Model-Based Control with Sparse Neural Dynamics
Supplemental Material

A Experiment Details

A.1 Perception

We use a single top-down RealSense D435i camera to capture color and depth images of the workspace (Figure 1). The color image is segmented using state-of-the-art detection and segmentation models, Grounding DINO [7] and Segment Anything Model (SAM) [5]. Given a prompt, Grounding DINO generates a bounding box for the corresponding objects in the image. To minimize detection errors, we implement specific thresholds for confidence and bounding box area for different prompts. The resulting bounding box coordinates are then utilized by SAM to produce an instance mask for the corresponding object. The mask, along with the depth image, camera intrinsics and extrinsics, are used to calculate the position of all segmented points in the global frame.

A.2 PWA Functions

We test our sparsification algorithm on recovering simple PWA functions, with two arguments and four affine pieces. The training data contains 1.6×10^3 transition pairs uniformly sampled within the input space with the ground truth function.

Our sparsification method starts with a Multi-Layer Perceptron (MLP) with $[96, 192, 192, 96]$ units in each layer and aggressively sparsifies into a compact model with only 2 ReLU units. We train using an Adam optimizer [4] with a learning rate of 1×10^{-4} until convergence, then finetune for 6×10^3 epochs in each sparsification round to obtain the final model.

A.3 Object Pushing

We provide a task-specific prompt “`letter t`” to the perception module along with confidence and bounding box thresholds to retrieve the bounding box around the T-shaped object (Figure 2a). After obtaining the segmentation mask of the object within the bounding box from SAM, we apply the mask to the depth image captured by the top-down camera to select only the subset of the pointcloud on the object. The position and orientation of the T-shaped object is determined by registering the captured pointcloud against a reference pointcloud sampled from a mesh model, using the Iterative Closest Points method [1].

The goal of the object pushing task is to position and orient a T-shaped object to align with a randomly sampled configuration within the workspace. Each action is a straight push from a start position to an end position. The state of the object at each time step is described with an ordered list of the coordinates of four selected keypoints. The dataset contains 5×10^4 transition pairs collected using the Pymunk simulator [2] (Figure 3a). To train our dynamics model, we concatenate the current keypoint state with the action, and pass through a three-layer MLP with 256 hidden units in each layer. We supervise the training with the mean squared error between the predicted next state $\hat{y}$ and the ground truth state y from executing the given action in the current state in the simulator, accompanied with an L_1 and L_2 weight regularization term and both ReLU and ID regularization terms as described in Section 3.4 of the main paper.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

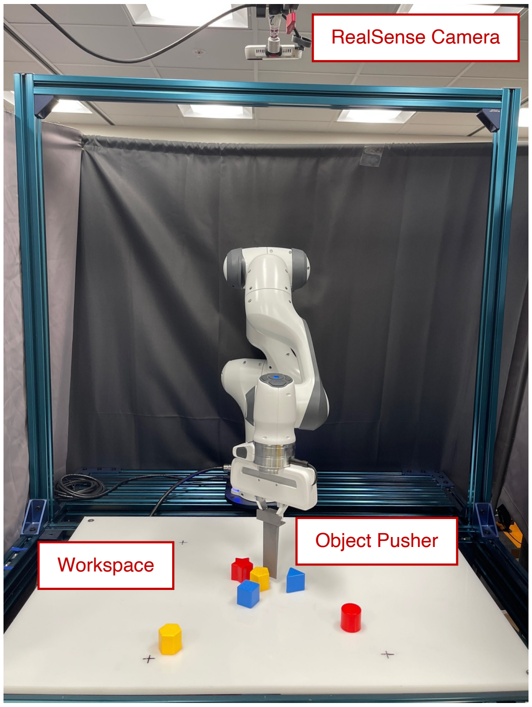
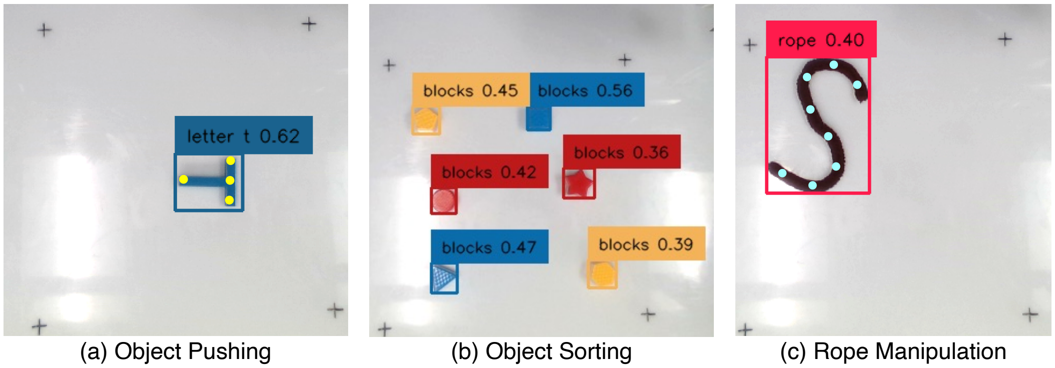


Figure 1: **Robot hardware experiments setup.** Robot experiments setup with a Franka Emika 7-DOF Panda robot arm and a top-down RealSense D435i RGB-D camera.



(a) Object Pushing (b) Object Sorting (c) Rope Manipulation

Figure 2: **Object and keypoint detection examples from our perception pipeline.** (a) Bounding box and detected keypoints positions. (b) Instance bounding boxes and detected object color. (c) Evenly spaced keypoints detected on the rope segment.

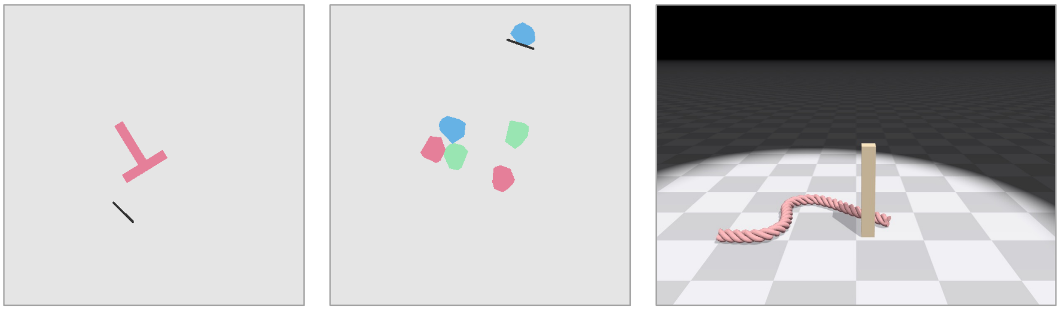
The full loss is defined as

$$\mathcal{L} = \text{MSE}(y, \hat{y}) + \lambda_{\text{ReLU}}\pi^1 + \lambda_{\text{ID}}\pi^2 + \lambda_{\text{reg}}(R_{L_1} + R_{L_2}), \tag{1}$$

where $\lambda_{\text{ReLU}} = 2 \times 10^{-3}$, $\lambda_{\text{ID}} = 1 \times 10^{-4}$, and $\lambda_{\text{reg}} = 3 \times 10^{-4}$ are constants balancing between the loss terms, selected for each task based on empirical performance on dynamics prediction. The sparsification process starts with the full model with 768 ReLUs and gradually reduces to 0 ReLUs (linear model), trained with an Adam optimizer [4] with a learning rate of 5×10^{-4}. We train the full model until convergence, then train each of the sparsified models for 400 epochs.

We formulate the closed-loop feedback control problem with the sparsified models following Section 3.5, and optimize for the objective to minimize the squared L_2 distance between the predicted state $\hat{y}$ by the dynamics model and the goal state y^*.

Our dynamics models are trained with one NVIDIA GeForce RTX 2080 Ti or one NVIDIA TITAN RTX GPU, followed by closed-loop control optimization using Gurobi [3] on an Intel i7-7700K 4.2 GHz CPU.



(a) Object Pushing (b) Object Sorting (c) Rope Manipulation

Figure 3: **Simulation environments used for data collection.** (a) The T-shaped object (pink) and the pusher (black) implemented in Pymunk for the object pushing task. (b) Objects of different colors interacting with the pusher (black) for the object sorting task. (c) The rope (pink) and the pusher (yellow) for the rope manipulation task, implemented with PyFleX [6].

A.4 Object Sorting

In the object sorting task, we initialize the environment with two or more randomly placed objects of two to three different colors. The goal is to collect objects of the same color near a distinctive goal region for each color. We experiment with six task variations, with two colors, one to four objects each color, or three colors, one or two objects each. The actions are specified by the start and end locations, similar to the object pushing task.

For perception, we prompt Grounding DINO with "`blocks`", then use the returned bounding boxes to obtain instance segmentation masks from SAM (Figure 2b). The position of each object is calculated as the centroid of the positions of all points captured by the segmentation mask. The color of each object is determined based on the average color of all pixels captured by the mask.

To demonstrate the applicability of our method on more complex neural dynamics models, and due to the permutation-invariant nature of the task, we choose graph neural networks as the function class. We adapt a model architecture similar to what was employed by Sanchez-Gonzalez et al. [8] with 64 hidden units per layer, and train with an Adam optimizer [4] using a learning rate of 1×10^{-3}. The loss function is the mean squared error between the predicted next state for all objects and the ground truth state, coupled with regularization terms as described in Equation 1, with $\lambda_{\text{ReLU}} = 3 \times 10^{-4}$, $\lambda_{\text{ID}} = 3 \times 10^{-5}$, and $\lambda_{\text{reg}} = 1 \times 10^{-4}$. The full model with 512 ReLUs is trained until convergence on a dataset with 1.5×10^5 transitions involving random interactions with only two objects collected in Pymunk [2] (Figure 3b), and each sparsified model is trained for 100 epochs.

The closed-loop control optimization is formulated using the same sparsified model for all task variations. We calculated the squared L_2 distance between each object and the goal of the corresponding color, and use the summation of all individual object-goal distances as the optimization objective.

A.5 Rope Manipulation

We use a prompt of "`rope`" for the rope manipulation task. Detecting the pose of keypoints on the deformable rope requires an additional step in the perception module. After generating a segmentation mask using SAM, we use Singular Value Decomposition to find the vector that approximately divides the rope mask into two segments of equal length, then recursively bisects each segment until we obtain eight segments of similar lengths. The keypoints are calculated as the mean of all points on the segmentation mask in each rope segment (Figure 2c).

The rope manipulation task requires deforming a rope segment to match a desired goal shape specified by eight keypoints evenly spaced on the rope. Leveraging the PyFleX simulator [6], we collected 6×10^4 transition pairs generated through random interaction as the dataset (Figure 3c). We adopt a similar formulation as in the object pushing task, using a three-layer MLP with 256 hidden units in each layer for the dynamics model, trained with the mean squared error loss between the predicted state and the ground truth state at the next time step. For the regularization terms, we follow

Equation 1 with $\lambda_{\text{ReLU}} = 2 \times 10^{-3}$, $\lambda_{\text{ID}} = 5 \times 10^{-5}$, and $\lambda_{\text{reg}} = 1 \times 10^{-4}$. We optimize the full model with an Adam optimizer [4] with a learning rate of 5×10^{-4} until convergence, then train each subsequent sparsified model for 500 epochs.

References

[1] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14:239–256, 1992.

[2] Victor Blomqvist. Pymunk. `https://pymunk.org`, November 2022. Version 6.4.0.

[3] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL `https://www.gurobi.com`.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

[5] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. *arXiv:2304.02643*, 2023.

[6] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. *arXiv preprint arXiv:1810.01566*, 2018.

[7] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, and Lei Zhang. Grounding dino: Marrying dino with grounded pre-training for open-set object detection, 2023.

[8] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin A. Riedmiller, Raia Hadsell, and Peter W. Battaglia. Graph networks as learnable physics engines for inference and control. *CoRR*, abs/1806.01242, 2018. URL `http://arxiv.org/abs/1806.01242`.