Combining Physical Simulators and Object-Based Networks for Prediction and Control

Anurag Ajay
CSAIL, MIT

Nima Fazeli
MechE, MIT

Maria Bauza
MechE, MIT

Jiajun Wu
CSAIL, MIT

Joshua B. Tenenbaum
CSAIL, MIT

Alberto Rodriguez
MechE, MIT

Leslie P. Kaelbling
CSAIL, MIT

Abstract

Simulators serve an important role in robot controller designs; however, practical real-world control problems involve difficult to model complex contact mechanics. Most simulators employ approximations that lead to a loss in precision. In this abstract, we introduce a hybrid dynamics model, simulator-augmented interaction networks (SAIN), combining a physics engine with an object-based neural network for dynamics modeling. SAIN captures the frictional interaction between objects more accurately and efficiently than either purely data-driven or analytical approaches. We demonstrate SAIN's capabilities on experiments in simulation and on a real robot. The results suggest that it also leads to better performance when used in complex control tasks with better generalization capabilities.

Introduction

Simulators are an essential tool for development of robot systems, in particular for systems well approximated by rigid-body dynamics. These systems benefit from relatively mature, fast, and general-purpose simulators that rely on approximate and efficient dynamics models. However, their practical application in robotics has been limited due to discrepancies between their predictions and real-world observations. Several recent studies [1–3] demonstrate that a major source of mismatches is the contact models used in these simulators. Contact is a complex physical interaction that is difficult to model accurately while maintaining computational efficiency. There are two methods of addressing the discrepancies: *data-driven* and *data-augmented* techniques.

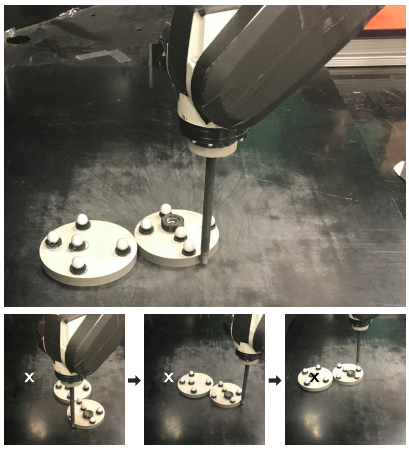


Figure 1: **Top**: the robot wants to push the second disk to a goal position by pushing on the first disk. **Bottom**: three snapshots within a successful push (target marked as X). The robot learns to first push the first disk to the right and then use it to push the second disk to the target position.

Recent data-driven techniques have been developed that account for states with a varied number of objects or generalize what they learn for one object to other similar ones [4, 5]. These methods are effective at generalizing over objects, modeling interactions, and handling variable numbers of objects. However, as they are purely data-driven, in practice they require a large number of training examples to arrive at a good model.

In this abstract, we present our most recent contribution simulator-augmented interaction networks (SAIN), incorporating interaction networks into a physical simulator for complex, real-world control problems.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

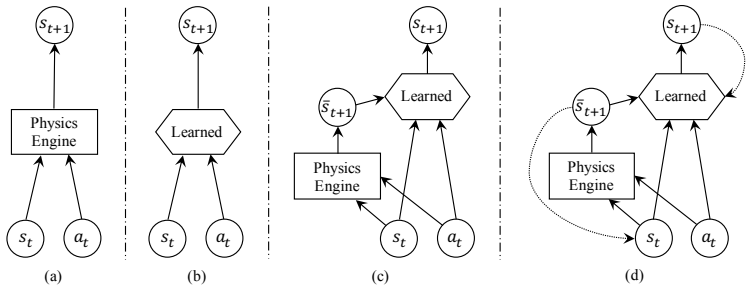


Figure 2: **Model classes:** (a) physics-based analytical models; (b) data-driven models; (c) simulator-augmented residual models; (d) recurrent simulator-augmented residual models.

Specifically, we show: i) Sample-efficient residual learning and improved prediction accuracy relative to the physics engine; ii) Accurate predictions for the dynamics and interaction of novel arrangements and numbers of objects, and iii) Utility of the learned residual model for control in highly under-actuated planar pushing tasks.

We demonstrate SAIN's performance on the experimental setup depicted in Fig. 1. Here, the robot's objective is to guide the second disk to a goal by pushing on the first. This task is challenging due to the presence of multiple complex frictional interactions and under-actuation. We demonstrate the step-by-step deployment of SAIN, from training in simulation to augmentation with real-world data, and finally control.

Background

Researchers have recently looked towards data-driven techniques to complement analytical models and/or directly learn dynamics. For example, Byravan and Fox [6] designed neural nets to predict rigid-body motions for planar pushing. Their approach does not exploit explicit physical knowledge. Kloss et al. [7] used neural net predictions as input to an analytical model; the output of the analytical model is used as the prediction. Here, the neural network learns to maximize the analytical model's performance. Fazeli et al. [8] also studied learning a residual model for predicting planar impacts.

The paper closest to ours is that from Ajay et al. [3], where they used the analytical model as an approximation to the push outcomes, and learned a residual neural model that makes corrections to its output. In contrast, our paper makes two key innovations: first, instead of using a feedforward network to model the dynamics of a single object, we employ an object-based network to learn residuals. Object-based networks build upon explicit object representations and learn how they interact; this enables capturing multi-object interactions. Second, we demonstrate that such a hybrid dynamics model can be used for control tasks both in simulation and on a real robot. Recent papers have explored model-predictive control with deep networks [9–11]. These approaches learn an abstract-state transition function, not an explicit model of the environment [12]. In contrast, we employ an object-based physical simulator that takes raw object states (e.g., velocity, position) as input.

Formulation

Let S be the state space and A be the action space. A dynamics model is a function $f : S \times A \to S$ that predicts the next state given the current action and state: $f(s, a) \approx s', \;\; \text{for all} \;\; s, s' \in S, \;\; a \in A$. There are two general types of dynamics models: analytical (Fig. 2a) and data-driven (Fig. 2b). Our goal is to learn a hybrid dynamics model that combines the two (Fig. 2c). Here, conditioned on the state-action pair, the data-driven model learns the discrepancy between analytical model predictions and real-world data (i.e. the residual). Specifically, let f_r represent the hybrid dynamics model, f_p represent the physics engine, and f_θ represent the residual component. We have $f_r(s, a) = f_\theta(f_p(s, a), s, a) \approx s'$. Intuitively, the residual model refines the physics engine's guess using the current state and action.

For long-term prediction, let $f^R_\theta : S \times S \times A \to S$ represent the recurrent hybrid dynamics model (Fig. 2d). If s_0 is the initial state, a_t the action at time t, $\bar{s}_t$ the prediction by the physics engine f_p at time t and $\hat{s}_t$ the prediction at time t, then

$$f^R_\theta(\bar{s}_{t+1}, \hat{s}_t, a_t) = \hat{s}_{t+1} \approx s_{t+1}, \quad f_p(\bar{s}_t, a_t) = \bar{s}_{t+1}, \quad \bar{s}_0 = \hat{s}_0 = s_0. \tag{1}$$

For training, we collect observational data $\{(s_t, a_t, s_{t+1})\}_{t=0}^{T-1}$. The recurrent formulation addresses the issues of compounding errors over a sequence of steps and the resulting mismatch between training and test phases from single-step predictions. To account for the possibility of multiple objects and with different physical properties we use *interaction networks* [4] as the data-driven model. An interaction network consists of 2 neural nets: f_{dyn} and f_{rel}. The f_{rel} network calculates pairwise forces between objects and the f_{dyn} network calculates the next state of an object, based on the states of the objects it is interacting with and the nature of the interactions. We adjust the formulation of the interaction network to account for multi-step predictions.

Let $s_t = \{o_t^1, o_t^2, \ldots, o_t^n\}$ be the state at time t, where o_t^i is the state for object i at time t. Similarly, let $\hat{s}_t = \{\hat{o}_t^1, \hat{o}_t^2, \ldots, \hat{o}_t^n\}$ be the predicted state at time t where o_t^i is the predicted state for object i at time t. In our work, $o_t^i = [p_t^i, v_t^i, m^i, r^i]$ where p_t^i is the pose of object i at time step t, v_t^i the velocity of object i at time step t, m^i the mass of object i and r^i the radius of object i. Similarly, $\hat{o}_t^i = [\hat{p}_t^i, \hat{v}_t^i, m^i, r^i]$ where $\hat{p}_t^i$ is the predicted pose of object i at time step t and $\hat{v}_t^i$ the predicted velocity of object i at time step t. Note that we do not predict any changes to static object properties such as mass and radius. Also, we note that while s_t is a set of objects, the state of any individual object, o_t^i, is a vector. Now, let a_t^i be the action applied to object i at time t. The equations for the interaction network are:

$$e_t^i = \sum_{j \neq i} f_{\text{rel}}(v_t^i, p_t^i - p_t^j, v_t^i - v_t^j, m^i, m^j, r^i, r^j), \quad \hat{v}_{t+1}^i = v_t^i + dt \cdot f_{\text{dyn}}(v_t^i, a_t^i, m^i, r^i, e_t^i),$$
$$\hat{p}_{t+1}^i = p_t^i + dt \cdot \hat{v}_{t+1}^i, \quad \hat{o}_{t+1}^i = [\hat{p}_{t+1}^i, \hat{v}_{t+1}^i, m^i, r^i].$$

Simulator-Augmented Interaction Networks (SAIN) extends an interaction network, where f_{dyn} and f_{rel} now take in the prediction of a physics engine, f_p. We now learn the residual between the physics engine and the real world. Let $\bar{s}_t = \{\bar{o}_t^1, \bar{o}_t^2, \ldots, \bar{o}_t^n\}$ be the state at time t and $\bar{o}_t^i$ be the state for object i at time t predicted by the physics engine. The equations for SAIN are

$$\bar{s}_{t+1} = f_p(\bar{s}_t, a_t^1, a_t^2, \ldots, a_t^n), \quad e_t^i = \sum_{j \neq i} f_{\text{rel}}(v_t^i, \bar{v}_{t+1}^i - \bar{v}_t^i, p_t^i - p_t^j, v_t^i - v_t^j, m^i, m^j, r^i, r^j),$$
$$\hat{v}_{t+1}^i = v_t^i + dt \times f_{\text{dyn}}(v_t^i, \bar{p}_{t+1}^i - \bar{p}_t^i, a_t^i, m^i, r^i, e_t^i), \quad \hat{p}_{t+1}^i = p_t^i + dt \times \hat{v}_{t+1}^i,$$
$$\hat{o}_{t+1}^i = [\hat{p}_{t+1}^i, \hat{v}_{t+1}^i, m^i, r^i].$$

These equations describe a single-step prediction. For multi-step prediction, we use the same equations by providing the true state s_0 at $t = 0$ and predicted state $\hat{s}_t$ at $t > 0$ as input.

We use a sample-based model-predictive controller with SAIN for the task. Our action space has two free parameters: the point where the robot contacts the first disk and the direction of the push. In our experiments, a successful execution requires searching for a trajectory of about 50 actions. Due to the size of the search space, we use an approximate receding horizon control algorithm with our dynamics model. The search algorithm maintains a priority queue of action sequences based on the loss introduced below. For each expansion, let s_t be the current state and $\hat{s}_{t+T}(a_t, \ldots, a_{t+T-1})$ be the predicted state after T steps with actions $a_t, \ldots, a_{t+T-1}$. Let s_g be the goal state. We choose the control strategy a_t that minimizes the the cost function $||\hat{s}_{t+T}(s_t, a_t, \ldots, a_{t+T-1}) - s_g||_2$ and insert.

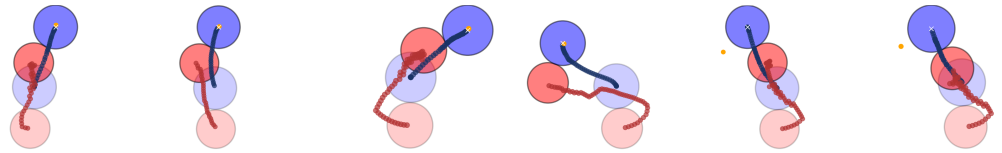
Results

We now test our models on a real robot. The setup used for the real experiments is based on the system from the MIT Push dataset [13]. The pusher is a cylinder of radius 4.8mm attached to the last joint of a ABB IRB 120 robot. The position of the pusher is recorded using the robot kinematics. The two disks being pushed are made of stainless steel, have radius of 52.5mm and 58mm, and weight 0.896kg and 1.1kg. During the experiments, the smallest disk is the one pushed directly by the pusher. The position of both disks is tracked using a Vicon system of four cameras so that the disks' positions are highly accurate.

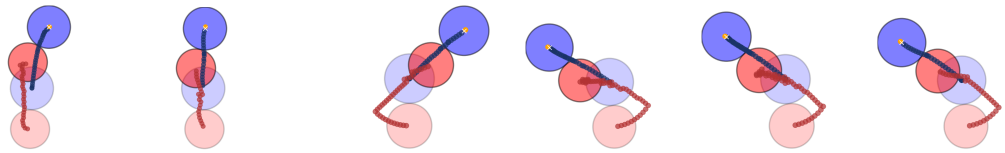
Results of forward simulation are shown in Table 1. SAIN outperforms IN on real data. While both models benefit from fine-tuning, SAIN achieves the best performance. This suggests residual learning also generalizes to real data well. All models achieve a lower error on real data than in simulation; this is because simulated data have a significant amount of noise to make the problem more challenging. We then evaluate SAIN (both with and without fine-tuning) for control, on 25 easy and 25 hard pushes. The results are shown in Fig. 3. As shown in the rightmost columns of Fig. 3a, the IN sometimes pushes the object too far and gets stuck in a local minimum whereas SAIN controls correctly.

Models	Fine-tuning	Object 1			Object 2		
		trans (%)	pos (mm)	rot (deg)	trans (%)	pos (mm)	rot (deg)
Physics	N/A	0.87	3.06	**0.32**	1.91	6.41	**0.17**
IN	No	0.86	2.96	0.96	1.84	5.75	0.32
SAIN (ours)	No	0.69	2.38	0.43	1.06	3.52	0.18
IN	Yes	0.63	2.23	0.41	0.61	2.05	0.19
SAIN (ours)	Yes	**0.42**	**1.50**	0.34	**0.43**	**1.52**	**0.17**

Table 1: **Errors on dynamics prediction in the real world.** SAIN obtains the best performance in both position and rotation estimation. Its performance gets further improved after fine-tuning on real data.



(a) Using the model trained on simulated data only



(b) Using the model trained on both simulated and real data

Figure 3: **Results on control tasks in the real world.** The goal: push red disk against the blue disk so that it reaches the target region (yellow). The left two columns are examples of easy pushes, while the right four show hard pushes. Top: The model trained on simulated data only performs well for easy pushes, but sometimes fails on harder control tasks; Bottom: The model trained on simulated and real data improves the performance, working well for both easy and hard pushes.

References

[1] N. Fazeli, S. Zapolsky, E. Drumwright, and A. Rodriguez, "Fundamental limitations in performance and interpretability of common planar rigid-body contact models," in *ISRR*, 2017.

[2] M. Bauza and A. Rodriguez, "Gp-sum. gaussian processes filtering of non-gaussian beliefs," *arXiv:1709.08120*, 2017.

[3] A. Ajay, J. Wu, N. Fazeli, M. Bauza, L. P. Kaelbling, J. B. Tenenbaum, and A. Rodriguez, "Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing," in *IROS*, 2018.

[4] P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu, "Interaction networks for learning about objects, relations and physics," in *NIPS*, 2016.

[5] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum, "A compositional object-based approach to learning physical dynamics," in *ICLR*, 2017.

[6] A. Byravan and D. Fox, "Se3-nets: Learning rigid body motion using deep neural networks," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 173–180.

[7] A. Kloss, S. Schaal, and J. Bohg, "Combining learned and analytical models for predicting action effects," *arXiv preprint arXiv:1710.04102*, 2017.

[8] N. Fazeli, S. Zapolsky, E. Drumwright, and A. Rodriguez, "Learning data-efficient rigid-body contact models: Case study of planar impact," in *CoRL*, 2017, pp. 388–397.

[9] I. Lenz, R. A. Knepper, and A. Saxena, "Deepmpc: Learning deep latent features for model predictive control," in *RSS*, 2015.

[10] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *ICML*, 2016.

[11] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn, "Universal planning networks," in *ICML*, 2018.

[12] D. Silver, H. van Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. Reichert, N. Rabinowitz, A. Barreto, and T. Degris, "The predictron: End-to-end learning and planning," in *ICML*, 2017.

[13] 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 *IROS*. IEEE, 2016, pp. 30–37.