±0.02 / 13.69±0.75	0.90±0.00 / 11.15±0.92	**0.92±0.06 / 10.07±0.53**
poly_5	0.28±0.02 / 13.17±2.66	0.35±0.11 / 10.99±2.46	0.83±0.12 / 11.81±0.40	**0.93±0.02 / 12.49±0.50**
poly_6	0.38±0.05 / 14.34±1.52	0.32±0.08 / 15.51±0.81	0.88±0.05 / 11.16±0.32	**0.92±0.05 / 11.18±1.02**
poly_7	0.43±0.02 / 15.62±0.76	0.32±0.09 / 12.96±1.25	0.88±0.05 / 11.82±0.92	**0.93±0.06 / 11.97±0.37**
poly_8	0.50±0.07 / 13.04±0.81	0.42±0.14 / 10.97±1.27	0.98±0.02 / 8.11±0.15	**1.00±0.00 / 7.53±0.42**
poly_9	0.28±0.05 / 11.19±1.68	0.47±0.02 / 12.69±0.88	0.92±0.06 / 10.63±1.09	**0.97±0.02 / 10.26±0.31**
poly_10	0.45±0.00 / 10.44±0.42	0.87±0.02 / 11.47±0.68	**1.00±0.00 / 8.13±0.50**	0.98±0.02 / 8.22±0.83
poly_11	0.37±0.02 / 16.77±0.83	0.50±0.07 / 13.49±0.80	0.95±0.04 / 9.23±0.42	**0.98±0.02 / 7.96±0.48**
poly_12	0.15±0.04 / 14.25±1.43	0.68±0.02 / 10.34±0.37	0.85±0.04 / 8.79±0.36	**0.92±0.05 / 9.22±0.23**

(b) Prisms with ellipse base.

objects	GRP	Push-Net-nomem	Push-Net-sim	Push-Net
ellp_1	0.63±0.05 / 16.74±0.46	0.65±0.04 / 10.09±0.87	0.95±0.00 / 8.02±0.74	**1.00±0.00 / 5.83±0.31**
ellp_2	0.17±0.02 / 17.47±0.34	0.47±0.04 / 13.94±0.40	0.88±0.05 / 10.28±0.60	**0.90±0.04 / 11.98±1.36**
ellp_3	0.19±0.02 / 8.53±1.56	0.56±0.04 / 14.06±1.54	**0.93±0.02 / 11.27±0.34**	0.92±0.05 / 8.99±0.85
ellp_4	0.73±0.05 / 11.96±0.08	0.82±0.02 / 10.94±0.77	0.97±0.02 / 7.26±0.63	**0.98±0.02 / 6.30±0.24**
ellp_5	0.15±0.00 / 5.44±0.63	0.53±0.02 / 11.10±1.35	0.95±0.08 / 8.15±0.25	**0.97±0.01 / 8.90±0.17**
ellp_6	0.22±0.02 / 12.92±1.48	0.75±0.04 / 13.71±1.01	0.93±0.06 / 7.52±0.81	**0.98±0.02 / 9.85±0.67**
ellp_7	0.57±0.05 / 10.54±0.53	0.57±0.06 / 12.67±1.03	0.85±0.00 / 7.67±0.08	**0.92±0.02 / 6.94±0.24**
ellp_8	0.25±0.00 / 15.33±0.41	0.45±0.04 / 13.29±1.33	**0.95±0.04 / 11.02±0.71**	0.92±0.01 / 9.94±0.87
ellp_9	0.17±0.02 / 11.39±0.80	0.63±0.02 / 9.96±0.26	0.88±0.02 / 9.52±1.18	**0.95±0.02 / 9.45±0.62**
ellp_10	0.15±0.00 / 19.00±0.54	0.58±0.05 / 14.10±2.07	0.83±0.05 / 10.33±0.33	**0.94±0.02 / 9.11±0.97**
ellp_11	0.23±0.00 / 4.33±0.47	0.53±0.02 / 14.61±0.89	0.93±0.02 / 11.58±0.80	**0.97±0.02 / 11.22±1.59**
ellp_12	0.52±0.06 / 13.21±0.71	0.65±0.04 / 13.21±0.67	0.97±0.05 / 9.17±0.80	**0.98±0.02 / 8.68±0.70**

each object, we sample 5 COMs and 4 initial orientations. Due to the random nature of action sampling, we repeat these 20 experiments for 10 times. In total, we perform 200 experiments per object. For fair comparison, the configurations (the initial poses of objects and the goals) of all 200*24 experiments are kept the same for all methods. We summarize results for three tasks in Table I, II and III respectively.

Each entry in the table reports the average success rate and the average number of steps with their corresponding standard deviation. For all tasks, we prioritize robustness over efficiency. Hence, the bold entry for each row represents the method with the highest average success rate. If there is a tie in the average success rate, the entry with fewer average number of steps is favored. The average number of steps is calculated based on successful cases only. Overall, Push-Net is the most robust policy with 97.88% average success rate, followed by Push-Net-sim (95.83%), Push-Net-nomem (74.40%) and GRP (66.72%). Push-Net is also the most efficient policy with 9.05 average number of steps, followed by Push-Net-sim (9.70), GRP (10.97) and Push-Net-nomem (12.50).

To answer question 1), we first take a look at two columns corresponding to Push-Net-nomem and Push-Net. It is clear that Push-Net outperforms Push-Net-nomem in all cases except for one (Table Ia row poly_8). In many cases, Push-Net gains by a large margin in term of average success rate. This confirms that LSTM is effective to utilize history of push interactions. As a result, Push-Net is able to push objects with unknown physical properties robustly.

We now look at two columns of GRP and Push-Net. Overall, Push-Net shows its robustness across all tasks and all objects, while GRP's performance varies a lot. For tasks of re-positioning, interestingly, GRP outperforms all other methods for prisms with ellipse base (Table Ib) but it is not the case for prisms with polygonal base. We hypothesize that GRP works effectively for objects of symmetric geometries such as ellipses. Further evidence can be found in Table Ia and IIa: GRP performs better for poly_3, poly_5, poly_8, poly_9 and poly_12, which have near symmetric geometries, while GRP suffers for non-symmetric objects.

Another observation is that the success rate of GRP is higher for re-positioning tasks than for re-orientation tasks, and it performs the worst for concurrent re-positioning and re-orientation tasks. This is the expected behavior of GRP. For re-positioning, GRP pushes an object through its geometric center toward its goal by $2.5cm$ every step. Despite unknown physical properties, an object's resultant position will not deviate much from the intended position. In contrast, for re-orientation, the resultant change in orientation depends largely on the location of COM. The further COM is away from the contact point, the greater the change in orientation will be. This creates the problem of overshooting in orientation. As a result, GRP fails to push an object to its desired orientation. To verify, we examine some of the failure cases. Indeed, GRP pushes objects to overshoot the intended orientation and then pushes in the reverse direction to recover the orientation, but again it overshoots. Apparently, the entanglement of re-positioning and re-orientation makes it even harder for GRP to achieve the goal as showed in Table IIIa and IIIb. Hence, it is often hard to handcraft a policy which can push objects robustly with unknown physical properties.

Question 2) concerns whether adding COM as an auxiliary learning objective helps learn a more efficient policy. We

TABLE IV: Results of re-positioning real objects: success rate / average number of steps

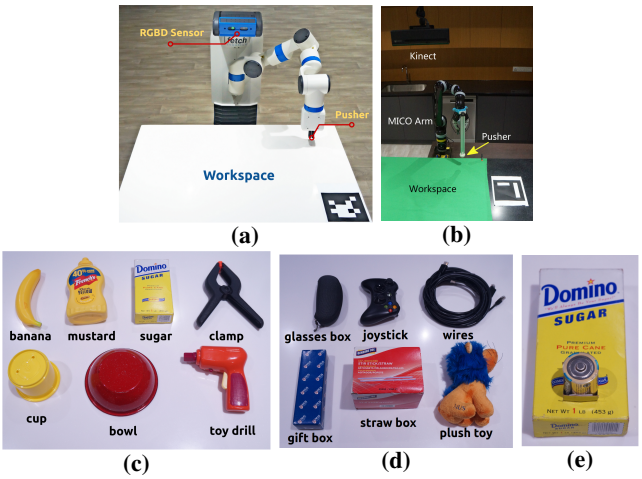
objects	straw box	cup	glasses box	clamp	plush toy	wires	bowl
Push-Net	1.00 / 5.00	1.00 / 4.40	1.00 / 4.90	1.00 / 7.00	1.00 / 6.40	1.00 / 4.80	1.00 / 4.50
Push-Net-nomem	0.50 / 7.20	0.70 / 8.14	0.60 / 7.33	0.30 / 6.00	0.30 / 7.33	0.50 / 6.60	0.70 / 8.14

TABLE V: Results of re-orienting real objects: success rate / average number of steps

objects	mustard	joystick	banana	gift box	toy drill	sugar	sugar with shifted COM
Push-Net	1.00 / 6.20	1.00 / 3.20	1.00 / 6.50	1.00 / 6.40	1.00 / 4.50	1.00 / 4.60	1.00 / 4.20
Push-Net-nomem	0.40 / 6.50	0.50 / 6.00	0.70 / 4.14	0.60 / 6.17	0.60 / 3.67	0.80 / 6.25	0.70 / 5.43

TABLE VI: Results of re-positioning and re-orienting real objects: success rate / average number of steps

objects	mustard	joystick	banana	gift box	toy drill	sugar	sugar with shifted COM
Push-Net	1.00 / 6.40	1.00 / 7.00	1.00 / 8.20	1.00 / 7.90	1.00 / 6.30	1.00 / 6.50	1.00 / 6.30
Push-Net-nomem	0.40 / 6.75	0.20 / 6.00	0.60 / 7.00	0.60 / 6.33	0.60 / 5.67	0.80 / 7.62	0.50 / 6.40



(a) (b) (c) (d) (e)

Fig. 6: (a) A Fetch robot is equipped with a RGBD head camera; its gripper tips serve as the pusher; the robot pushes objects in the workspace. (b) An extended bar is attached to a MICO arm as the pusher; A Kinect2 sensor provides RGBD data. (c) A subset of YCB dataset. (d) Some common household objects. (e) The sugar box with non-centered COM. A heavy battery is inserted into the sugar box to shift its COM towards its bottom.

compare Push-Net with Push-Net-sim. Both methods achieve high success rates across tasks and objects. Overall, Push-Net is more efficient as it takes fewer steps than Push-Net-sim to push objects to its goal. We argue that this is because embedding prior knowledge (COM) into the network helps better estimate SIM. Consequently, better estimation of SIM helps to select more efficient actions. To support this argument, we measure the average prediction errors in SIM per step for Push-Net and Push-Net-sim. For re-positioning, the errors from Push-Net-sim are 11.19% and 4.54% larger than that of Push-Net for polygonal bases and ellipse bases respectively. For re-orientation, the errors from Push-Net-sim are 14.25% and 7.95% larger than that of Push-Net for polygonal bases and ellipse bases. For concurrent re-positioning and re-orientation, the errors difference are 12.40% and 3.25% respectively.

Question 3) pertains to verify if Push-Net can be generalized to push objects of novel shapes and unknown physical properties. This is partly substantiated by the results on testing objects in Table I, II and III. Push-Net obtains consistently high average success rates on the testing objects. Finally, we demonstrate the capability of Push-Net to push novel objects with unknown physical properties with a real robot system.

C. Real Robot Experiments

We perform extensive real robot experiments on novel objects. The setup is shown in Fig. 6a. The Fetch's gripper tip serves as the pusher. We select 7 objects from YCB dataset and 6 common household objects for the evaluation of Push-Net. For each object, we perform 10 experiments for re-positioning and/or re-orientation tasks. The task of re-orientation is meaningless for near circular objects, such as the cup. Hence, we only consider the task of re-positioning for these objects. An object is randomly placed in the workspace initially. Positional goals in x or y direction are sampled uniformly from $(-0.4m, -0.2m) \cup (0.2m, 0.4m)$. Rotational goals are sampled uniformly from $(-90°, -30°) \cup (30°, 90°)$. We use Fetch's head RGBD camera to capture the workspace. We segment an object's point cloud to obtain its mask at each step. We report the success rate and average number of steps among the successful trials in Table IV, V and VI.

As shown in the tables, Push-Net can be successfully generalized to efficiently push real novel objects with unknown physical properties for all tasks. We also compare Push-Net with Push-Net-nomem to confirm the necessity of incorporating history of push interactions for robustly pushing real objects. Push-Net-nomem struggles across all test cases and on average takes more steps than Push-Net to achieve the goal.

Push-Net is also able to handle objects with non-uniform mass distribution. We manually modified the COM of the sugar box (Fig. 6e) by adding a heavy battery to shift its COM towards one side. The results show that Push-Net was able to robustly push the sugar box with varying COMs, whereas the performance of Push-Net-nomem degrades.

Interestingly, Push-Net also works for some objects of concave geometry, such as the clamp, the toy drill and the joystick, while Push-Net was trained using only convex objects. We show three qualitative results in Fig. 7, 8 and 9. Fig. 7 shows pushing a pile of wires to a goal position. Despite the imperfection in the object's mask, Push-Net was robust enough to push it to the goal region. In Fig. 8, the robot was pushing to re-orient a banana. It is worth noticing that at step 1, Push-Net pushes aggressively to change the banana's orientation, whereas at step 4 and 5, Push-Net gently pushes the banana to fine tune its orientation so that it won't overshoot the goal. In Fig. 9, Push-Net was able to simultaneously re-position and re-orient the toy drill to its goal configuration with just 6 pushes. Push-Net also works equally well across different robotic platforms. A Kinova MICO robotic arm (Fig. 6b) equipped with Push-Net capability is able to push objects robustly and efficiently to reach goals. More qualitative results on the Fetch robot and the MICO arm can be found in the accompany video at: **https://tinyurl.com/rss2018pushnet**.

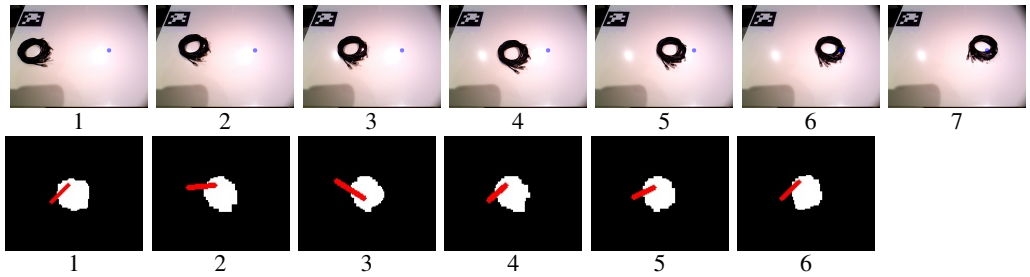


Fig. 7: Re-position a pile of wires. The first row lists the RGB images from the head camera at each step. The blue dot indicates the goal position on the image plane. The second row lists the centered object mask at each step with the red line segment being the push actions.

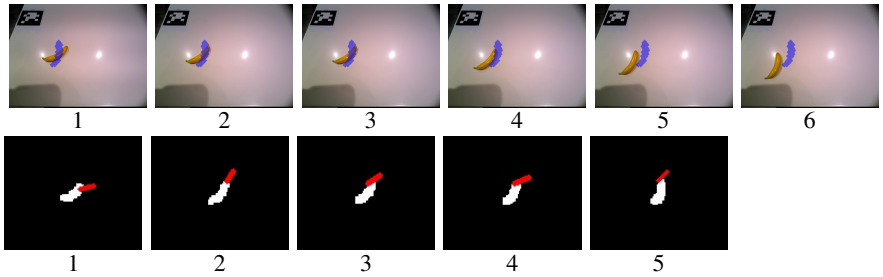


Fig. 8: Re-orient a banana. The first row lists the sequence of RGB images. The blue shaded shape indicates the goal orientation of the banana. The second row is the sequence of the object's masks with corresponding push actions.

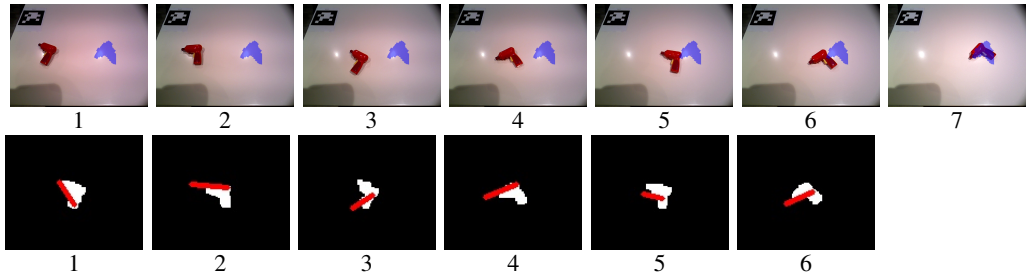


Fig. 9: Re-position and re-orient a toy drill. The first row lists the sequence of RGB images. The blue shaded shape indicates the goal configuration of the drill. The second row is the sequence of the object's masks with corresponding push actions.

D. Discussion

While the experimental results demonstrate the robustness, the efficiency and the generalizability of Push-Net, there are a few areas for improvement. First, action sampler can be improved by incorporating prior knowledge and domain constraints. For example, given a cup with handle, it is more effective to rotate the cup by pushing its handle. Such prior knowledge can guide to sample more effective actions compared with randomly sampling actions. Second, although Push-Net was only trained from convex-shape objects, it was able to push some real objects of concave geometries. It will be beneficial to understand the underlying reasons. This can potential help discover critical features in this problem domain. Third, in this work, we only deal with pushing a single object on a plane. In many cases, there are more than one objects present. Hence, how to push objects through clutter and handle interactions among objects remains as our future work.

VI. Conclusion

We presented Push-Net, a deep recurrent neural network model for planar pushing novel objects with unknown physical properties. By capturing history of push interactions in the network, Push-Net is able to re-position and/or re-orient objects with unknown physical properties robustly. We trained Push-Net only using simulation data. Extensive simulation experiments demonstrate the superiority of Push-Net over other baselines. Experimental results also shows that incorporating physics, COM, as an auxiliary learning objective helps train the network to discover more efficient push policies. We also conducted real robot experiments on a subset of YCB dataset and some common household objects. The result shows that Push-Net can be generalized to push real objects with equal robustness. As discussed in Section V-D, future work will involve adding prior knowledge to the action sampler. We will also probe to understand why Push-Net works effectively for objects of concave geometries. Additionally, pushing objects through clutter remains as a challenge for future work.

Acknowledgements

This work is supported in part by the Singapore MoE Tier 2 grant MOE2016-T2-2-068 and the Office of Naval Research Global/US Air Force Research Laboratory grant N62909-18-1-2023.

References

[1] Pulkit Agrawal, Ashvin V 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*, pages 5074–5082, 2016.

[2] Arunkumar Byravan and Dieter Fox. Se3-nets: Learning rigid body motion using deep neural networks. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 173–180. IEEE, 2017.

[3] Berk Calli, Aaron Walsman, Arjun Singh, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set. *IEEE Robotics & Automation Magazine*, 22(3):36–52, 2015.

[4] Mehmet Dogar and Siddhartha Srinivasa. A framework for push-grasping in clutter. *Robotics: Science and systems VII*, 1, 2011.

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

[6] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in Neural Information Processing Systems*, pages 64–72, 2016.

[7] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[8] Eric Huang, Ankit Bhatia, Byron Boots, and Matthew Mason. Exact bounds on the contact driven motion of a sliding object, with applications to robotic pulling. *Robotics: Science and systems XIII*, 2017.

[9] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Jeongseok Lee, Michael X Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha S Srinivasa, Mike Stilman, and C Karen Liu. Dart: Dynamic animation and robotics toolkit.

[11] Adam Lerer, Sam Gross, and Rob Fergus. Learning physical intuition of block towers by example. *arXiv preprint arXiv:1603.01312*, 2016.

[12] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, page 0278364917710318, 2016.

[13] Wenbin Li, Seyedmajid Azimi, Aleš Leonardis, and Mario Fritz. To fall or not to fall: A visual approach to physical stability prediction. *arXiv preprint arXiv:1604.00066*, 2016.

[14] Kevin M Lynch and Matthew T Mason. Stable pushing: Mechanics, controllability, and planning. *The International Journal of Robotics Research*, 15(6):533–556, 1996.

[15] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017.

[16] Matthew T Mason. Mechanics and planning of manipulator pushing operations. *The International Journal of Robotics Research*, 5(3):53–71, 1986.

[17] Michael McCloskey. Intuitive physics. *Scientific american*, 248(4):122–131, 1983.

[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. 2017.

[19] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 3406–3413. IEEE, 2016.

[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*, pages 3–18. Springer, 2016.

[21] Zi Wang, Stefanie Jegelka, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Focused model-learning and planning for non-gaussian continuous state-action systems. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3754–3761. IEEE, 2017.

[22] Jiajun Wu, Ilker Yildirim, Joseph J Lim, Bill Freeman, and Josh Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In *Advances in neural information processing systems*, pages 127–135, 2015.

[23] Jiaji Zhou, Robert Paolini, J Andrew Bagnell, and Matthew T Mason. A convex polynomial force-motion model for planar sliding: Identification and application. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 372–377. IEEE, 2016.

[24] Shaojun Zhu, Andrew Kimmel, and Abdeslam Boularias. Information-theoretic model identification and policy search using physics engines with application to robotic manipulation. *arXiv preprint arXiv:1703.07822*, 2017.