Augmenting Physical Simulators with Stochastic Neural Networks: Case Study of Planar Pushing and Bouncing

Anurag Ajay[1], Jiajun Wu[1], Nima Fazeli[2], Maria Bauza[2],
Leslie P. Kaelbling[1], Joshua B. Tenenbaum[1], Alberto Rodriguez[2]

***Abstract*— An efficient, generalizable physical simulator with universal uncertainty estimates has wide applications in robot state estimation, planning, and control. In this paper, we build such a simulator for two scenarios, planar pushing and ball bouncing, by augmenting an analytical rigid-body simulator with a neural network that learns to model uncertainty as residuals. Combining symbolic, deterministic simulators with learnable, stochastic neural nets provides us with expressiveness, efficiency, and generalizability simultaneously. Our model outperforms both purely analytical and purely learned simulators consistently on real, standard benchmarks. Compared with methods that model uncertainty using Gaussian processes, our model runs much faster, generalizes better to new object shapes, and is able to characterize the complex distribution of object trajectories.**

I. Introduction

Simulators are an essential for the development of robot systems. An important class of systems in robotics is well approximated by rigid-body dynamics; relatively mature, fast, and general-purpose simulators have been developed for these systems. These simulators (*e.g.*, ODE [1], Bullet [2], MuJoco [3]) rely on approximate and efficient dynamics models and do not reason about uncertainty explicitly.

These simulators are an important tool, yet their practicality has been limited due to discrepancies between their predictions and real-world observations. A major source of mismatches is the contact models used in these simulators. Contact is a complex physical interaction with near impulsive forces over a small duration of time that involves local deformations and vibrations. Matters are complicated by the sensitivity of contact outcomes to initial conditions. These models are coarse approximations to contact, and recent studies ([4], [5], [6]) have shown the discrepancies between their predictions and real-world data. Further, Fazeli *et al.* [6] showed that there exist real-world contact outcomes which the models are unable to predict for any choice of their parameters. This suggests that generating uncertainty by defining distributions over contact parameters does not yield a sufficiently rich and descriptive distribution over outcomes.

In this study, we provide a framework for augmenting analytical motion models with empirical data that has higher accuracy while capturing uncertainty in predictions. We achieve this by using a novel type of recurrent neural networks, namely decoupled conditional variational recurrent neural nets, to learn the residual errors made by the analytical models. Once the neural networks are trained, they can correct model predictions and provide distributions over possible outcomes.

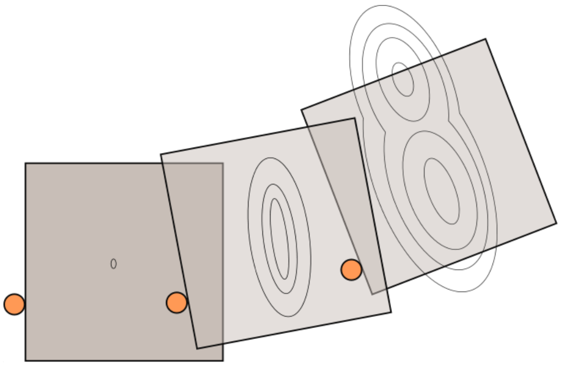


Fig. 1: The motion of an object being pushed appears stochastic and possibly multi-modal due to imperfections in contact surfaces, non-uniform coefficient of friction, stick/slip transitions, and micro surface interactions. In this study, we propose to augment analytical models to more accurately predict such outcomes while reasoning about uncertainty.

We demonstrate the efficacy of the data-augmented stochastic simulation framework in two cases: 1) a toy bouncing ball problem, and 2) planar pushing with a single point pusher using the empirical dataset from Yu *et al.* [5]. First, we use the toy problem to illustrate the implementation and details of the proposed model in simulation. We then move to the experimental planar pushing dataset to demonstrate the ability of the approaches to capture real-world data. We show that the data-augmented model outperforms its purely analytical and purely data-driven counterparts. Further, we demonstrate that this approach is data-efficient, as learning residuals is an easier and better formulated problem than learning full motion models. Experiments also suggest that the learned residual model generalizes better to different shapes than the pure learning-based dynamics model. The data-augmented residual model can reason about uncertainty, which plays an important role in planning and control.

II. Related Work

A. Models for Planar Pushing

Planar pushing is an important instance of planar manipulation, in which the robot moves objects on a horizontal surface through a set of pushes, in particular for objects that are too heavy or too large to be picked up by the robot. To model planar pushing, Goyal *et al.* [7] proposed the notion of "Limit Surfaces" (LS) as an invertible mapping between

[1] A. Ajay, J. Wu, L. P. Kaelbling, and J. B. Tenenbaum are with the Computer Science and Artificial Intelligence Laboratory at Massachusetts Institute of Technology, Cambridge, MA, USA

[2] N. Fazeli, M. Bauza, and A. Rodriguez are with the Department of Mechanical Engineering at Massachusetts Institute of Technology, Cambridge, MA, USA

a push and a consistent set of friction forces and object motions. Given the object's current pose and force applied to it, the LS predicts the subsequent motion assuming quasi-static motion. The LS assumes a pressure distribution over the contact patch and Coloumb friction law; it integrates over all possible instantaneous centers of rotation for the object to yield the mapping.

In general, the LS does not have a closed form solution; however, Howe and Cutkosky [8] showed that the LS can be approximated by an ellipsoid for uniform pressure distributions, constant friction coefficient, and quasi-static motions. Lynch *et al.* [9] used the ellipsoidal LS to develop a motion model to predict object motion given pusher motion. The ellipsoidal LS has been used for planar push control by Hogan and Rodriguez [10] and for shape reconstruction by Yu *et al.* [11]. In this study we also use the model proposed by Lynch *et al.* [9] as our analytical motion model.

B. *Learning Contact Dynamics*

Recently, researchers have looked towards data-driven techniques to complement existing analytical models and/or learn dynamics directly from data. The work by Kloss *et al.* [12] is the closest to ours, where the authors trained a neural network that provides input to an analytical model. In this framework, the output of the analytical model is used as the prediction; the neural network learns the best input parameters to maximize the performance of the analytical model. A benefit of this approach is that the model predictions are always feasible because of the analytical model, but the approach is deterministic, and relies on the expressiveness of the analytical model.

In the planar pushing case, these models may be sufficiently expressive to span the full range of outcomes, but this is not always the case in other contact interactions as shown by Fazeli *et al.* [6]. Further, the models in [12] only make single-step predictions—an approach that may not work well for long-horizon predictions due to compounding errors at each time step. In our paper, we use the analytical model as an approximation to the push outcomes, and learn a residual model that makes corrections to its output. We are thus not limited by the model's expressivity, as the neural network can make corrections outside the predictive range of the models. Further, we learn a stochastic recurrent network that makes long-horizon predictions in the form of a distribution over possible outcomes. We believe reasoning about the degree of confidence in outcome prediction can be used effectively in planning and control.

Fazeli *et al.* [13] also proposed to learn a residual model for prediction of empirical planar impacts. The residual learner in their paper is a Gaussian process and achieves significant improvement over the analytical contact models in terms of its prediction accuracy. Gaussian processes are however limited to Gaussian predictive distributions and are computationally slower, compared with neural networks. Further, the authors also did not study the effect of making long-term predictions, as their focus is on individual impact prediction accuracy. Zhou *et al.* [14] supplied a data-efficient approach to model the frictional interaction between an object and a support surface, by directly approximating the mapping between frictional wrench and slipping twist. Later, Zhou *et al.* [15] extended the model to simulate parametric variability in planar pushing and grasping.

Byravan and Fox [16] showed how to design a neural network to predict rigid-body motions in a planar pushing scenario. In this study, as a robot pushes an object, the neural network differentiates between the object and the table. The neural network makes predictions by explicitly predicting $SE(3)$ transformations and jointly learning the full motion model and the observation model. This approach is still deterministic and does not use any more physics knowledge.

C. *Uncertainty Modeling*

Reasoning about the uncertainty in actions and motions is a powerful tool in planning and control [17], [18], [19], [20]. In the context of planar manipulation, Bauza and Rodriguez [20] used Gaussian processes to learn the motion model of planar shapes and to propagate uncertainty using the GP-SUM algorithm. The GP-SUM algorithm is a hybrid Bayes and particle filter; it exploits the Gaussian structure of the motion model to efficiently approximate the distribution over outcomes as a mixture of Gaussians. Bauza and Rodriguez [20] showed that pushing can exhibit multi-modality and their approach is able to capture it. We use the model and algorithm from [20] as benchmarks for our approach and compare the two on the MIT push dataset [5].

A practical example of using the knowledge of uncertainty in planar manipulation was introduced by Zhou *et al.* [21]. They proposed a probabilistic algorithm that generates sequential actions to iteratively reduce uncertainty of objects in the plane, before grasping it with a parallel jaw gripper.

III. Formulation

In this section we provide the details of our proposed data-augmented stochastic simulation framework. The simulation framework has two components: an analytical model and a data-driven residual model. We first define each component; we then provide a detailed exposition of the data-driven residual model and its role as a method to improve simulation accuracy and to maintain a belief over states.

Let S represent the state space, A represent the action space, and (s, a, s') represent a state-action-state tuple, where $s, s' \in S$, $a \in A$, and s' is the state obtained after applying action a in state s. A dynamics model is a function $f : S \times A \rightarrow S$ that predicts the next state given the current action and state: $f(s, a) \approx s', \ \ s, s' \in S, \ \ a \in A$.

We distinguish between two classes of dynamic models: physics-based analytical models and data-driven models.

A. *Physics-Based Analytical Models*

These models are constructed from the laws of physics, domain knowledge, and convenient approximations often made for mathematical tractability. In this paper, we also refer to them as Physics Engines and use the terms interchangeably, though in practice physics engines may not be faithfully implementing the mathematical models. Generally, these

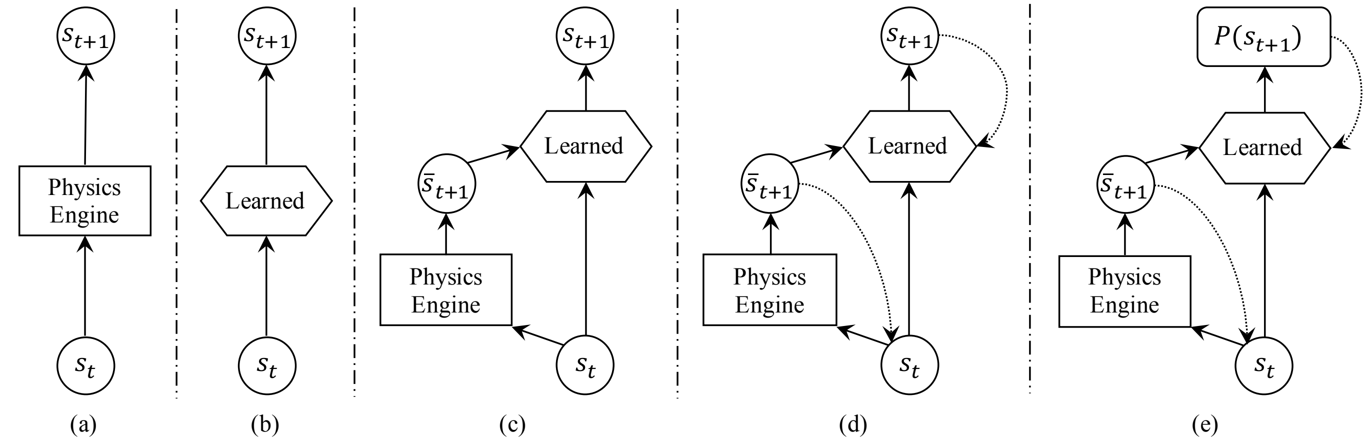


Fig. 2: Model classes: (a) physics-based analytical models; (b) data-driven models; (c) data-augmented residual models; (d) recurrent data-augmented residual models; and (e) stochastic recurrent data-augmented residual models.

models work well close to their assumptions and in structured environments, but their performance degrades as we move away from their nominal working conditions. Further, finding tractable models for complex tasks is difficult and requires extensive domain specific expertise. For the rest of this paper, let $f_p : S \times A \rightarrow S$ represent the analytical model.

B. Data-Driven Models

Rather than being hand engineered, these models are learned using data collected from the real world. They can be either parametric (*e.g.*, neural networks) or non-parametric (*e.g.*, Gaussian processes). For the purpose of discussion, let's assume a parametric model represented by $f_\theta : S \times A \rightarrow S$, where θ is the parameter vector. The model is learned using data collected from the real world; for example, the robot may take actions according to a fixed pushing policy and collect (s, a, s') tuples that represent the states of the object being pushed and the motion of the pusher. After collecting data $\{(s_t, a_t, s_{t+1})\}_{t=0}^{T-1}$, we solve the following optimization problem to obtain optimal parameters for the model:

$$\theta^* = \arg\min_\theta \sum_{t=0}^{T-1} \|f_\theta(s_t, a_t) - s_{t+1}\|_2^2 + \lambda\|\theta\|_2^2 \qquad (1)$$

where λ is a constant for regularization. After obtaining θ^*, we use f_{θ^*} as the representation of our motion model. While this approach requires no hand-engineering and directly learns from the data without making any assumptions, it does not make use of any domain knowledge, and consequently may require many examples to learn.

C. Data-Augmented Residual Models

We leverage advantages of both model classes and develop a new hybrid class of models, which we call *data-augmented residual models*, by combining a physics engine with a data-driven model. In this modeling framework, the data-driven part of the model takes the current state-action pairs and the predictions made by the physics engine as input, and effectively learns the discrepancy between analytical model predictions and real-world data (*i.e.* the residual). If f_r represents the data-augmented residual model, f_p represents its physics engine, and f_θ represents its 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.

D. Recurrent Data-Augmented Residual Models

Planning and control require long-horizon predictions of future states of the world, given actions taken by an agent using dynamic models. No matter how accurate the model is, it will have some error which will compound over a sequence of time steps. Moreover, the data-driven and data-augmented models are trained using data from real world trajectories. While simulating the future, these dynamics models will recursively use their own prediction as input for the next time step. As there will be error in their predictions at each time step, the input data given during simulation phase will have a different distribution than the input data during the training phase. This creates data distribution mismatch between training and test (or simulation) phases for both data-augmented residual models and purely data-driven models.

To address this problem, we propose to use a recurrent data-augmented residual model, trained to predict the entire trajectory based on an initial state and an action sequence. The *recurrent data-augmented residual model* consists of two components: a physics engine and a recurrent data-augmented residual model. The physics engine takes in the initial state and a sequence of actions at every time step; it generates an entire trajectory which serves as a good initial guess for the recurrent residual model. The residual model takes the initial state, a sequence of actions, and the trajectory predicted by the physics engine; it then predicts the next state. If f_r^R represents the data-augmented recurrent residual model, f_p represents its physics engine, and f_θ^R represents its residual component, we have

$$f_r^R(\bar{s}_t, \hat{s}_t, a_t) = f_\theta^R(f_p(\bar{s}_t, a_t), \hat{s}_t, a_t) = \hat{s}_{t+1} \approx s_{t+1}, \qquad (2)$$

$$f_p(\bar{s}_t, a_t) = \bar{s}_{t+1}, \quad \bar{s}_0 = \hat{s}_0 = s_0, \qquad (3)$$

where $(\hat{s}_t)_{t=0}^{T-1}$ is the predicted trajectory. The model is fully differentiable and can be trained by minimizing $\min_\theta \sum_{t=0}^{T-1} \|\hat{s}_t - s_t\|_2^2 + \lambda\|\theta\|_2^2$.

E. Stochastic Recurrent Data-Augmented Residual Models

No model is perfect, therefore the ability to provide a measure of uncertainty over possible future states is an important capability, allowing better informed planning and control. To this end, we formulate a stochastic model. Let f_r^R represents stochastic data-augmented recurrent residual model, we have

$$f_r^R(\bar{s}_t, \hat{s}_t, a_t) = f_\theta^R(f_p(\bar{s}_t, a_t), \hat{s}_t, a_t) = \hat{P}_{s_{t+1}}(\cdot), \quad (4)$$

$$f_p(\bar{s}_t, a_t) = \bar{s}_{t+1}, \quad \bar{s}_0 = \hat{s}_0 = s_0, \quad (5)$$

where $\{\hat{P}_{s_t}(\cdot)\}_{t=0}^{T-1}$ is the predicted trajectory distribution. The model is also differentiable and can be trained by minimizing $\min_\theta \sum_{t=0}^{T-1} \log \hat{P}_{s_t}(s_t|\theta) + \lambda\|\theta\|_2^2$.

IV. Methods

We now present how we realize the stochastic recurrent data-augmented residual model by providing an overview of the components and the way they are connected.

A. The Analytical Push Motion Model

As mentioned in Sec. II, we use the model proposed in [9] as our analytical pushing motion model (physics engine). The motion model takes the current configuration of the object and the pusher motion, and returns the predicted object motion at each time step. To make predictions, the motion model first computes a motion cone for the current pusher velocity and object configuration. Next, depending on whether the pusher velocity lies inside or outside of the cone, the model identifies the contact as sticking or sliding respectively. Finally the model computes the object motion using the sticking/sliding classification and the ellipsoidal LS.

In the experimental setup of the planar push dataset [5], the time history of object and robot pusher motion are recorded. We can compute the analytical model predictions using the current configuration of the object and robot pusher motion. In doing this, we observe discrepancies between the model prediction and the measured data. The discrepancy is due in part to the ellipsoidal approximation, and in part to variations in coefficients of friction across the surface, micro-interactions between the object and surface, and potentially anisotropic frictional properties. The latter effects are impractical to model analytically and difficult to predict ahead of time.

B. Stochastic Neural Networks

We implement our recurrent data-augmented residual model as a GRU [22], a widely used recurrent network for modeling long-term correlations. The model is however deterministic. The simplest way to incorporate stochasticity is to model $P_{s_t}(.)$ in Eqn. 4 as a Gaussian distribution, *i.e.*, $\hat{P}_{s_t}(.) = \mathcal{N}(\hat{\mu}_{s_t}, \hat{\sigma}_{s_t})$. However, this limits our model's ability to characterize complex distributions in real world.

Chung *et al.* [23] proposed to incorporate variational autoencoders [24] into recurrent nets and named their model variational RNNs (VRNNs). A VRNN supports modeling highly complex distributions over time. Their model however cannot be conditioned on additional inputs such as control variables (*e.g.* push forces). We instead embed a conditional variational autoencoder into our GRU. It therefore becomes a variant of VRNNs, namely *Conditional VRNNs*.

1) Variational Recurrent Neural Networks: VRNNs are recurrent generative models used for modeling multi-modal trajectories. It has three interconnected components: priors, an encoder, and a decoder. Suppose we represent a given trajectory as $\{x_t\}_{t=0}^T$. During training, the encoder takes the trajectory as input and infers latent random variables $\{z_t\}$ as

$$P(z_t|x_t) \sim \mathcal{N}(\mu_{z,t}, \sigma_{z,t}), \quad (6)$$

$$[\mu_{z,t}, \sigma_{z,t}] = \varphi^{\text{enc}}(\varphi^x(x_t), h_{t-1}), \quad (7)$$

where φ^{enc} represents the encoder, φ^x is a function that extracts features of x_t, and h is the hidden vector in the GRU. We then sample the latent random variable from the above distribution using a reparameterization trick [24], formulated as $z_t = \mu_{z,t} + \epsilon_t \times \sigma_{z,t}, \quad \epsilon_t \sim \mathcal{N}(0, I)$. After that, the decoder uses the sampled latent variable z_t to reconstruct the trajectory, following $P(x_t|z_t) \sim \mathcal{N}(\mu_{x,t}, \sigma_{x,t})$, where

$$[\mu_{x,t}, \sigma_{x,t}] = \varphi^{\text{dec}}(\varphi^z(z_t), h_{t-1}). \quad (8)$$

Here, φ^{dec} is the decoder and φ^z is a feature extractor for z_t.

In a VAE, we enforce the distribution of the latent vector $\{z_t\}$ to be close to a prior distribution [24]. In VRNN, the prior φ^{prior} is learned and follows the distribution

$$P(z_t) \sim \mathcal{N}(\mu_{0,t}, \sigma_{0,t}), \quad \text{where } [\mu_{0,t}, \sigma_{0,t}] = \varphi^{\text{prior}}(h_{t-1}). \quad (9)$$

Finally, the RNN f^{RNN} updates its state as

$$h_t = f^{\text{RNN}}(\varphi^x(x_t), \varphi^z(z_t), h_{t-1}). \quad (10)$$

VRNN is trained by minimizing

$$\sum_{t=1}^{T} D_{\text{KL}}\left(\mathcal{N}(\mu_{z,t}, \sigma_{z,t}||\mathcal{N}(\mu_{0,t}, \sigma_{0,t})\right) - \frac{1}{L}\sum_{t=1}^{T}\sum_{i=1}^{L} P(x_t|\mu_{z,t} + \epsilon_{i,t} \times \sigma_{z,t}) + \lambda\|\theta\|_2^2, \quad (11)$$

where $\epsilon_{i,t} \sim \mathcal{N}(0, I)$, λ is a regularization constant, and θ is a vector containing all the parameters in our model. Once the VRNN is trained, we use the prior to sample latent random variables and use them to generate trajectories.

2) Conditional VRNNs: We want a VRNN to be conditioned on a sequence $\{u_t\}_{t=0}^T$ (*e.g.*, the control inputs). To this end, the posterior distribution of z_t (Eqn. 6) is now $P(z_t|x_t, u_t) \sim \mathcal{N}(\mu_{z,t}, \sigma_{z,t})$, where

$$[\mu_{z,t}, \sigma_{z,t}] = \varphi^{\text{enc}}(\varphi^x(x_t), \varphi^u(u_t), h_{t-1}). \quad (12)$$

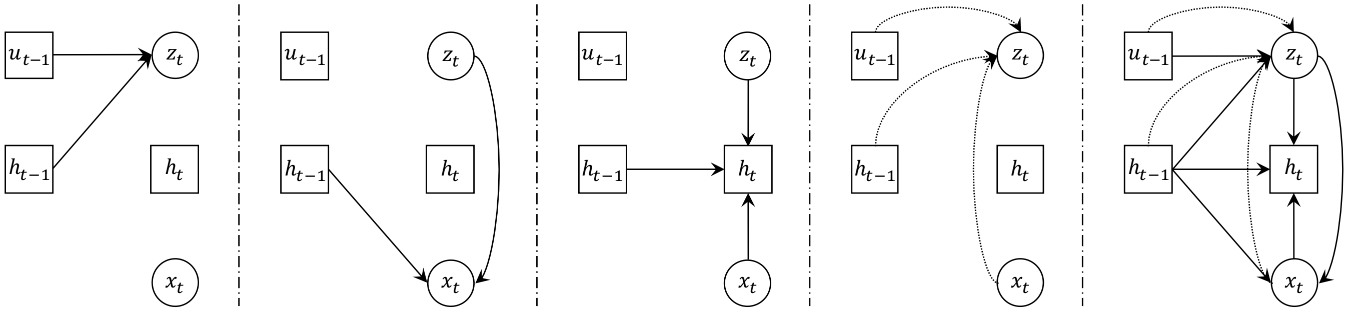
The prior distribution (Eqn. 9) also becomes conditional $P(z_t|u_t) \sim \mathcal{N}(\mu_{0,t}, \sigma_{0,t})$, where

$$[\mu_{0,t}, \sigma_{0,t}] = \varphi^{\text{prior}}(\varphi^u(u_t), h_{t-1}) \quad (13)$$

And the state update equation (Eqn. 10) becomes

$$h_t = f^{\text{RNN}}(\varphi^x(x_t), \varphi^z(z_t), \varphi^u(u_t), h_{t-1}). \quad (14)$$

Here, the decoder (Eqn. 8) does not depend on $\{u_t\}_{t=0}^T$ because the latent vectors $\{z_t\}_{t=0}^T$ already have the capacity to contain all information about the control sequence $\{u_t\}_{t=0}^T$.



(a) Prior (b) Generation (c) Recurrence (d) Inference (e) Overall

Fig. 3: Illustrations for conditional VRNNs: (a) the prior model of the latent representation z_t; (b) the generation model for x_t; (c) the recurrence model for h_t; (d) the inference model for z_t; and (e) the overall conditional VRNN design.

3) Decoupled Conditional VRNNs: In our experiments, we predict trajectories whose length varies from 100 to 1000. Because training and evaluating a VRNN becomes slower with these long trajectories, we propose an approximation to conditional VRNNs, which we call *Decoupled Conditional VRNNs*. A conditional VRNN is slow, because updates in a RNN have temporal dependence. However, we observe that the encoding, decoding, and prior networks are not interdependent: for example, the encoder only needs ϵ_t to sample z_t internally; it does not take signals from the decoding and the prior networks. Thus, in DCVRNNs, we disentangle the model into three recurrent neural nets, one each for priors, the encoder, and the decoder. Specifically, we first have the Gaussian noises sampled as $\epsilon_t \sim \mathcal{N}(0, I)$. Then the equation for the encoder becomes $P(z_t|x_t, u_t) \sim \mathcal{N}(\mu_{z,t}, \sigma_{z,t})$, where

$$h_t, [\mu_{z,t}, \sigma_{z,t}] = f_{\text{enc}}^{\text{RNN}}(\varphi^x(x_t), \varphi^u(u_t), \epsilon_t, h_{t-1}). \quad (15)$$

The decoder is now $P(x_t|z_t) \sim \mathcal{N}(\mu_{x,t}, \sigma_{x,t})$, where

$$h_t, [\mu_{x,t}, \sigma_{x,t}] = f_{\text{dec}}^{\text{RNN}}(\varphi^z(z_t), h_{t-1}). \quad (16)$$

The prior is now $P(z_t|u_t) \sim \mathcal{N}(\mu_{0,t}, \sigma_{0,t})$, where

$$h_t, [\mu_{0,t}, \sigma_{0,t}] = f_{\text{prior}}^{\text{RNN}}(\varphi^u(u_t), \epsilon_t, h_{t-1}). \quad (17)$$

The loss function remains unchanged.

We use a DCVRNN as our stochastic data-augmented residual model by having

$$x_t = s_t, \quad u_t = [s_0, a_t, \hat{s_{t+1}}], \quad \hat{s}_{t+1} = f_p(\hat{s}_t, a_t), \quad (18)$$

where s_t represents the state at time t, a_t represents the action at time t, f_p represents the physics engine, and $\hat{s}_t$ represents the state predicted by the physics engine.

V. Experiments

We study two scenarios: ball bouncing and planar pushing. We first generate synthetic data of ball bouncing and use them as an illustrative example to demonstrate the efficacy of our model. We then evaluate our model on the MIT Push dataset [5] and compare it with baselines and state-of-the-art motion models. We further present analyses on how well our model generalizes across shapes and materials.

A. Experiments with Bouncing Balls

Data. When a ball bounces against the ground, it may reach different heights due to the irregularities in the ground surface which leads to different coefficients of restitution. Here we simulate the process using PyBullet. Specifically, we choose a height randomly from [4m, 5m] and a coefficient of restitution for ball-ground interaction from $\mathcal{N}(0.5, 0.1)$. We then drop a ball of radius 0.5m from the sampled height, and record its height and vertical velocity for 400 time steps with a sampling frequency of 60Hz. For our physics engine, we create another PyBullet environment but fix the coefficient of restitution to 0.65. We want our physics engine to produce trajectories that are different from our training data, but serve as a good initial guess.

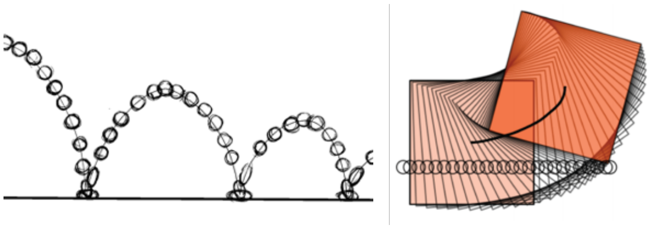


Fig. 4: The two scenarios: ball bouncing and planar pushing.

Metrics. We use three metrics for evaluation. The first two are the average error in object height reported as a percentage (trans) and in absolute values with meters as unit (pos). The third is the average error on the object's vertical velocity (vel) in metres per second.

Methods. We compare with three baselines. The first is just the average translation and rotation over the dataset. This is equal to the error of always predicting zero movement, and we therefore name it Zero. The second (Physics) is the full deterministic, analytical model described above. The third (Neural) is to use the stochastic neural network alone without the simulator. Our full model (Hybrid) combines the simulator's and the network's predictions. For the stochastic Hybrid and Neural models, we sample 10 trajectories for each input and take their mean as our prediction.

Setup. We implement our network in PyTorch. We train our network using the loss function in Eqn. 11. We use the ADAM optimizer [25] with a learning rate of 10^{-3}, a decay of 0.5 every 2,500 iteration for a total of 10,000 iteration, and a batch size of 100. Our training set contains 800 trajectories, while our test set has 100. For this experiment, φ^x, φ^u, φ^z are all identity functions. $f_{\text{enc}}^{\text{RNN}}$ and $f_{\text{prior}}^{\text{RNN}}$ are both GRUs with 2 hidden layers and a hidden size of 16, followed by a linear layer of hidden size 4 for mean, and another parallel

	Train				Test			
Models	loss ($\times 10^{-2}$)	trans (%)	pos (mm)	rot (deg)	loss ($\times 10^{-2}$)	trans (%)	pos (mm)	rot (deg)
Zero	N/A	N/A	N/A	N/A	N/A	99.99	359.44	49.46
Physics	N/A	N/A	N/A	N/A	N/A	1.93	6.91	7.71
Neural	0.41	0.72	2.42	1.85	0.68	0.84	2.81	2.48
Hybrid	0.36	0.54	1.86	1.73	0.47	0.60	2.04	2.03

TABLE I: Our hybrid model achieves the best performance in both position and rotation estimation for *rect1*, compared with methods that rely on physics engines or neural nets alone. Here we show results on both training and test sets, as well as the optimization losses. These numbers suggest that our Hybrid model is overfitting to the training set less than the pure Neural model. As we focus on long-term prediction, we include the Zero baseline to show the scale and the challenging nature of the problem.

Models	trans (%)	pos (m)	velocity (m/s^2)
Zero	100.00	0.64	1.60
Physics	27.41	0.16	1.06
Neural	9.16	0.058	0.43
Hybrid	2.42	0.016	0.14

TABLE II: Our hybrid model achieves the best performance in both position and velocity estimation of the ball, compared with methods that rely on physics engines or neural nets alone.

Materials	Models	trans (%)	pos (mm)	rot (deg)
plywood	Zero	99.99	339.12	48.36
	Physics	2.51	5.49	10.38
	Neural	0.92	3.43	2.16
	Hybrid	0.77	2.16	1.65
delrin	Zero	99.99	357.98	52.67
	Physics	1.89	5.78	12.07
	Neural	0.81	2.81	2.50
	Hybrid	0.62	2.09	2.19

TABLE III: Our Hybrid model performs well consistently across object materials. Here for the rectangle made of plywood and delrin, our model again outperforms all other baseline models.

linear layer of hidden size 4 with softplus activation for standard deviation. $f_{\text{dec}}^{\text{RNN}}$ is a GRU with 2 hidden layers and a hidden size of 16, followed by a linear layer of hidden size 2 for mean, and another parallel linear layer of hidden size 2 with softplus activation for standard deviation. The decoder's standard deviation is kept fixed to identity.

Results. Table II suggests that our model is able to outperform the baselines on the synthetic dataset. The Physics baseline, designed to be deterministic, is not performing very well as expected. However, with the help of the physics engine, our Hybrid model achieves much better performance compared to the Neural model, which learns everything from scratch. The intuition is that learning from a good guess makes the learning problem significantly easier.

B. Experiments on Planar Pushing

Data. We use the MIT Push dataset [5] for the scenario of planar pushing, which contains object pose and force recordings from real robot experiments. For uncertainty modeling in particular, we use the straight-line push experiment which was repeated 2,000 times. In this experiment, the object shape is a rectangle, the contact location is half way in between the block's center and edge, the contact is made perpendicular to the edge, the speed of the pusher is set to 20 mm/s with no acceleration, and the total pusher displacement is 15cm.

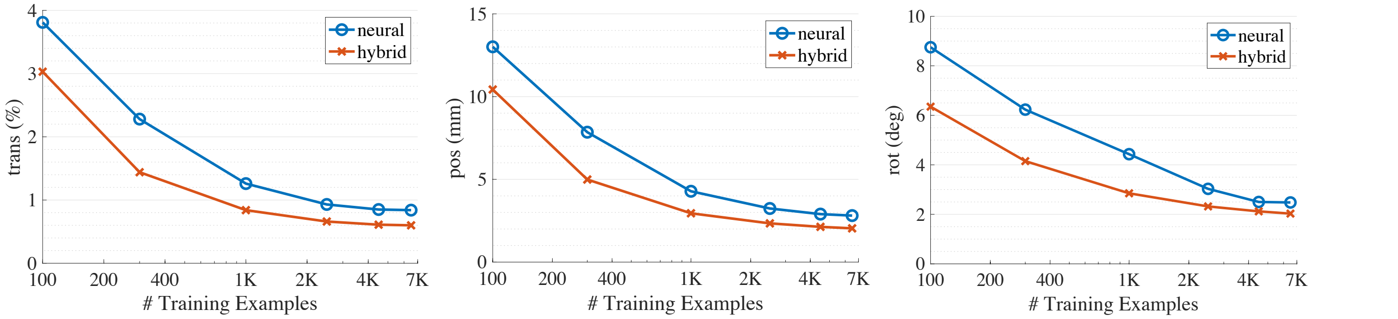
Part of the uncertainty comes of measurement noise, which we want to minimize. The object in the experiments is instrumented with reflective markers and tracked with Vicon motion tracking system. Vicon, when correctly calibrated, has 1mm or better accuracy with a unimodal distribution of noise, well approximated by a Gaussian. Because of its high fidelity, our model can focus on learning the uncertain in dynamics.

Metrics and methods. We use three metrics for evaluation, following Kloss *et al.* [12]. The first two are the average Euclidean distance between the predicted and the ground truth object reported as a percentage relative to the initial pose (trans) and as absolute values (pos) in millimeters. The third is the average error of object rotation (rot) in degree. We compare with the same baselines as in the ball experiment, except that the physics engine is now the analytical model described in Sec. IV-A.

Setup. For experiments on MIT push dataset, φ^x and φ^u consist of two bilinear layers with hidden sizes as 32 and 16 respectively and both followed by tanH activation. φ^z is a single linear layer with hidden size 16 followed by tanH activation. $f_{\text{enc}}^{\text{RNN}}$ and $f_{\text{prior}}^{\text{RNN}}$ are both GRUs with 2 hidden layers and a hidden size of 16, followed by a linear layer of hidden size 16 for mean, and another parallel linear layer of hidden size 16 with softplus activation for standard deviation. $f_{\text{dec}}^{\text{RNN}}$ is a GRU with 2 hidden layers and a hidden size of 16, followed by a linear layer of hidden size 4 for mean, and another parallel linear layer of hidden size 4 with softplus activation for standard deviation. We again use ADAM optimizer [25] with a learning rate of 10^{-3} and a weight decay of 0.5 every 5,000 iteration for a total of 50,000 iteration. Our training set contains 6,500 trajectories, while our test set contains 628 trajectories.

Results. Table I shows the main results on the MIT Push data-set, using the *rect1* object, a 837g square with a side length of 9cm, on the ABS surface. Our full model (Hybrid) significantly outperforms the baseline methods that rely only on physics engines or neural nets. Here we list results on both training and test sets. A pure neural net–based approach achieves a relatively low error on the training set, close to our Hybrid model. However, it generalizes much worse to the test set. Its prediction errors are much higher for both position estimation (2.81 *vs*. 2.04) and rotation estimation (2.48 *vs*. 2.03).

Our formulation is not constrained by the object's material. Table III shows results on objects made of two other materials: plywood and delrin. Our Hybrid model consistently outperforms the baselines.



(a) translation (%)

(b) position (mm)

(c) rotation (deg)

Fig. 5: Prediction errors *vs*. training data size. Our hybrid model not only performs better, but also requires much less data to achieve a given level of performance. In contrast, purely using purely data-driven models requires a larger training set and is not performing as well.

Our hybrid model is also more sample-efficient. As shown in Fig. 5, compared with the Neural model, our Hybrid model not only has higher prediction accuracies, but also achieves such accuracies much faster. Our model converges with as little as 2,500 training examples; in contrast, even with 6,500 training examples, purely data-driven models are not able to achieve performance comparable to ours.

Our model captures the uncertainty of the object motion well. To evaluate this, from the repeated pushes, we collect the ground truth distribution of the position of the object (*rect1*) after being pushed for one second. We then sample 2,000 points from the state-of-the-art stochastic motion modeling approach—GP-SUM [20]. We also sample 2,000 trajectories from our Hybrid model. We present qualitative and quantitative results in Fig. 6. Compared with GP-SUM, our model can better capture the underlying uncertainty. Quantitatively, we compute the Chamfer distance [26] between each model's output distribution S and the ground truth distribution T, defined as

$$\mathrm{CD}(S,T) = \frac{1}{|S|}\sum_{p\in S}\min_{q\in T}\|p-q\|_2 + \frac{1}{|T|}\sum_{q\in T}\min_{p\in S}\|p-q\|_2. \quad (19)$$

Our model achieves a lower error compared to GP-SUM.

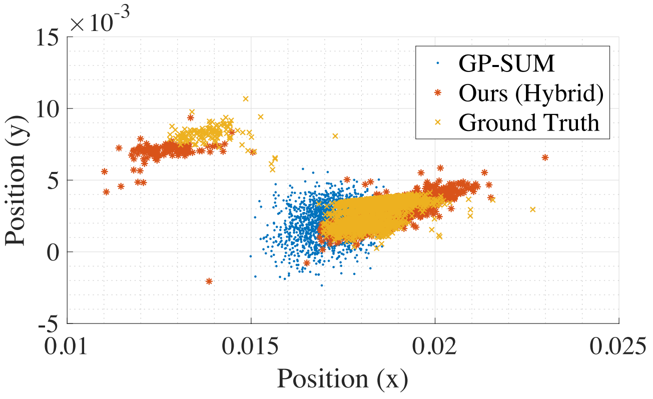
C. Generalization Power

We want our prediction models to generalize to real-world objects, which can be of any shape and material. In this section, we evaluate how our model and the baselines generalize to new object materials and shapes.

For materials, we evaluate our models predictive abilities on different surfaces. We consider the push data-set on plywood, polyurethane, and delrin. Figs. 7a and 7b summarize the results of our Hybrid model and of the Physics and Neural baselines. Our model has lower generalization errors in both position and rotation prediction.

For shapes, we evaluate our model's predictions on a new object—*rect2*, a 1045g rectangle with side lengths of 9cm and 11.26cm. Fig. 7c suggests that our model also generalizes better than the Physics and the Neural baselines on both position and rotation estimation.

We hypothesize that our model generalizes better across shapes and materials because it learns residuals—errors of the physics models that are supposed to be similar across various regimes. This assumption critically depends on the quality of the physics model; when it no longer holds, our model's generalization power may be limited.



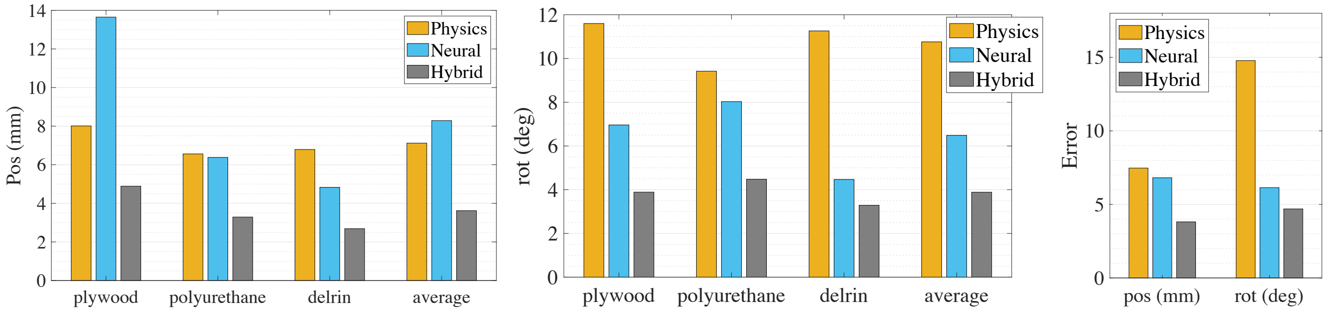
	GP-SUM [20]	Ours (Hybrid)
Chamfer Distance ($\times 10^{-4}$)	6.80	2.77

Fig. 6: Our Hybrid model captures the distribution of possible push outcomes. Measured in Chamfer distance, our model achieves a lower error compared with GP-SUM [20].

VI. Discussion and Conclusion

We have proposed a simulation framework using a data-augmented residual motion model. Our underlying philosophy is to first exploit analytical models to model real-world data as much as possible, and then to learn the remaining residuals. This residual learning formulation adapts the model to specific real-world scenarios, with little need for domain specific knowledge or hand-crafting. In this study, we have demonstrated its efficacy in predicting real-world planar pushing and its generalization power across shapes and materials. The particular choice of our residual learner enables accurate long-term predictions and generates complex posterior distributions over future states. The improved accuracy may be attributed to the model's implicitly learning of the details in object-surface frictional interactions, so that it can account for variations in the coefficient of friction and potentially anisotropic friction.

One may be tempted to do away with the analytic model entirely and rely on a purely data-driven approach. This approach results in learning the full motion model from scratch with no priors. Aside from being more data-hungry, this approach does not generalize well to other shapes and materials. Starting from a good initial guess to the trajectory,



(a) Position prediction on different materials (b) Rotation prediction on different materials (c) Shape generalization

Fig. 7: Our method generalizes well to different materials and to a new shape (*rect2* in the push dataset). The error bars show our hybrid model achieves a consistently lower generalization error for both position and rotation prediction, compared with baseline methods.

the residual model has less to learn and generalizes better.

No model will ever be perfect; therefore, our model's ability in reasoning about possible outcomes can be a powerful tool in planning and control. For example, in the context of the planar pushing task, identifying a predictable push can lead the planner to exploit this property.

While the data-augmented residual model is more accurate and reasons about uncertainty, it does have certain drawbacks. Given that the residual model is data-driven, it can predict outcomes that are physically impossible, because there is no mechanism to enforce basic physics principles (*e.g.*, the conservation of energy) at the output level. In practice, however, these implausible outcomes would not appear if we can have the model sufficiently trained.

The simulation framework only requires a coarse domain-specific analytical motion model to be applied to other robotic tasks. For future work, we plan on applying the framework to more difficult modeling tasks. In this paper's experimental setup, we have assumed access to the noisy full state space; an interesting extension would be to learn an observation model along with the residual model to simultaneously perform state estimation and prediction.

Acknowledgement This work is supported by NSF #1420316, #1523767, and #1723381, AFOSR grant FA9550-17-1-0165, ONR MURI N00014-16-1-2007, Toyota Research Institute, Honda Research, Facebook, and Draper Laboratory. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

References

[1] R. Smith, "Open Dynamics Engine (ODE)," 2006.

[2] E. Coumans, "Bullet physics engine," *Open Source Software: http://bulletphysics. org*, 2010.

[3] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 5026–5033.

[4] R. Kolbert, N. Chavan Dafle, and A. Rodriguez, "Experimental Validation of Contact Dynamics for In-Hand Manipulation," in *International Symposium on Experimental Robotics (ISER)*, 2016.

[5] 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 (IROS)*, 2016.

[6] N. Fazeli, S. Zapolsky, E. Drumwright, and A. Rodriguez, "Fundamental limitations in performance and interpretability of common planar rigid-body contact models," in *International Symposium on Robotics Research (ISRR)*, 2017.

[7] S. Goyal, A. Ruina, and J. Papadopoulos, "Planar Sliding with Dry Friction Part 1 . Limit Surface and Moment Function," *Wear*, vol. 143, pp. 307–330, 1991.

[8] R. D. Howe and M. R. Cutkosky, "Practical force-motion models for sliding manipulation," *Int. J. Robotics Res.*, vol. 15, no. 6, pp. 557–572, 1996.

[9] 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 (IROS)*, 1992.

[10] F. Hogan and A. Rodriguez, "Feedback Control of the Pusher-Slider System: A Story of Hybrid and Underactuated Contact Dynamics," in *Workshop on Algorithmic Foundation of Robotics (WAFR)*, 2016.

[11] K.-T. Yu, J. Leonard, and A. Rodriguez, "Shape and pose recovery from planar pushing," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 1208–1215.

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

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

[14] J. Zhou, R. Paolini, A. Bagnell, and M. T. Mason, "A convex polynomial force-motion model for planar sliding: Identification and application," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 372–377.

[15] J. Zhou, A. Bagnell, and M. T. Mason, "A fast stochastic contact model for planar pushing and grasping: Theory and experimental validation," in *Robotics: Science and Systems (RSS)*, 2017.

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

[17] M. Bauza and A. Rodriguez, "A probabilistic data-driven model for planar pushing," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[18] M. Babaeizadeh, C. Finn, D. Erhan, R. H. Campbell, and S. Levine, "Stochastic variational video prediction," *arXiv:1710.11252*, 2017.

[19] T. Xue, J. Wu, K. Bouman, and B. Freeman, "Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks," in *Neural Information Processing Systems (NIPS)*, 2016, pp. 91–99.

[20] M. Bauza and A. Rodriguez, "GP-SUM. gaussian processes filtering of non-gaussian beliefs," *arXiv preprint arXiv:1709.08120*, 2017.

[21] J. Zhou, R. Paolini, A. M. Johnson, J. A. Bagnell, and M. T. Mason, "A probabilistic planning framework for planar grasping under uncertainty," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2111–2118, 2017.

[22] K. Cho, B. V. Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv:1409.1259*, 2014.

[23] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio, "A recurrent latent variable model for sequential data," in *Neural Information Processing Systems (NIPS)*, 2015.

[24] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *International Conference on Learning Representations (ICLR)*, 2014.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[26] H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf, "Parametric correspondence and chamfer matching: Two new techniques for image matching," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 1977.