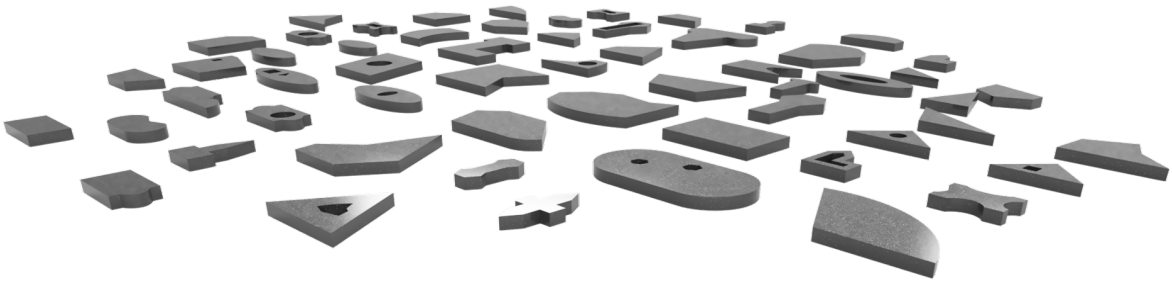
Planar Pushing of Unknown Objects Using a Large-Scale Simulation Dataset and Few-Shot Learning

Gao Ziyan, Armagan Elibol, and Nak Young Chong



SimPush: Large-Scale Planar Pushing Dataset available at https://github.com//SimPush

***Abstract*— Contact-rich object manipulation skills challenge the recent success of learning-based methods. It is even more difficult to predict the state of motion of novel objects due to the unknown physical properties and generalization issues of the learning-based model. In this work, we aim to predict the dynamics of novel objects in order to facilitate model-based control methods in planar pushing. We deal with this problem in two aspects. First, we present a large-scale planar pushing simulation dataset called SimPush. It is characterized by a large number of pushes and a variety of object physical properties, providing a wide avenue for exploring the object responses to the pusher action. Secondly, we propose a novel task-aware representation for pushes. This method keeps the spatial relation between the object and pusher and emphasizes the local contact features. Finally, we propose an encoder-decoder structured model possessing a cascaded residual attention mechanism to integrate prior knowledge to infer novel object motions. We experimentally show that the proposed model purely trained by SimPush attains good performance and robust prediction of novel object motions.**

I. INTRODUCTION

Non-prehensile pushing gives a simple yet efficient way to change the state of motion of an object. On the other hand, contact-rich robotic pushing is long-standing challenges with the nature of highly nonlinear dynamics. Reasoning the effect of the robot action becomes the core issue for action sampling for robot-object interaction. Many methods have been proposed to deal with the problem of predicting the object dynamics [1]. Recently, learning-based methods catch our eyes due to the capability of modeling complex object dynamics under weaker assumptions. However, the main issue is how well it can be generalized to the object that has not been encountered before.

There are two main challenges. First, a large-scale accurate object pushing dataset is needed. Currently, two different datasets in [2] and [3] have been published. However, they contain a limited number of objects all similar in shape and size. Secondly, humans usually infer how an object moves with a couple of interactions that they have not seen before. These capabilities are still far away for robots even with the current machine learning models. It remains an open issue for robots how to utilize the limited data to infer the object motion and represent the object state and action [3]. In this research, on the one hand, we present a large-scale, contact-rich pushing dataset called **SimPush** containing 59 objects with diverse dimensions and shapes that appear either convex or concave. Five physical properties considered are the surface friction, contact friction, center of mass (COM), mass, and moment of inertia of the object. The above properties are arranged in different ways to change the motion of the object pushed at various contact points in different directions. Eventually, more than 2 million pushes are collected. On the other hand, we propose a novel few-shot learning method to embed push priors and make use of an attention module to combine encoded knowledge from different aspects. Specifically, a novel task-aware representation of planar pushing is proposed that keeps the spatial relation between the object and the pusher by stressing the local contact features. It helps the learning model to encode the relation between object state changes and the applied action. A novel encoder-decoder model then embraces the flexibility in combining the encoded knowledge using a cascaded residual attention mechanism. We aim to train the learning model by SimPush and predict the motion of **unknown real objects** without any empirical dataset. The evaluation of the proposed model is conducted both by comparing with other models and by predicting the real novel object dynamics. The results show that the proposed method not only outperforms other models but also demonstrates the robust prediction of novel object motions.

II. RELATED WORK

There are two datasets made publicly available [2], [3]. Omnipush [3] contains objects created by combining four different magnet sides. However, they tend to have similar

All authors are with the School of Information Science, Japan Advanced Institute of Science and Technology, Nomi, Ishikawa 923-1292, Japan {s1920013, aelibol, nakyoung}@jaist.ac.jp

shapes sharing the same sides. [2] contains a limited number of objects. Notably, SimPush is much more diverse in object shape, size, and physical properties, and larger in sample size. Closing the simulation-to-reality gap, we experimentally show that the model trained by SimPush achieves comparable prediction accuracy on real object dynamics.

Mason [4] proposed an analytic model for quasi-static planar pushing. Goyal *et al*. [5] introduced the limit surface which reasons the frictional forces with object motion. Kloss *et al*. [6] proposed a hybrid model combining a data-driven model implemented by deep learning and the analytical model in [7]. The above-mentioned studies assumed uniform physical properties or known surface frictions.

Recently, Li *et al*. [8] proposed a data-driven method that utilizes the experience of push interactions with novel objects to implicitly learn a forward model encoding the relationship between the action and object state. However, the accuracy of the learned forward model was not investigated. In this work, we adapt their Push-Net model to explicitly learn to predict the state of motion of the object pushed. Our experiments reveal that Push-Net does learn to make use of pushing interactions but far from being accurate. On the other hand, in [9], the use of the Kalman filter was proposed to estimate object physical properties such as the COM, friction, mass, and others. However, the prediction accuracy was directly related to the quality of the estimation which seems to be intractable for the objects having complicated contact phenomena. Xu *et al*. [10] proposed a learning model to predict the physical properties based on pushing and collision trials. The idea was promising but only evaluated on a classification problem.

Some works tried to learn an inverse model to find the pushing action given the target. Agrawa *et al*. [11] attempted to enable the robot to understand the physical properties of objects and to predict their dynamics given external forces. Hermans *et al*. [12] proposed a regression model which predicts object motion for each contact location. [13] used a mixture density network [14] to predict diverse actions for pushing. These methods did not learn the object dynamics during pushing it. Some sampling-based methods have been proposed for pushing novel objects to the target pose [8]. In [9], a sampling and optimization method was proposed. Gao *et al*. [15] proposed a 2-stage framework that makes use of estimated physical properties to influence the sampling space to narrow the number of samplings.

Bauza *et al*. [3] proposed to use of Attentive Neural Process (ANP) [16] to learn the dynamics of the object under the external pushing forces. These models are developed upon ground object state and the influence of object shape to the motion dynamics is not considered.

III. SimPush

Simulation Environment The simulation environment was created with CoppeliaSim [17] and the Vortex physics engine was used to simulate the interaction between the pusher and objects. A sphere with a diameter of $0.95cm$ is attached to a cylinder with a length of $20cm$ to be the pusher. The pusher's position and motion is controlled to interact with objects. Each object is pushed across 5 floors with different coefficients of friction.

Objects We designed 59 objects that come in a variety of shapes, either convex or concave, as shown on the title page. The objects are diverse in size, ranging from $3.5cm \times 6cm$ to $20cm \times 17cm$. We align ten different COMs inside the object, and for each object with a unique COM, two different coefficients of contact friction (μ_c) are defined between the pusher and object. We then set the mass of the object and randomly sample two inertia tensors (I) by scaling the original inertia tensor. Finally, we obtain 200 different combinations of μ_s, COM, μ_c, mass, I. Table. I shows the details of objects and surfaces in **SimPush** dataset.

TABLE I
SUMMARY OF SIMPUSH

Surface friction coefficients (μ_s)	0.2, 0.4, 0.6, 0.8, 1.0
Contact friction coefficients (μ_c)	0.5, 1.0
Number of center of mass for each object	10
Range of object moment of inertia	$[0.01, 100]$
Range of object mass	[50, 400]
Number of pushes for each object	180
Number of shapes	59

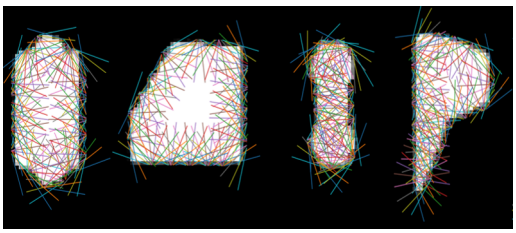


Fig. 1. Pushes for different shapes. For each shape, 17-18 contact points are randomly selected, each of which has 10 different push directions.

Data Collection Process

1) Initialize simulation environment
   - Set surface friction μ_s
   - Load object, determine initial pose, set COMs, μ_c, inertia, and mass of object.
   - Check if the object is stable. If not, reload the object, update COM, or scale inertia and mass.
2) Select contact points
   - Extract object contour
   - Sample 18 contour points with the same interval
   - Calculate 10 push directions in range of $[\frac{-2\pi}{5}, \frac{2\pi}{5}]$ *w.r.t.* the normal of each contact point.
3) For each contact point and push direction
   - Move pusher to the starting location
   - Execute a pushing of length $3cm$.
   - If object is flipped, or crushed into the surface, delete this push.

With combinations of properties, around 180 pushes are collected for each object as shown in Fig. 1. We capture RGB-D image to record the object state before and after pushing. The pusher is either in contact or not in contact

with the object at the start location. We repeat this procedure for each object until the simulation goes through all the combinations of physical properties and surfaces. We finally create more than 2 million pushes. For each push sample, the following information is recorded.

RGB-D Image: We capture the RGBD image data with size 224×224 using a simulated camera in orthographic projection mode, which is mounted on top of the table.

COM: We record the position of COM of the object before and after pushing represented by a 2-D position vector in the image frame.

Action: Actions are represented by the starting and terminating position of the pusher in the image frame.

Object POSE: We record the pose of the object before and after pushing.

Properties: Mass, inertia, contact friction, and surface friction are used in the implementation of the baseline model.

In Table II, SimPush was compared with the existing datasets.

TABLE II

COMPARISON WITH EXISTING PUSH DATASETS

Dataset	objects	surfaces	pushes	Platform	Size
SimPush	**2360**	5	180	Simulation	**∼2M**
Omnipush	250	1	250	Real	∼63K
Yu [2]	11	4	6000	Real	∼264K

IV. METHOD

Few-Shot Learning (FSL) is a type of machine learning problem that mainly focuses on fast adaptation to new tasks given a limited number of examples with supervised information [18]. The labeled examples are regarded as priors and the unlabeled ones are called test. In this work, the proposed model aims to predict the resulting pose of pushed objects by leveraging limited context.

The problem can be formulated as follows: given m context examples $\{O, A, \Delta O\}^m$ and n test examples $\{O, A\}^n$, where O is the observation of the object state represented by the object mask image, A is the pushing action, and ΔO is the change in object state represented by $\Delta x, \Delta y, \Delta\theta$. Both the O in the context and the test are the same. Now we give an introduction to the push embedding representation in context and test. Afterward, we detail the proposed learning model and attention modules used therein.

A. Push Embedding

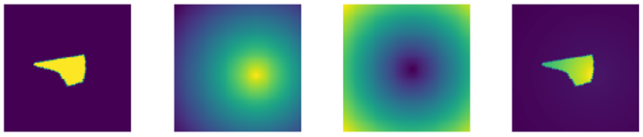


Fig. 2. Action maps: The first one is the mask image and the two middle images are the action maps. The last image is obtained by multiplying object mask to the action map at the pusher's start position.

1) Action maps: Before diving into task-aware representation, we introduce the way to describe pushing action that plays an essential role in the representation. Pushing action can be easily described by the starting and terminating positions of the pusher [6], [8], [9], commonly represented by a 4×1 vector in which two dimensions are for the starting position and the other two dimensions for the terminating position.

We generate two attention maps called action maps based on the starting and terminating positions of the pusher. We apply the Euclidean distance transformation to the both positions by Eq. 1. We then apply Eq. 2 if (x_p, y_p) refers to the starting position, while Eq. 3 is applied for the terminating position. $s(x,y)$ and $t(x,y)$ refer to pixel values of two action maps correspondingly. c in Eqs. 2 and 3 are the normalization terms, which depend on the size of the image used, here we set c to be the furthest distance to the image center. Fig. 2 shows an example of the generated action maps. The last image is obtained by multiplying the first two images. Intuitively, action maps provide an attention mechanism to make the model focus on the local geometric features such as contact points.

$$a(x,y) = \sqrt{(x-x_p)^2 + (y-y_p)^2} \tag{1}$$

$$s(x,y) = e^{-\frac{a(x,y)}{c}} \tag{2}$$

$$t(x,y) = \frac{a(x,y)}{c} \tag{3}$$

2) Task-aware representation: We stack the object mask and action maps along the channel axis to be the task-aware representation. The reason is twofold: (1) the spatial relation between the pusher and object state is kept, and (2) this representation flatters the learning model to easily focus on the local contact feature and pushing direction. Instead of feeding the object mask and 4-dimensional action vector to separate the network and combine them at the end, combining the object state and action maps at the beginning gives the learning model more flexibility to learn a meaningful representation. We will show that this method significantly improves the performance of the learning model.

3) Push Embedding Model: Based on the aforementioned representation, we propose the push embedding model shown in Fig. 3(left). CNN takes the stacked object mask and action maps as input and outputs f_d. ΔO is projected into high dimension space by Fully Connected Network(FCN). Finally, a residual attention module is used to combine them into the output f_e. In the proposed encoder-decoder model, there are two push embedding models: one is shown in Fig. 3(left) to encode the causality of pushes in context. The other one, without having the FCN and residual attention components, is used to embed pushes in the test.

For the CNN part of the push embedding module, we use 5 pre-trained layers of ResNet50 [19] to construct the base structure of the push embedding module. On top of pre-trained layers, we build a 1×1 2D convolution layer and one FCN. CNN outputs a 256-dimensional feature vector f_d. For obtaining f_e, FCN is constructed by two layers with the same dimension of 256. We then use residual attention modules to

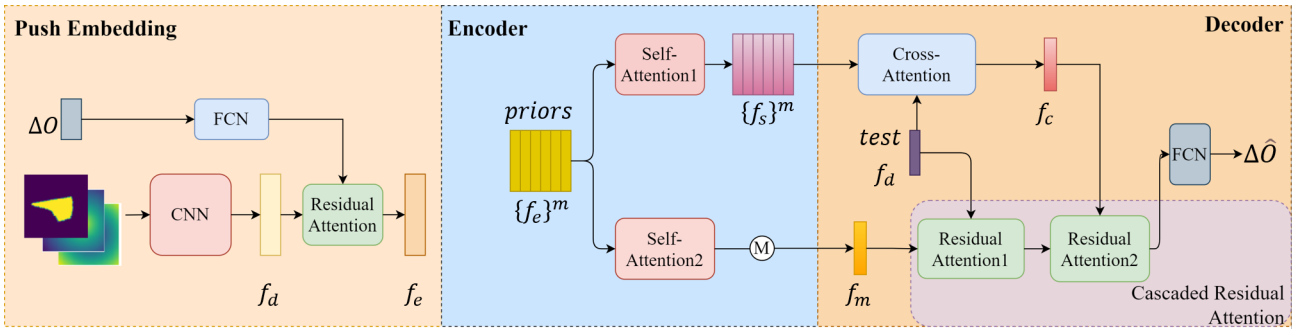


Fig. 3. Overview of the proposed encoder-decoder learning model.

combine it with f_d to obtain f_e. Residual attention modules will be explained in IV-C.2.

B. Proposed Model

Fig. 3 shows the proposed model. Similar to ANP, the encoder consists of two self-attention modules and models the interaction of pushing priors. The decoder consists of one cross-attention module, a cascaded residual attention module, and an FCN to output the predicted object motion. Instead of concatenating features from different aspects together and directly feed into multi-layer perceptron (MLP), cascaded residual attention modules selectively combine different source inputs to enhance feature representation. We experimentally show that this design achieves a lower loss.

The data flow is shown in Fig. 3. Assume that there are m number of context examples and only one test. During encoding, the pushing priors and test are fed into the proposed push embedding module to get $\{f_e\}^m$ and one f_d. Then $\{f_e\}^m$ is fed into two self-attention modules. $\{f_s\}^m$ is the output of the first self-attention module, while f_m is the output of the second self-attention module which is the mean of encoded priors as in [16]. Mean operation is adopted since it ensures the permutation invariance property, which is important to regularize the model. During the decoding, the cross-attention module measures the similarity between $\{f_s\}^m$ and f_d to output f_c. Then, f_c, f_d, f_m are fed into a cascaded residual attention module. Finally, FCN takes the output of the cascaded residual attention module to predict the object motion for test. In subsection IV-C, we detail the attention modules used in our model.

C. Attention Modules

1) DotProduct Attention: We use the dot-product attention model [20] as the self-attention and cross-attention module. Given a set of key-value pairs $\{(k_i, v_i)\}$ and a query q, attention refers to the weight of each key *w.r.t.* the query. The sum of each weighted value forms the value of the query. Eq. 4 gives the formula for computing the weight *w.r.t.* the query Q.

$$\textbf{DotProduct}(Q,K,V) := softmax(QK^T/\sqrt{d_k})V \in \mathbb{R}^{n\times d_v} \tag{4}$$

where $\sqrt{d_k}$ is the normalization term. Dot-product operation followed by softmax measures the similarity between Q and K. It is common to use multi-head attention mechanism, where Q, K and V are linearly transformed into sub-space and do the scaled dot-production given by

$$\textbf{MultiHead(Q,K,V)} = head_{1:h}W^O, where \tag{5}$$

$$head_i = \textbf{DotProduct}(QW_i^Q, KW_i^K, VW_i^V) \tag{6}$$

$head_{1:h}$ is formed by concatenating each head along the last dimension, W^O is the matrix that linearly transforms the $h_{1:h}$ to the corresponded value of Q.

For self-attention module in the proposed model, Q, K, V matrices are created from the the same source $\{f_e\}^m$. On the other hand, for cross-attention module, Q matrices are created using f_d computed from test data, and K, V matrices are created from $\{f_s\}^m$.

The self-attention and cross-attention modules are implemented as multi-head attention module, followed by layer normalization. We use 8 heads and both the output dimension of self-attention and cross-attention are 256.

2) residual attention module: The residual attention module merges two inputs into one feature vector. In this work, it only has one fully connected layer. The process can be represented by Eq. 7

$$F_{attn}(f_{in1}, f_{in2}) = \tanh(concat(f_{in1}, f_{in2})W^T + b) \tag{7}$$

$$f_{out} = F_{attn}(f_{in1}, f_{in2}) \cdot f_{in2} + f_{in1} \tag{8}$$

where W and b are the parameters of F_{attn}. tanh function normalizes the output of F_{attn} to range $(-1, 1)$. We use Eq. 8 to combine f_{in1} and f_{in2}. Here, f_{in2} will be selectively added to f_{in1}. In our cascaded residual attention module, the output can be represented by Eqs. 9 and 10.

$$f_{d,m} = F_{attn1}(f_d, f_m) \cdot f_m + f_d \tag{9}$$

$$f_{out} = F_{attn2}(f_c, f_{d,m}) \cdot f_{d,m} + f_c \tag{10}$$

The method selectively combines f_c, f_d, f_m, which enhances the representation and removes the redundant features. We evaluate this module by comparing it with an ablation model that combines f_c, f_d, f_m together directly.

V. EXPERIMENT

A. Training Dataset

As a step to training a few-shot learning model, we prepare the dataset that consists of $\{context, test, label\}$ tuples. For

each tuple, the context contains a series of pushing priors, which include object mask image, applied actions, and ΔO, while the test contains multiple actions whose future outcomes are expected to be predicted. The label contains the outcome of test actions. In each tuple, all of the actions are applied to the same object with the same pose. Our model utilizes 12 pushing priors to predict outcomes of the other actions.

In our SimPush dataset, there are about 180 pushes for each object. We randomly select 30 pushes of which 12 are used as pushing priors and the remaining ones are used as test data. For generating the training dataset, we choose an orientation randomly as the object state for a tuple. Then all the initial object states are transformed to the chosen orientation, and all actions and the object states after pushing are transformed with respect to the object state chosen. Finally, we crop the object mask image around the object center with (100,100) pixels. This procedure is repeated about 50 times for each object. By doing so, we obtain around $10k$ tuples for each object shape as there are 200 different combinations of internal parameters associated for each shape. We use 57 shapes for training and 2 shapes for testing. In total, the training set contains more than $570k$ tuples and the test set contains around $20k$ tuples.

B. Baseline and Ablation Models

1) Baseline models: We compare our model with the following baseline models.

NaiveCNN It takes all the features containing object mask image, action maps, position (x and y), the location of COM (x and y), surface friction (scalar) μ_s, contact friction (scalar) μ_c, the mass of object (scalar) and scale ratio for inertia (scalar) as input and outputs the ΔO. It consists of three sub-modules: The first one is a convolution network the same as the one in the aforementioned push embedding module. The second one is a fully connected layer that has $8,128,256$ to process low-dimensional vector. Finally, we use another FCN with $512,256,128,3$ to predict object motion.

Push-Net This model takes historical pusher-object interactions into consideration to encode the transformation of the object state. We adapt it to explicitly predict object motion. In this work, it takes 18 object masks and actions as input and outputs the action-outcome for the current object state. We also add the loss term of the center of mass to the loss function [8].

2) Model Ablations: We investigate contributions of object mask, action maps, and attention layers to the proposed model. We implemented the following three ablation models.

Model I (feature) This model is used to verify if the object mask helps predict object motion. We replace the push embedding modules with 3-layer FCN of sizes 7, 128, and 256, respectively. Actions are represented by 4 dimension vector. Actions and the changes in object state are combined directly to be the context, and test only contains actions. We keep other settings the same as the proposed model.

Model II (without action maps) This model is used to show the importance of action maps. Actions are represented by 4 dimension vector instead of action maps. In order to reuse pre-trained layers of ResNet50, we tile the object mask into 3 channels. Push embedding module takes the tiled object mask and action vector as input to output a 256 dimension vector. We keep other settings the same as the proposed model.

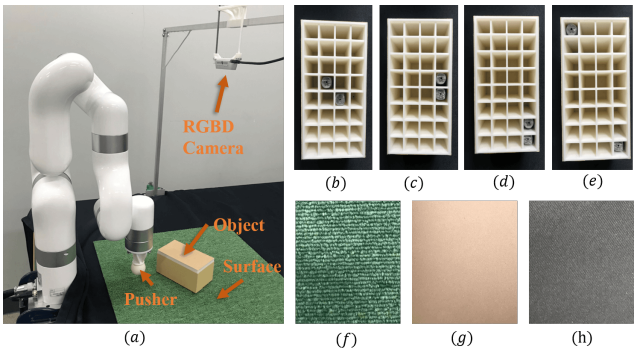


Fig. 4. Illustration of real experiment setting(a), different lead block positions ((b)-(e)) and surface frictions ((f) carpet, (g) foam, and (h) cloth).

Model III (without residual attention module) This model is used to show the contribution of residual attention modules of the decoder. We directly concatenate f_m, f_q, f_c together and feed them into MLP to predict object motion.

C. Training

We implemented all the models using Pytorch. Mean Squared Error is used as the loss function. We set the batch size to 128 for NaiveCNN and 32 for other models. The Adagrad optimizer [21] was used and the learning rate was set to 0.001 with exponential time decay. We stopped training at around the 20th epoch for all models since there were no significant changes in the loss curve thereafter. The training was conducted using an NVIDIA GTX 3090 GPU.

D. Real Experiment

Fig. 5. Real novel objects used in our experiment.

For evaluating the performance of the proposed model on the real platform, we conduct several experiments using the XArm 5 Lite robot. The experimental setup is shown in Fig. 4. The robotic arm holds the pusher to push the object and an RGBD camera (Intel RealSense D450) is used to obtain the object mask. We generate a point cloud from the camera and re-project it to the surface plane to get the object mask. Compared to the simulation, the object masks captured

TABLE III

PREDICTION ERROR ON THE TEST DATA OF ALL THE IMPLEMENTED MODELS.

Models	translation (mm)				rotation (degree)			
	mean	std	max	min	mean	std	max	min
NaiveCNN	3.55	2.16	10.62	0.69	3.14	2.63	12.35	0.20
Push-Net (without auxiliary)	4.76	3.02	14.03	0.76	4.31	3.59	16.50	0.29
Push-Net	4.75	3.01	14.36	0.83	4.57	3.72	16.90	0.26
Model I (feature)	3.52	2.33	11.33	0.65	3.49	2.96	13.96	0.22
Model II (without action maps)	3.14	2.17	10.71	0.56	2.93	2.67	12.98	0.17
Model III (without residual attention module)	2.99	2.05	10.12	0.53	2.78	2.52	12.28	0.16
Proposed Model	**2.64**	**1.96**	**9.73**	**0.42**	**2.53**	**2.42**	**11.86**	**0.13**

TABLE IV

TRANSLATION PREDICTION ERRORS FOR ALL THE COMBINATIONS OF OBJECT CENTER OF MASS AND SURFACES, MEASURED IN MILLIMETERS

	Carpet				Foam				Cloth			
	mean	std	max	min	mean	std	max	min	mean	std	max	min
com1	3.1	1.3	6.4	1.72	3.15	1.78	6.89	0.78	4.98	2.32	9.96	1.71
com2	5.29	2.31	8.78	1.55	4.52	1.93	7.53	1.5	4.86	3.25	10.52	1.08
com3	3.61	1.29	6.49	1.78	3.85	1.84	6.55	1.21	5.56	3.5	13.19	1.84
com4	3.92	1.5	6.29	1.79	3.84	1.88	7.42	1.57	6.24	2.63	11.29	2.6

TABLE V

ROTATION PREDICTION ERRORS FOR ALL THE COMBINATIONS OF OBJECT CENTER OF MASS AND SURFACES, MEASURED IN DEGREE

	Carpet				Foam				Cloth			
	mean	std	max	min	mean	std	max	min	mean	std	max	min
com1	4.17	3.38	12.96	1.03	3.36	2.05	7.2	0.4	3.38	2.09	6.63	0.53
com2	3.99	2.49	9.57	1.0	3.28	1.94	6.55	0.61	4.46	3.0	11.55	0.47
com3	3.02	2.55	9.13	0.2	2.48	1.64	5.68	0.19	3.92	2.53	10.35	0.47
com4	2.95	1.63	6.26	0.84	2.89	2.44	8.1	0.61	3.29	2.31	7.83	0.41

TABLE VI

PREDICTION ERRORS IN TRANSLATION AND ROTATION FOR REAL NOVEL OBJECTS.

	translation (mm)				rotation (degree)			
	mean	std	max	min	mean	std	max	min
obj1	4.23	1.95	8.88	1.30	4.03	4.08	13.53	0.20
obj2	3.34	1.72	6.30	0.57	2.92	2.23	7.71	0.05
obj3	4.67	2.36	9.28	0.26	3.72	3.07	11.84	0.59

in real experiments have more noise, and the shapes of the object are not clear. Moreover, calibration causes some errors making the prediction for object motion difficult.

For the first experiment, we used a 3D printer to print a COM controllable box of size $14 \times 7 \times 6cm$, in which we design 4×8 grids inside the box shown in Fig 4. We put two lead blocks into different positions to change its center of mass. We use 3 different surfaces made of artificial carpet, foam, and cloth having different friction coefficients and textures. We choose 4 different patterns to put two lead blocks to the box and push the box across 3 different surfaces shown in Fig. 4. For each combination of COM pattern and surface, the pusher executes a linear motion with $3cm$ to push the object 30 times at different contact points and pushing directions. Finally, we collect 360 pushes for all the combinations of COM settings and surfaces. We select 12 pushes as the context and the remains as the test for each COM setting and surface combination, utilizing the model purely trained by the simulated data to predict the pose change of the pushed object.

For the second experiment, we collected pushes for three unknown objects shown in Fig. 5. These objects are more complex in shape and contact condition between objects and floor surfaces. We use the artificial carpet as the surface and push each object 30 times while keeping other parameters the same as the first experiment.

VI. RESULT AND DISCUSSION

The performance of the models implemented on the test set are given in Table III. The last two columns represent prediction errors on translation and rotation. We use the 5th and 95th quantiles to represent the mean, standard deviation, max, and min. The average translation and rotation of the objects in the dataset are $19.13mm$ and $10.65°$.

The performance of two Push-Nets [8] are provided in Table III. Here we only show the performance of the last time step. These two models perform worse than NaiveCNN. This is mainly due to the fact that Naive CNN can directly access the internal parameters of the object. On the other hand, the result shows that the Push-Net trained with auxiliary COM target performed similarly to the Push-Net without auxiliary learning in translation but worse in rotation. It can be conjectured that COM plays less dominant role in object motion. Compared with the proposed model, Push-Net performs less accurately both in translation and rotation.

In the ablation study, we compare the proposed model to a variety of reasonable alternatives.

- By comparing Model I (feature) with other ablations, we show that the models using object mask images

perform better. One possible reason is that we leveraged the advance of CNN in push embedding. Therefore, the models have more flexibility to encode the relation between pushing action and object shape.

- By comparing Model II (no action maps) with the proposed model, we show that the action maps have greatly helped improve the performance in predicting both translation and rotation.
- By comparing Model III (no residual attention module) with the proposed model, we show that instead of concatenating the input from a different source and feeding to MLP, the cascaded residual attention model selectively combines them and leads to better performance.

Table IV and Table V show the result of predicting the motion of the COM controllable box pushed across different surfaces. As expected, because of the sim-reality gap, noisy mask image, calibration error, the results are not on par with the one shown in Table III. However, the proposed model purely trained by the simulation dataset predicted pretty close. The average translation and rotation of all the combinations are 20.8*mm* and 11.28°, and the mean prediction errors on translation and rotation are 4.17*mm* and 3.43°, respectively. We found that our model performed worst on the cloth surface both in translation and rotation. We argue that one main reason is due to the strong deformation of the cloth surface during pushing.

Finally, we show the performance of our model on real novel objects shown in Table VI. The average translation and rotation of the objects are 23.13*mm*, 21.41*mm*, 19.72*mm*, and 11.71°, 11.28°, 12.93°, respectively. Similarly, when comparing with the result in the first experiment, there is no significant difference both in translation and rotation error.

VII. CONCLUSION AND FUTURE WORK

Inspired by the related work [3], [2], we collected a large-scale simulation dataset called **SimPush** containing a variety of objects diverse in shape and size. We simulated planar pushing under hundreds of varying conditions of contact friction, surface friction, mass, inertia, and COM. Furthermore, we proposed a novel method to encode pushes, which greatly improved the model performance when comparing with baseline models. We evaluated the proposed model purely trained by SimPush on the real platform. We designed a COM controllable box and pushed it across different surfaces. Due to the noisy input and the simulation-to-reality gap, our model was not on a par with the result in simulation. However, our model still predicted object motions with reasonable accuracy. We pushed three unknown real objects with complicated contact conditions with the surface to challenge our model. Notably, the proposed model performed encouragingly well. Using the large-scale dataset, our proposed model efficiently learned to make use of context data to infer the novel action outcome.

There is still room for improvement in the proposed model. Directions for future research include: (1) multiple object or a single object pushing in an environment occupied by obstacles, (2) multiple contact pushing, (3) non-planar object pushing, and (4) multi-step ahead prediction.

REFERENCES

[1] J. Stüber, C. Zito, and R. Stolkin, "Let's push things forward: A survey on robot pushing," *Frontiers in Robotics and AI*, vol. 7, p. 8, 2020.

[2] K.-T. Yu, M. Bauza, N. Fazeli, and A. Rodriguez, "More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 30–37, 2016.

[3] M. Bauza, F. Alet, Y.-C. Lin, T. Lozano-Pérez, L. P. Kaelbling, P. Isola, and A. Rodriguez, "Omnipush: accurate, diverse, real-world dataset of pushing dynamics with rgb-d video," *arXiv preprint arXiv:1910.00618*, 2019.

[4] M. T. Mason, "Mechanics and Planning of Manipulator Pushing Operations," *International Journal of Robotics Research*, vol. 5, no. 3, pp. 53–71, 1986.

[5] S. Goyal, A. Ruina, and J. Papadopoulos, "Planar sliding with dry friction part 1. limit surface and moment function," *Wear*, vol. 143, no. 2, pp. 307–330, 1991.

[6] A. Kloss, S. Schaal, and J. Bohg, "Combining learned and analytical models for predicting action effects," *CoRR*, vol. abs/1710.04102, 2017.

[7] K. M. Lynch, H. Maekawa, and K. Tanie, "Manipulation and active sensing by pushing using tactile feedback.," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 416–421, 1992.

[8] J. K. Li, W. S. Lee, and D. Hsu, "Push-net: Deep planar pushing for objects with unknown physical properties," in *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018* (H. Kress-Gazit, S. S. Srinivasa, T. Howard, and N. Atanasov, eds.), 2018.

[9] A. Kloss, M. Bauza, J. Wu, J. B. Tenenbaum, A. Rodriguez, and J. Bohg, "Accurate vision-based manipulation through contact reasoning," in *IEEE International Conference on Robotics and Automation*, pp. 6738–6744, 2020.

[10] Z. Xu, J. Wu, A. Zeng, J. B. Tenenbaum, and S. Song, "Densephysnet: Learning dense physical object representations via multi-step dynamic interactions," *CoRR*, vol. abs/1906.03853, 2019.

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

[12] T. Hermans, F. Li, J. M. Rehg, and A. F. Bobick, "Learning contact locations for pushing and orienting unknown objects," in *IEEE-RAS International Conference on Humanoid Robots*, pp. 435–442, 2013.

[13] Z. Gao, A. Elibol, and N. Y. Chong, "Non-prehensile manipulation learning through self-supervision," in *IEEE International Conference on Robotic Computing*, pp. 93–99, 2020.

[14] C. M. Bishop, "Mixture density networks," 1994.

[15] Z. Gao, A. Elibol, and N. Y. Chong, "A 2-stage framework for learning to push unknown objects," in *Joint IEEE International Conference on Development and Learning and Epigenetic Robotics*, pp. 1–7, 2020.

[16] H. Kim, A. Mnih, J. Schwarz, M. Garnelo, A. Eslami, D. Rosenbaum, O. Vinyals, and Y. W. Teh, "Attentive neural processes," *arXiv preprint arXiv:1901.05761*, 2019.

[17] E. Rohmer, S. P. N. Singh, and M. Freese, "Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[18] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," *ACM Computing Surveys*, vol. 53, no. 3, pp. 1–34, 2020.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[21] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.