Learning Pushing Dynamics for Arbitrary 2D Rigid Bodies

April 28, 2025

Art Boyarov
Robotics Department
University of Michigan
Ann Arbor, MI
aboyarov@umich.edu

Zichen Zhang
EECS Department
University of Michigan
Ann Arbor, MI
zhangzzc@umich.edu

I. Introduction

Robotic manipulation of objects with arbitrary and complex shapes is a fundamental challenge in robotics, with broad applications in automation, manufacturing, logistics, and service robotics [1]. Many real-world tasks such as warehouse sorting, assembly line automation, and household assistance require robots to interact with objects whose shapes, sizes, and physical properties are not known a priori. In these settings, the ability to predict how an object will move in response to a given action is crucial for planning and control.

Traditional approaches to modeling object dynamics often rely on analytical models or require extensive prior knowledge of the object's geometry and physical parameters. However, these methods struggle to generalize to novel objects or to handle the diversity of shapes encountered in unstructured environments. Recent advances in data-driven methods, particularly those leveraging deep learning, have shown promise in learning complex dynamics directly from data, enabling robots to manipulate a wider variety of objects with minimal prior assumptions [2].

A particularly challenging instance of this problem is the task of planar pushing, where a robot must predict the motion of a 2D rigid body as it is pushed at different points and angles along its perimeter. The dynamics of planar pushing are highly sensitive to the geometry of the object, the contact location, and the direction of the applied force. Accurately modeling these dynamics for arbitrary shapes is essential for robust and flexible manipulation.

In this work, we address the problem of learning the dynamics of arbitrary 2D rigid bodies under pushing actions. We first introduce a parametrization of the action space that captures the push location, direction, and distance, enabling a standardized way to represent pushing actions at a wide range of points on the object's perimeter. We then develop two neural network architectures for learning the dynamics: a baseline shallow multi-layer perceptron (MLP) and a deeper, more advanced network that takes in the shape's point cloud and incorporates techniques like batch normalization and dropout for improved stability.

Both models are trained using simulation data collected with a Franka Panda robotic arm in PyBullet, pushing three complex shapes: an L shape, a U shape, and a Block M shape. We demonstrate that the deeper network achieves lower and more stable validation losses compared to the baseline, indicating its improved capacity to model complex dynamics.

Finally, we integrate the learned models into a Model Predictive Path Integral (MPPI) controller for closed-loop control and evaluate the system on the object pushing tasks for all shapes.

Our experiments show that the learned dynamics models enable the successful execution of pushing tasks across all shapes, validating the generalizability and practical effectiveness of our approach.

II. Implementation

A. Parametrization of action space

In our pushing of rigid bodies of arbitrary geometries, we parametrize the pushing action space as $u = [p \quad \phi \quad l]$. $p \in \mathbb{Z}$ is the point on the shape's perimeter where we push the shape, ϕ is the angle at which we push the shape relative to the edge at which we push, and l is the length that we push the shape by. To execute a pushing action specified by $u^* = [p^* \quad \phi^* \quad l^*]$ we first find the location of the point in the part's body frame using a look-up table with points on the shape's perimeter defined ahead of time. Let the point p^*'s location on the part surface in the local frame be $[p_x^* \quad p_y^*]$. The following formula is used to calculate the 2D start and end pushing locations x_{start} and x_end:

$$x_{start} = R_B^W \begin{bmatrix} p_x^* & p_y^* \end{bmatrix} + \begin{bmatrix} B_x & B_y \end{bmatrix} \tag{1}$$

$$\begin{aligned} \psi &= \text{atan2}(n_y, n_x) \\ \phi_B^* &= \phi^* + \psi \\ x_{end} &= R_B^W \begin{bmatrix} p_x^* - l^* \cos(\phi_B^*) & p_y^* - l^* \sin(\phi_B^*) \end{bmatrix} \\ &\quad + \begin{bmatrix} B_x & B_y \end{bmatrix} \end{aligned} \tag{2}$$

Where $[B_x \quad B_y]$ is the object's 2D position in the world frame, R_B^W is the rotation matrix from the object's local frame to the world frame, and $[n_x \quad n_y]$ is the normal to the object's surface at the pushing point. After calculating x_{start} and x_{end}, the robot arm's end effector is lowered to a height of 0.02m and moved from x_{start} to x_{end} while pushing the object.

B. Baseline Pushing Network

To learn a dynamics function $x_{t+1} = f(x_t, u_t)$ that describes the motion of a rigid body under pushing action u, we implemented two neural networks that took in different data about the object and the pushing action.

The first neural network took in the block's current state x_t and applied action $u = [p \quad \phi \quad l]$ encoded as $I = [x_t \quad l \quad \phi \quad b_{n_p \times 1} \quad p_x^W \quad p_y^W]$. $b_{n_p \times 1}$ is a one-hot encoding of the points: $b_i = 0$ for $i \neq p$, and $b_p = 1$. p_x^W and p_y^W are the x and y positions of the point p in the world frame.

This neural network has three layers with 200 neurons in both hidden layers. The first activation function is a ReLU due to the binary nature of the input data, and the second activation function is a tanh function to capture nonlinear relationships [3].

The baseline model is a relatively shallow multi-layer perceptron (MLP) with two hidden layers. It is designed to capture the basic relationships between the input features and the resulting object motion, and may be sufficient for simple or well-structured shapes. However, its limited depth and capacity may restrict its ability to model the more intricate dynamics in the training data we collect.

C. Advanced Point-Cloud Based Network

While both our baseline and deep network approaches use the same input representation—comprising the object's current state, action parameters, a one-hot encoding of the push point, and the coordinates of the selected push point, the key difference lies in the complexity and design of the neural network architecture itself [4].

In contrast to the simple MLP in the baseline, our improved approach employs a significantly deeper neural network architecture, drawing inspiration from networks developed for point cloud processing, such as PointNet [5], which is developed for 3D object classification and segmentation in computer vision. Although our input is not a true point cloud, we hypothesize that the more complex, hierarchical structure of these networks can better capture subtle interactions between the object's geometry and the applied action.

Specifically, the point cloud-inspired network features three hidden layers with 256, 128, and 64 neurons, respectively, and incorporates batch normalization and dropout (with a rate of 0.1) after each layer to improve generalization and training stability. The point cloud-inspired network has over twice as many trainable parameters and a more sophisticated structure, enabling it to capture more complex relationships in the data.

The motivation for this architectural choice is twofold. First, deeper networks have been shown to learn richer internal representations [6], which can be critical for tasks involving complex spatial reasoning [7]. Second, by adopting techniques from point cloud networks, we aim to future-proof our approach for scenarios where richer geometric input may become available. Even with the current input, our experiments test whether increased model capacity alone can yield improved predictive performance.

III. Results

A. Collecting Training Data

We developed a simulation using the PyBullet libraries where a Franka Panda robotic arm pushed a block according to randomly sampled actions described using the aforementioned parametrization [8]. For each shape, we collected 200 trajectories with 10 steps in them. Each trajectory was an ordered list of tuples $[x_t \quad u_t \quad x_{t+1} \quad p_t \quad p_{t+1}]$. x_t and x_{t+1} are the current and next states respectively, u_t is the pushing action, and p_t and p_{t+1} are the point clouds of the objects during states x_t and x_{t+1} respectively. The data collection took up to 20 minutes per shape.

B. Training the Baseline Neural Network

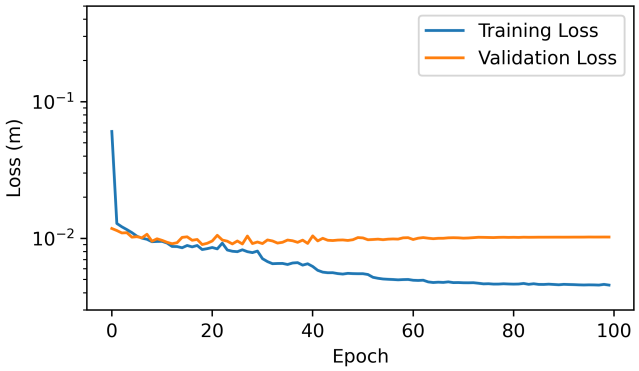

Fig. 1. Training and Validation losses for the U-shape baseline dynamics neural network.

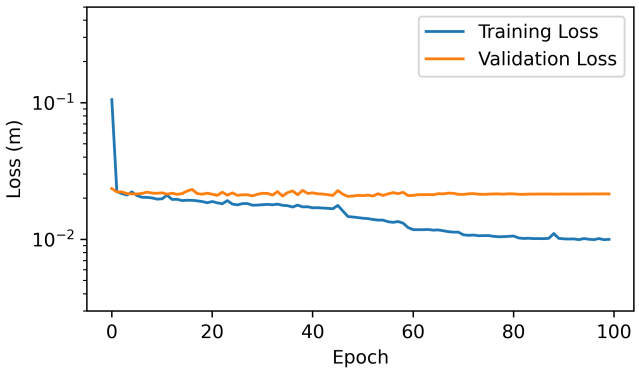

Fig. 2. Training and Validation losses for the L-shape baseline dynamics neural network.

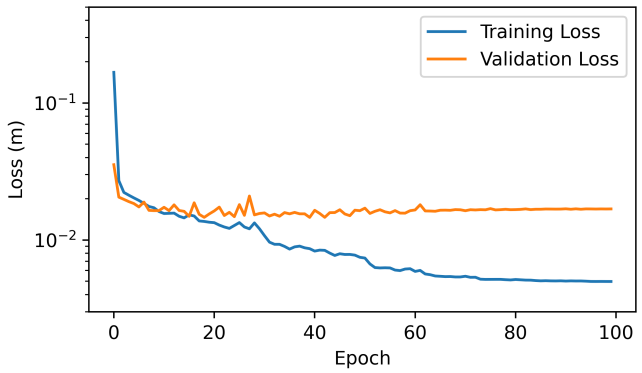

Fig. 3. Training and Validation losses for the Block M shape baseline dynamics neural network.

This neural network was trained using the Adam optimizer with dynamic lowering of the learning rate. Loss curves for each shape are given in Figures 1, 2, and 3.

C. Training the Deeper Point Cloud-Inspired Neural Network

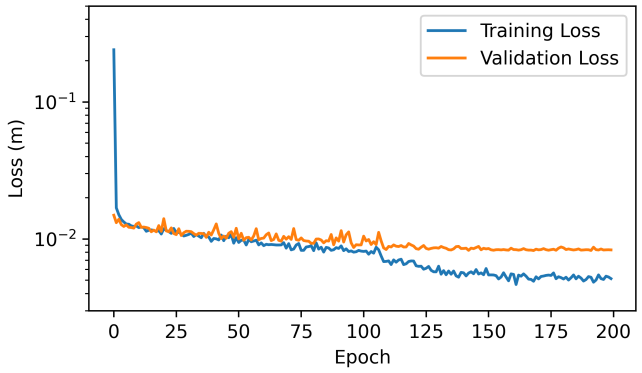

Fig. 4. Training and Validation losses for the U-shape point cloud dynamics neural network.

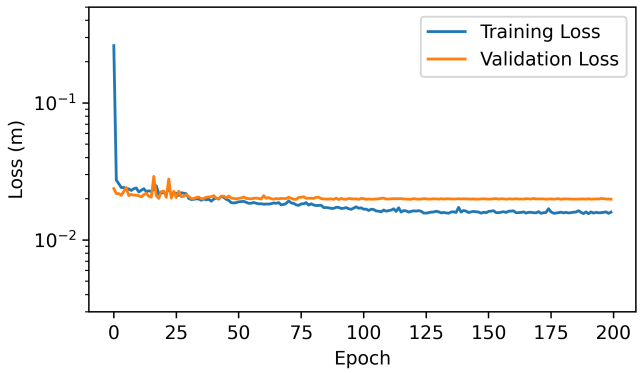

Fig. 5. Training and Validation losses for the L-shape point cloud dynamics neural network.

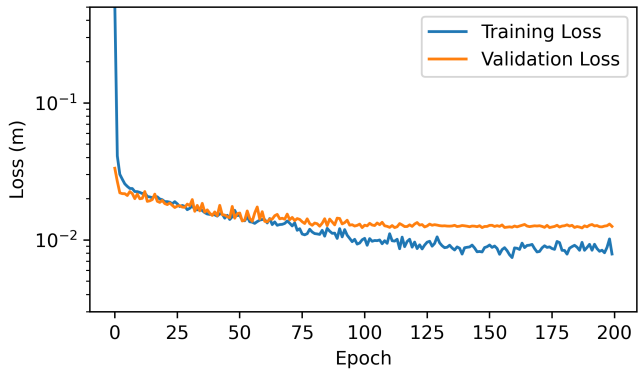

Fig. 6. Training and Validation losses for the Block M shape point cloud dynamics neural network.

For the point cloud-inspired network which has deeper architecture and more parameters, we employ the same training hyperparameters to ensure equal comparison with the baseline network training. However, the advanced network is trained for 200 epochs due to the larger number of parameters. The training and validation losses for the advanced network trained on each shape are given in Figures 4, Fig 5, and Fig 6.

As seen in the L-shape validation loss between Fig 2 and Fig 5, the advanced network shows a more stable decrease in validation loss. For the other two shapes, the validation loss of the advanced network also achieves lower or equal losses. This shows that our advanced network does not overfit to the training data we collect, and there are indeed more complex representations in the training data that are not captured by the shallow MLP architecture adopted in the baseline.

Both neural networks were trained using the following loss function to ensure predicted next states $\hat{x} = [\hat{x} \quad \hat{y} \quad \hat{\theta}]$ were the same as actual states $x = [x \quad y \quad \theta]$:

$$\mathcal{L}(x, \hat{x}) = \sum_{i=0}^{n} |\hat{R}p_i + \hat{t} - (Rp_i + t)|_2 \tag{3}$$

Where

$$R = \begin{bmatrix} \cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$$

$$t = [x \quad y]$$

and p_i is the i-th point of the shape in the shape's local frame. $\hat{R}$ and $\hat{t}$ are the predicted rotation matrix and translation.

D. MPPI Control for Trajectory Following

Both neural networks were integrated into an existing MPPI controller. MPPI is a model predictive control algorithm which is capable of planning control trajectories when given a learned dynamics model. MPPI randomly samples control action pertubations and adds them to a control trajectory, rolls out the different control trajectories, and averages the trajectories around the one giving the highest costs [9]. We had to modify our MPPI implementation to ensure the point parameter in our actions (specifying which point on the shape

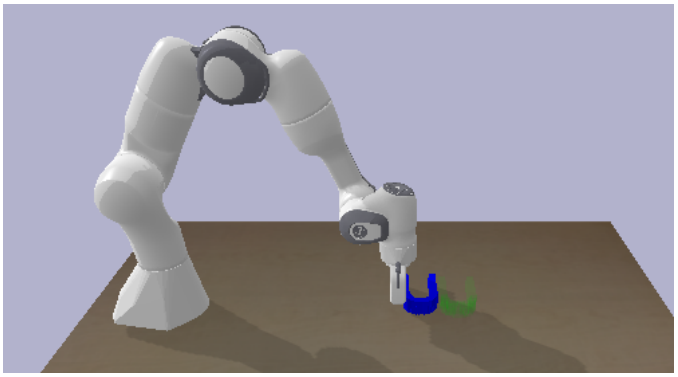

Fig. 7. The simulation of the pushing task with the Franka Panda pushing the U shape (blue) to the target position (green).

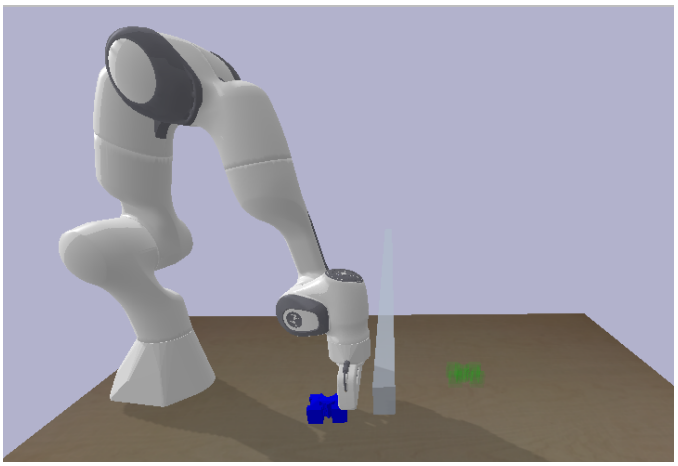

Fig. 8. The simulation of the obstacle avoidance task with the Franka Panda pushing the block M shape (blue) to the target position (green) while avoiding the grey obstacle.

to push at) was an integer, as MPPI's weighted averaging caused it to be a decimal. Additionally, we noticed that the points had to be numbered going clockwise around the shape's perimeter. This is because MPPI assumes that similar control actions lead to similar results. If two points close by lead to the same pushing outcome and cost, MPPI averages out the point indices, but if the indices are different then the final averaged index will be different from either of the points and may be an index for a point at which it is difficult to push the block. By numbering the points clockwise around the shape's perimeters, points close by that had a similar pushing outcome had similar indices, meaning when the indices are averaged in MPPI, and index which is close to those 'good' pushing indices is chosen.

The MPPI controller with our learned dynamics model was used to complete three tasks for each shape: pushing the shape forward; pushing the shape to the side and rotating it; and pushing the shape around one obstacle. A demonstration of the U-shape being pushed around one obstacle is shown in Fig. 7. Each task was run five times for each shape and the average number of steps needed to successfully execute the tasks is given in Tables I, II, III. A successful outcome is

TABLE I

STEPS TAKEN TO COMPLETE THE OBJECT PUSHING TASK USING MPPI WITH THE BASELINE AND POINT CLOUD LEARNED DYNAMICS MODELS

Shape	**Steps taken to complete pushing task**	
	Baseline	***Point Cloud***
U shape	3	3
L shape	8.20	**5.80**
Block M	9.20	**7**

TABLE II

STEPS TAKEN TO COMPLETE THE OBJECT ROTATING TASK USING MPPI WITH THE BASELINE AND POINT CLOUD LEARNED DYNAMICS MODELS

Shape	**Steps taken to complete rotating task**	
	Baseline	***Point Cloud***
U shape	18	**14.60**
L shape	17	**16.20**
Block M	**17.40**	18

defined as when the distance to the final object pose is within a tolerance of 0.05 m.

From the results, we can see that the point cloud method executes the tasks in fewer steps on average. This proves our prior hypothesis that a deeper and wider network with more parameters can better capture the complex latent relationship between object geometry and the action.

We observe that our methods will occasionally fail, primarily due to *overshooting* in certain actions, which causes the objects to slide out of the reach of the robotic arm. In addition, we suspect that the wide range of possible actions makes MPPI hard, since MPPI works well when actions are similar due to the averaging of costs step. However, in our case, the pushing actions are diverse, as we can push the shape in any direction from its current state. This increases the range of actions and makes it harder to find an optimal trajectory from the possible trajectories. As next steps, we plan to train the model with an expanded training dataset of more diverse shapes and actions for more epochs, incorporate richer geometric features into the input, and add uncertainty modeling to guide MPPI planning.

REFERENCES

[1] Ying Zheng et al. "A Survey of Embodied Learning for Object-Centric Robotic Manipulation". In: *arXiv preprint arXiv:2408.11537* (2024).

[2] Davis Rempe et al. *Learning Generalizable Physical Dynamics of 3D Rigid Objects*. 2019. arXiv: 1901.00466 [cs.CV]. URL: https://arxiv.org/abs/1901.00466.

TABLE III

STEPS TAKEN TO COMPLETE THE OBJECT PUSHING AMID OBSTACLES TASK USING MPPI WITH THE BASELINE AND POINT CLOUD LEARNED DYNAMICS MODELS

Shape	**Steps taken to complete obstacle avoidance task**	
	Baseline	***Point Cloud***
U shape	41	**35**
L shape	47.40	**32**
Block M	41.40	**37.60**

[3] Mwamba Kasongo Dahouda and Inwhee Joe. “A Deep-Learned Embedding Technique for Categorical Features Encoding”. In: *IEEE Access* 9 (2021), pp. 114381–114391. DOI: 10.1109/ACCESS.2021.3104357.

[4] Mona Alzahrani et al. “Deep models for multi-view 3D object recognition: a review”. In: *Artificial Intelligence Review* 57.12 (Oct. 2024). ISSN: 1573-7462. DOI: 10.1007/s10462-024-10941-w. URL: http://dx.doi.org/10.1007/s10462-024-10941-w.

[5] Charles R Qi et al. “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: *arXiv preprint arXiv:1612.00593* (2016).

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[7] Hyunjae Kim et al. “Exploring the spatial reasoning ability of neural models in human IQ tests”. In: *Neural Networks* 140 (Aug. 2021), pp. 27–38. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2021.02.018. URL: http://dx.doi.org/10.1016/j.neunet.2021.02.018.

[8] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org. 2016–2021.

[9] Grady Williams et al. “Aggressive driving with model predictive path integral control”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1433–1440. DOI: 10.1109/ICRA.2016.7487277.