A Appendix

Contents

A.1 Training Details

Tab. 4 contains training parameters. In addition, we used 2,000 on-surface points and 20,000 off-surfaces for training VIRDO++ . We used the Adam optimizer with learning rate = 10^{-4} and epoch = 200.

l_c	λ	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9	λ_{10}	λ_{11}
6	5e1	1e1	1e1	1e4	1	1e2	1e5	5e1	3e6	1e1	1e1	1e1

Table. 4: Parameters used for training VIRDO++

A.2 Inference Details

A.2.1 Parameters & Hardware

For the inference, we use number of particles $n = 40$, resampling parameter $\gamma = 0.7$, and epoch $e = 10$. This results in inference time = 0.7 [s] with GPU = NVIDIA A6000 and CPU= 32-Core 3.50 GHz AMD. We note that one can balance the state-estimation accuracy between the inference time by changing the number of particles and number of epochs. Here, we note that the epochs do not need to be large due to Alg. 1 which provides a close-to-ground-truth initial guess on the object's state.

A.2.2 Inference Algorithm Variations

Algorithm 2 Resampling Algorithm

1: **function** RE-SAMPLING($\boldsymbol{w}_t, \mathbf{C}_t$)
2: Draw $\{\bar{\boldsymbol{C}}_t^0, \bar{\boldsymbol{C}}_t^1, ..., \bar{\boldsymbol{C}}_t^{k-1}\} \sim \mathcal{N}(0, 0.01)$ ▷ Exploration
3: Draw $\{\bar{\boldsymbol{C}}_t^k, \bar{\boldsymbol{C}}_t^{k+1}, ..., \bar{\boldsymbol{C}}_t^{N-1}\} \sim \{(\boldsymbol{C}_t^0, w_t^0), (\boldsymbol{C}_t^1, w_t^1)..., (\boldsymbol{C}_t^{N-1}, w_t^{N-1})\}$ ▷ Exploitation
4: **return** $\bar{\mathbf{C}}_t$
5: **end function**

The input of the resampling function in Alg. 1 is the beliefs over the current states ($\mathbf{C}_t$) and the corresponding reliability, a.k.a the sampling weight ($\boldsymbol{w}_t$). The resampling function refines $\mathbf{C}_t$ to $\bar{\mathbf{C}}_t$ through Alg. 2. In the

CD ($x10^3$)	$\beta = 0\%$	$\beta = 25\%$	$\beta = 50\%$	$\beta = 75\%$
Geom. – S.E.	0.936 (0.206)	0.937 (0.209)	0.941 (0.236)	0.943 (0.202)
Geom. – Pred.	1.035 (0.335)	1.046 (0.358)	1.049 (0.370)	1.117 (0.497)
Force – Pred.	1.241 (1.282))	1.125 (1.068)	1.149 (1.069)	1.326 (1.282)
Torque – Pred.	0.140 (0.109)	0.134 (0.099)	0.134 (0.095)	0.158 (0.111)

Table. 5: $\beta = 0 \sim 50\%$ does not significantly increase nor decrease the performance of our Inference algorithm. However, when exploration becomes dominant over the exploitation ($\beta = 75\%$), a distinguishable drop in performance is observed.

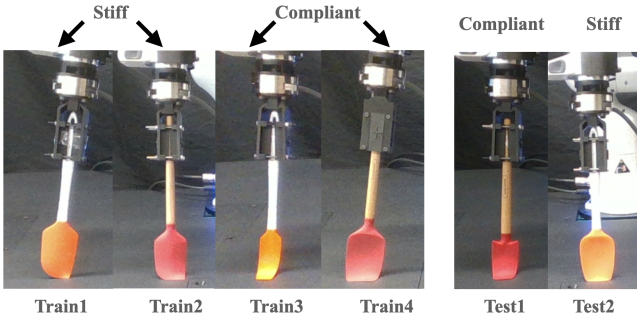

Fig. 8: We maintained the pressing force of 6 [N] for all six spatulas and measured the difference in deformations. The combination of material property and geometry determines stiffness/compliance. The spatulas with the same tip color consist of the same material; however, they deform very differently depending on their structures.

exploration step, we compose k particles in $\bar{\mathbf{C}}_t$ by sampling from $\sim \mathcal{N}(0, 0.01)$. In the exploitation step, we sample the rest of $\bar{\mathbf{C}}_t$ by sampling from $\boldsymbol{w}_t$.

Our ablation study (presented in Tab. 5) aims to find an optimal balance between exploration and exploitation. Here, the "Force-Prediction" is calculated as $\|\hat{\boldsymbol{f}}_t[:3] - \boldsymbol{f}_t[:3]\|$ and "Torque-Prediction" is calculated as $\|\hat{\boldsymbol{f}}_t[3:] - \boldsymbol{f}_t[3:]\|$. We found that $\beta = 0$ and 25% has comparable state estimation results. Geometry prediction is the most accurate when $\beta = 0\%$; however, wrench predictions are most accurate when $\beta = 25\%$. When compared to $\beta = 0\%$, $\beta = 25\%$ has a 10% increased geometry prediction error and $9 - 10\%$ reduced wrench prediction error.

A.3 Experiment Details

A.3.1 Differences in Physical Properties of Spatulas

In this section, we illustrate how the train/test spatulas deform relative to each other when brought into contact with the table in a similar configurations (affected by the arm impedance controller) and same normal force (here, 6 [N] along the z axis). Fig. 8 shows a wide distribution of deformations achieved within the same object category. With the same force, train object 1 barely deforms and is close to it's nominal shape, whereas train object 3 bends significantly. This difference in physical properties makes generalization very difficult, even within the same object class. For example, test object 2 visually resembles train object 4; however, it has stiffness properties more similar to train object 2. This has lead to VIRDO++ producing higher errors in the generalization task for test 2 objects.

A.3.2 Coordinate System and Action Distribution

Pointcloud are collected with respect to a reference frame defined at the end of the F/T sensor mounted at the robot's wrist. Actions are the Cartesian motion of the end-effector of the robot with respect to the base frame, Fig. 9. Fig. 10 visualizes action distributions of train and test trajectories of training objects. Both are normally distributed and share mean and standard deviation because they were generated from the same action policy.

A.3.3 Distribution of Geometry and Wrench State Estimation and Predictions

Fig. 11 shows detailed distribution of errors reported in Tab. 1. The Kernel Density Estimate (KDE) plot shows that the errors are normally distributed. The training error mean is smaller than the test error for all graphs

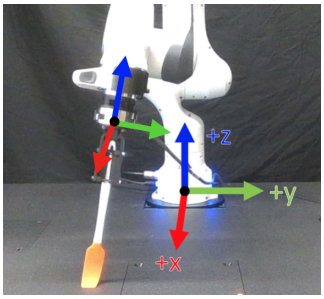

Fig. 9: Action frame is defined at the robot's base frame while pointclouds are registered with respect to the end-effector frame.

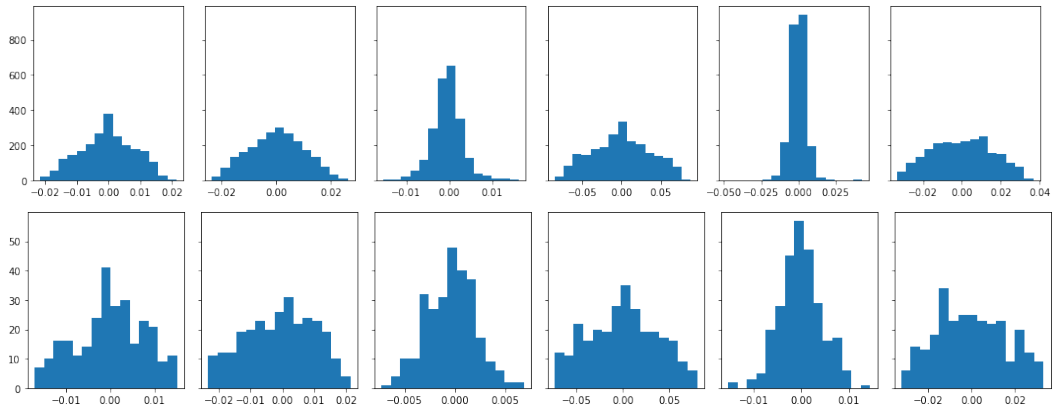

Fig. 10: Distributions of training and testing action trajectories. These distributions are intentionally centered around zero. Top row = Train, Bottom row = Test. $\delta x, \delta y, \delta z, \delta r, \delta p, \delta y$ order. y-axis quantifies the number of samples.

in both demos. In addition, geometry estimation is more accurate than geometry predictions as it uses sensor measurements. For example, in the spatula demo, the Chamfer distance error at the peak geometry estimation from the training set (green dashed line) is smaller than the test set (solid dashed line) and is also smaller than the prediction estimates (yellow dashed line).

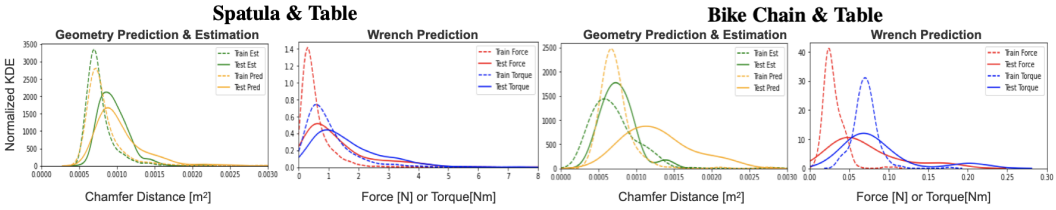

Fig. 11: Kernel Density Estimate (KDE) plots for spatula and bike chain applications show that VIRDO++ 's geometry and wrench predictions and state-estimation errors are within small margins. Force prediction errors = $\|\hat{\boldsymbol{f}}_t[:3] - \boldsymbol{f}_t[:3]\|$ [N]. Torque prediction error as $10 * \|\hat{\boldsymbol{f}}_t[3:] - \boldsymbol{f}_t[3:]\|$ [Nm].

A.3.4 Reaction Wrench Prediction Errors

Tab. 6 shows the wrench predictions results of trained objects (Tab. 3's first column). Because we use L2 norm for reaction wrench prediction during training, error distributions are centered around 0, and the degree of errors appears as standard deviation.

The standard deviation in error for the chain task is approximately an order of magnitude smaller than the spatula. We hypothesize that this because i) the ground truth wrench is an order of magnitude smaller than the spatula and ii) the action space dimension is half the spatula resulting in a simpler model to learn.

l_1 Error	f_x [N]	f_y [N]	f_z [N]	τ_x [Nm]	τ_y [Nm]	τ_z [Nm]
Sp. Train	0.01 (0.31)	-0.00 (0.36)	0.03 (0.65)	-0.00 (0.11)	0.00 (0.09)	0.00 (0.02)
Sp. Test	-0.03 (0.35)	0.08 (0.54)	-0.28 (1.60)	-0.02 (0.16)	-0.01 (0.11)	-0.00 (0.02)
Ch. Train	0.02 (0.01)	0.01 (0.01)	0.02 (0.01)	-0.01 (0.00)	0.00 (0.01)	0.01 (0.00)
Ch. Test	0.02 (0.06)	0.01 (0.03)	0.03 (0.01)	-0.00 (0.00)	0.00 (0.01)	0.00 (0.00)

Table. 6: **Mean and Std Wrench Prediction Errors:** $\hat{\boldsymbol{f}}_t - \boldsymbol{f}_t$ for the spatula and chain experiments.

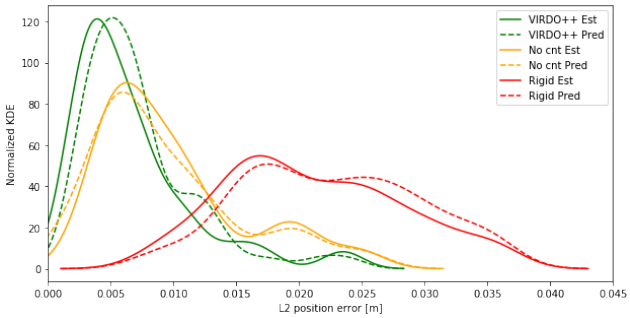

Fig. 12: VIRDO++ 's contact detection outperforms the baselines without c_t and rigid body assumption where the detailed quantitative analysis is in Tab. 2

A.3.5 Details of Contact Detection Errors

Fig. 12 contains detailed error distributions of Tab. 2 from the main text. VIRDO++ (green)'s Chamfer distance has a significantly smaller mean and standard deviation compared to the baseline models without contact embedding and compliance assumption. One interesting observation is that the geometry prediction (yellow dashed line) is located left to the geometry estimation (solid yellow line). This is because the model without c_t produces deterministic reconstructions as a function of p_t and f_t, leaving no room for improving state estimation with the Bayesian filtering algorithm.

A.3.6 Details on Generalization to Novel Objects Dynamics

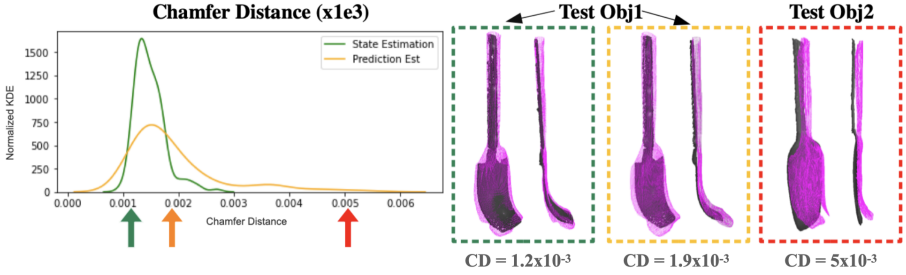

Fig. 13: **Generalizing to unseen object dynamics**. Left panel: KDE plot of Chamfer distances (x1e3) $[m^2]$ using normalized geometries. We show both test objects state estimation using partial point clouds and geometry predictions. Similar to training objects, state estimation has smaller mean and standard deviation of the error. Right panel: We visualized three representative generalization results and corresponding Chamfer distances.

Fig. 13 is a detailed Chamfer distance distribution of Tab. 3 from the main text and the visualization of each Chamfer distance loss. The purpose of reconstructions visualized in Fig. 13 (right) is to further provide an intuition of each error and how to interpret its magnitude. For example, the Chamfer distance value of 0.002 for the unseen objects is about double the amount of the training objects' average state estimation error; however, in visualization, the reconstructions can reasonably predict the ground truth's curvature. We observed that the majority of errors are due to small undesirable artifacts at the edges of objects which do not significantly impact the general trend of deformations.

A.4 Generalization to Novel Environments

In this section, we show VIRDO++ generalization capability to novel environments. Here, the novel environment is a wok scraping task. The wok presents high curvatures producing greater variances in contact formations with respect to the flat table. Here, we add a training dataset with 2 scraping trajectories on the wok and perform state estimation and predictions given a test trajectory with 13 transitions. As in Fig. 14, VIRDO++ provides high-fidelity dense geometry of deformable objects in high occlusion cases without a 3D

model or configuration of the environment; however, due to the complex environment, it occasionally produces undesired artifacts (2 out of 13 trans.).

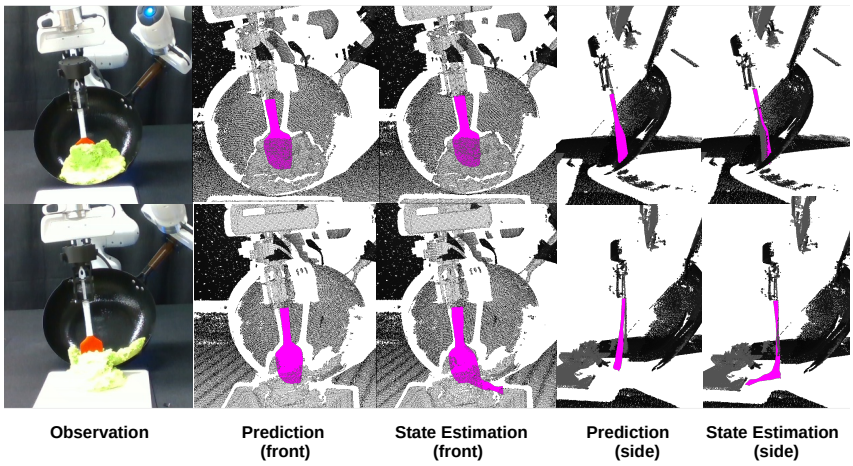

Fig. 14: VIRDO++ can reconstruct high-fidelty dense geometry of deformable objects under occlusions in new environments, such as scraping objects off a wok (successes, top row), but can also at times produce undesirable artifacts (failure modes shown in bottom row).

A.5 Ablation study on design choices

A.5.1 Explicit Geometry Representation

In this section, we compare VIRDO++ with explicit models by replacing the geometry representations (deformation and object module) with GRNet's pointcloud encoder-decoder structure [48]. Here, we use the same force module as VIRDO++ because the baseline does not have a multi-modal sensory encoder. The explicit dynamics model directly takes in a partial pointcloud at time t to generate a dense pointcloud at time $t+1$. The inputs to the dynamics model are pointcloud at the current time step P_t, force code z_t, contact latent vectorc_t, and action a_t and the output is full pointcloud at the next time step P_{t+1}. P_t is directly input to [48]'s encoder and other inputs are concatenated to the bottle-neck features.

Here, we used the same dataset in Sec. 4.1. For the loss function, we replaced $\mathcal{L}_t^{geo}$ to Chamfer distance as [48] proposed. The second column of Tab. 7 shows the result of the explicit dynamics model based on [48], where the learned explicit model showed better result than VIRDO++ for training trajectories. However, the performance dropped significantly given the pointcloud with occlusions from the test trajectories. We speculate that because the explicit dynamics model directly consumes the partial pointcloud to roll out the dynamics, it is more susceptible to visual outliers caused by occlusions. However, VIRDO++ is more robust to occlusions due to its implicit nature, using partial pointcloud implicitely via solving the optimization problem in Sec. 3.4

Besides dealing with occlusions, VIRDO++'s implicit dynamics model is also advantageous over the explicit counterparts in multi-object interactions. Specifically, the detection of contact and collision at a query point can easily be done by evaluating the sign-distances from each object's signed distance field at a time complexity of $\mathcal{O}(number\ of\ objects)$. On the other hand, explicit geometric representations require special contact/collision detection techniques such as k-d tree [49] or sphere coverings [50]. These methods have higher time complexities dependent on the length of each point cloud [51, 50, 49]. In Sec.4.3, we demonstrate the accurateness of VIRDO++ for detecting contact location.

A.5.2 Implicit Geometry Representation with Different Model Architectures

In this section, we evaluate variations of VIRDO's geometry representation using DeepSDF [37], SIREN [36], and a positional encoder [52]. In this ablation study we first attempted to replace the VIRDO++ 's entire geometry representation modules (i.e., both deformation and shape module) with [37], [36], and [52], then train each model end to end with the VIRDO++ 's force and action modules. Under this condition, we set the input as $(\boldsymbol{x}, \alpha, z_t)$ and the output as a signed distance function $s \in \mathbb{R}$, where $\boldsymbol{x}$ is the query point instead of the entire pointcloud as Sec. A.5.1. However, training end to end without residual layers resulted in failure of convergence for all variations.

We observed that fixing the weights of the pre-trained object modules and learning only the residuals (deformations) significantly helped the models converge. We chose to replace only the deformation module with the proposed variations and compare performance. The inputs are $(\boldsymbol{x}, \alpha, z_t)$ and the output is $\Delta\boldsymbol{x}$. We used the

CD ($\times 10^3$)	VIRDO++	GRNet	DeepSDF	SIREN	Pos. Enc.
Train Est.	0.777 (0.215)	–	5.087 (0.737)	5.521 (1.015)	5.569 (2.847)
Train Pred.	0.830 (0.330)	0.759 (0.667)	5.667 (1.306)	5.457 (0.888)	5.886 (2.561)
Test Est.	0.934 (0.210)	–	10.016 (3.253)	6.349 (1.388)	8.081 (3.790)
Test Pred.	1.041 (0.348)	1.230 (0.826)	9.552 (2.767)	6.988 (1.280)	7.188 (3.603)

Table. 7: Results are evaluated in the real-world, showing that VIRDO++ outperforms its implicit and explicit counterparts.

same dataset and loss functions as VIRDO++ except for training DeepSDF, where we removed $\mathcal{L}_{hyper}$ when training DeepSDF as the architecture does not have a hyper-network.

Template-based DeepSDF: For an apple-to-apple comparison, we set the number of hidden layers = 2 and hidden features = 256 for DeepSDF structures to the same size in VIRDO++ 's deformation module. We observe that Chamfer distance error becomes $6.54 \sim 6.82$ time greater for the training trajectories and $9.18 \sim 10.72$ times greater for the unseen trajectories when changing the hypernetwork to the DeepSDF architecture. The hypernetwork structure has higher representation power, resulting in more precise geometry reconstructions.

SIREN: Here, we replaced the geometry representation with SIREN's hypernetwork and trained it with the force and action modules. As suggested in the paper, we implemented SIREN with 5 layers and hidden features of size 256. HyperNetwork (ReLU MLP) was initialized with Kaiming initialization, and the HypoNetwork was initialized as $w_i \sim \mathcal{U}(-1/l, +1/l)$, where w_i is the network's weight and l is the total number of network parameters. We found that the sine activation function generates bumpy textures resulting in high reconstruction errors for geometry estimation and predictions.

Positional Encoder: For this experiment, we add a positional encoder from [52] right before VIRDO++ 's deformation modules. Similar to what we observed from SIREN, adding sine or cosine non-linearity to the network can produce undesirable artifacts. We found that the positional encoder generate floating blobs in the free space, resulting in much higher errors than the original VIRDO++ .