

Figure A.1: Architecture Details for the Components of the Contact Feature Dynamics Model.

## Appendix A Network Architecture Details

Here, we describe in detail the network architecture of our contact feature dynamics model introduced in Sec. 4.1. Our network has three main components, an encoder  $e$ , dynamics module  $f$ , and decoder  $d$ :

1. **Encoder:** The encoder takes in a pointcloud  $\mathbf{v}_0 \in \mathbb{R}^{P \times 3}$  and four most recent wrench values from the force/torque sensor  $\mathbf{h}_0 \in \mathbb{R}^{4 \times 6}$ . The pointcloud is encoded using a PointNet encoder [29], designed to specifically handle unstructured pointclouds. We encode the wrench inputs by flattening to a vector length 24 and passing through a Multi-Layer Perceptron (MLP) of three layers with hidden sizes 64 and 64, with ReLU non-linearities. The output of both the visual and wrench encodings are latent vectors of size 64 that are concatenated to yield the final latent state code  $\mathbf{z}_t \in \mathbb{R}^{128}$ . The “Vision-Only” model does not have the MLP for tactile, and instead outputs  $\mathbf{z}_t \in \mathbb{R}^{128}$  directly from the PointNet encoder.
2. **Dynamics:** The dynamics module has three MLP modules. First is a single layer network to increase the dimensionality of the action  $\mathbf{a}_t$  from 6 to a vector  $\mathbf{z}_t^a \in \mathbb{R}^{64}$ . This vector is concatenated with the latent vector  $\mathbf{z}_t$  to yield the combined state action latent vector,  $\mathbf{z}_t^q \in \mathbb{R}^{192}$ . The second MLP module takes in  $\mathbf{z}_t^q$  and passes through three layers, with hidden sizes 256 and 128 and ReLU non-linearities, yielding the next latent state code  $\mathbf{z}_{t+1} \in \mathbb{R}^{128}$ . The third MLP module takes in  $\mathbf{z}_t^q$  and passes through three layers, with hidden sizes 128 and 64 and ReLU non-linearities, yielding the action offset  $\Delta \mathbf{a}_{t+1} \in \mathbb{R}^6$ . The first three terms of  $\Delta \mathbf{a}_{t+1}$  is the translation and the second three terms are the axis-angle rotation for the delta action. The “No Offset” model does not contain this last MLP module, as it does not predict the offset action.
3. **Decoder:** The decoder module has three MLP modules that predict each component of the contact feature. Each network takes in the current latent state code  $\mathbf{z}_t \in \mathbb{R}^{128}$ . The first MLP is a classification head, predicting the binary contact state. The network has a single hidden layer of size 64, with a ReLU non-linearity. The final prediction is passed through a Sigmoid to recover the likelihood of  $c_t^b = 1$ , i.e., likelihood the system is in contact. The second MLP regresses the contact lines. The network has 3 layers, with hidden sizes 128 and 64 and ReLU non-linearities. The output of the network is a vector  $\mathbf{c}^l \in \mathbb{R}^6$  which is reshaped to be  $\mathbf{c}^l \in \mathbb{R}^{2 \times 3}$ , interpreted to be the two endpoints of the contact line in 3D space. The final MLP regresses the end-effector wrench. The network has 3 layers with hidden size 128 and 64 and ReLU non-linearities. The final prediction is a wrench  $\mathbf{c}_t^w \in \mathbb{R}^6$ .

The detailed module architectures are shown in Fig. A.1.

## Appendix B Data Collection

### B.1 Extrinsic Contact Dynamics Labeling Examples

Here we show qualitative examples of labeled contact lines, collected on a real world system as described in Sec. 4.3. Fig. B.1 visualizes our procedure of using a Photoneo PhoXi 3D Scanner to recover contact lines from high fidelity pointclouds. Fig. B.2 shows several examples of labels

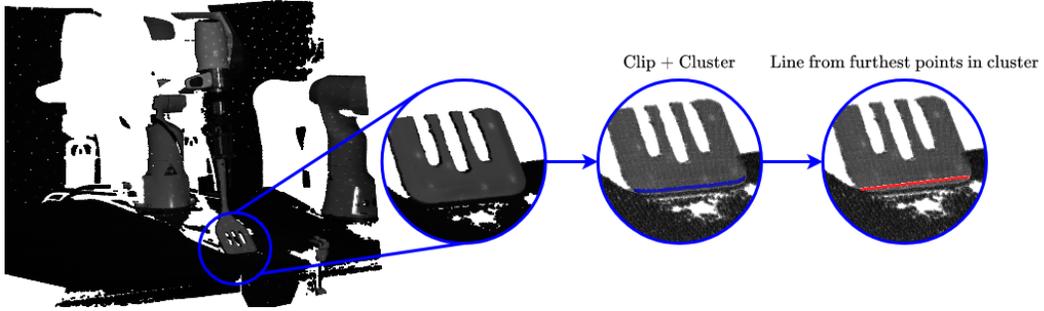


Figure B.1: Examples of labeling contact lines automatically from high fidelity point clouds.

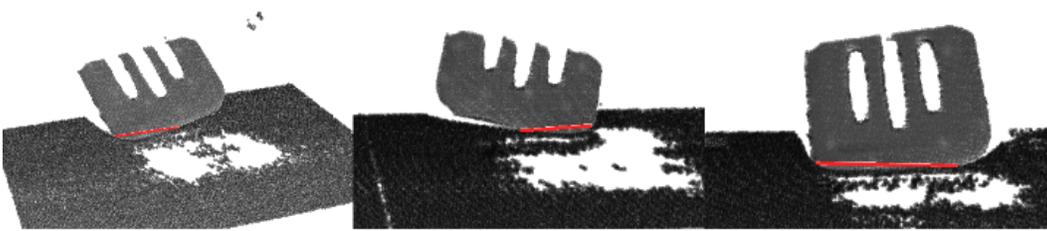


Figure B.2: Labeled contact lines from various tool-environment contact scenarios. Our labeling procedure automatically derives accurate contact lines.

from different tool-environment scenarios. We see our labeling procedure can automatically recover accurate contact lines.

## B.2 Data Augmentation

Here we show examples of augmented pointclouds from our dataset, as described for the “With Obstacles” case in Sec. 5.4. Fig. B.3 shows examples of pointclouds with randomly generated ellipsoids inserted to provide occlusions during training. Note, that while the ellipsoids here are colored green, we only input the point positions into the network.

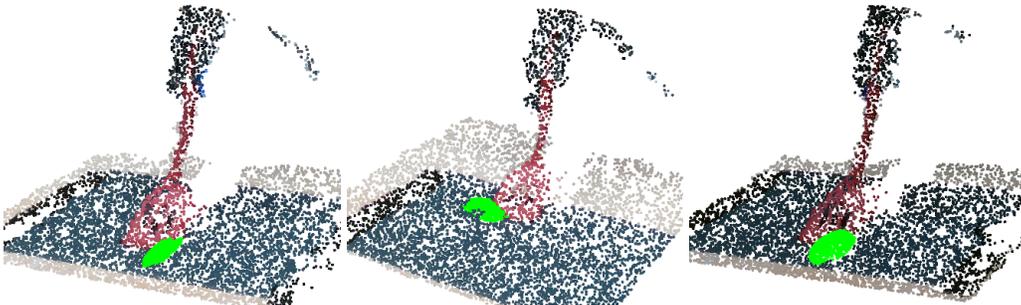


Figure B.3: Examples of input pointclouds augmented with randomly sampled ellipsoids (in green) to encourage robustness to visual occlusions. Note: color is not input to our model.

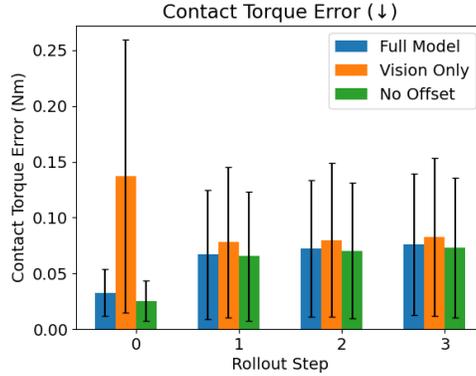


Figure C.1: Model performance in future end effector torque prediction. Similar to the case of force prediction, we are able to accurately model future torques. Learning without wrench inputs struggles to accurately predict future torques.

## Appendix C Additional Results

### C.1 Modeling Contact Feature Dynamics

#### C.1.1 Torque Prediction Results

In Fig. C.1 we show the performance of all models discussed in Sec. 5.3 on predicting end effector *torque*. Similar to force prediction quality, “Full Model” outperformed “Vision-Only” due to having access to wrench inputs. This makes predicting 0th step wrench equivalent to reconstruction, and helps the model predict future torques accurately. “Vision-Only” performs better at predicting future wrench values - we think this could be because the *action* is highly discriminative with regards to the resulting wrench.

#### C.1.2 Performance on Unseen Tool Data

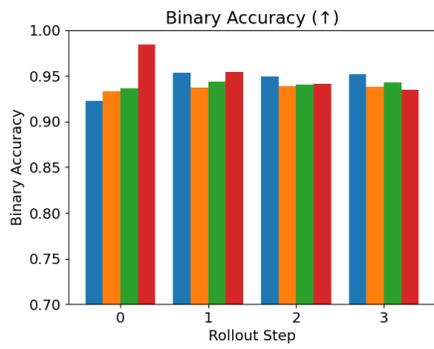
We also test our model performance when running our model on data from a tool unseen during training (right-most tool in Fig. 5a). The prediction performance is shown in Fig. C.2. Overall, the results indicate that our proposed method generalizes well to the unseen tool, with high accuracy on binary contact, roughly 1cm error on the contact line, and similar errors on force and torque as those found for the training tools. Additionally, our method outperformed the rigid body baseline (Sec. 5.2) on the contact line error and performed comparably on binary accuracy. Note that the baseline here is given access to the unseen spatula geometry still, and thus has more information than our method.

### C.2 Obstacle Scraping Results

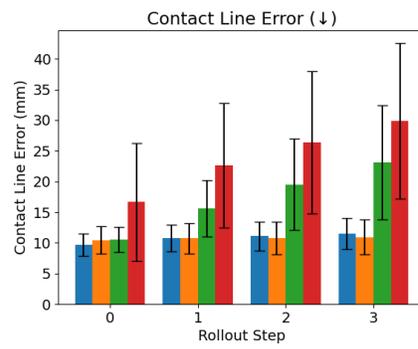
Here we show qualitative results of our target object footprint metric. Fig. C.3 shows the masks generated before and after several scraping examples that show how we estimate footprint removed. These also convey the difficulty of this metric; even small errors in the contact line can leave residue which is picked up by our masking procedure, lowering the score.

We segment the starting object footprint using a binary segmentation method, tuned by hand to accurately capture the starting footprint. For the finished object footprint, we compare each pixel in the color space to the rough nominal color of the target object. We then apply a threshold to choose which points we consider to still contain the object. We manually select a threshold such that a very small amount of remaining residue is not captured, while thicker remaining areas are penalized.

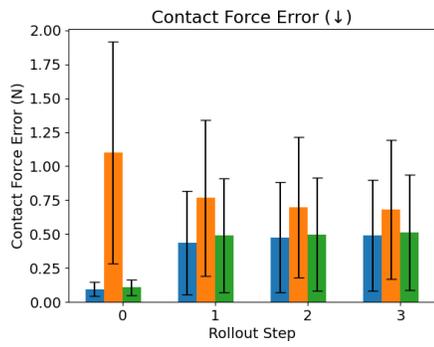
We balance this metric with the percent mass removed metric. The mass metric is easier to perform well on, as the residue left on the surface often has far less mass than the overall obstacle object to be removed. Between the two metrics, we believe our results show early indication that our method of modeling tool-environment interactions allows us to solve interesting contact servoing tasks, even in the presence of visual occlusions and novel reaction forces.



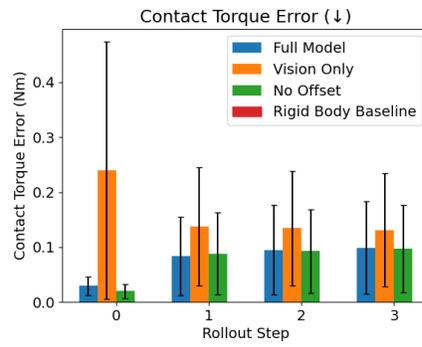
(a)



(b)



(c)



(d)

Figure C.2: Contact Feature Dynamics Performance on Unseen Tool

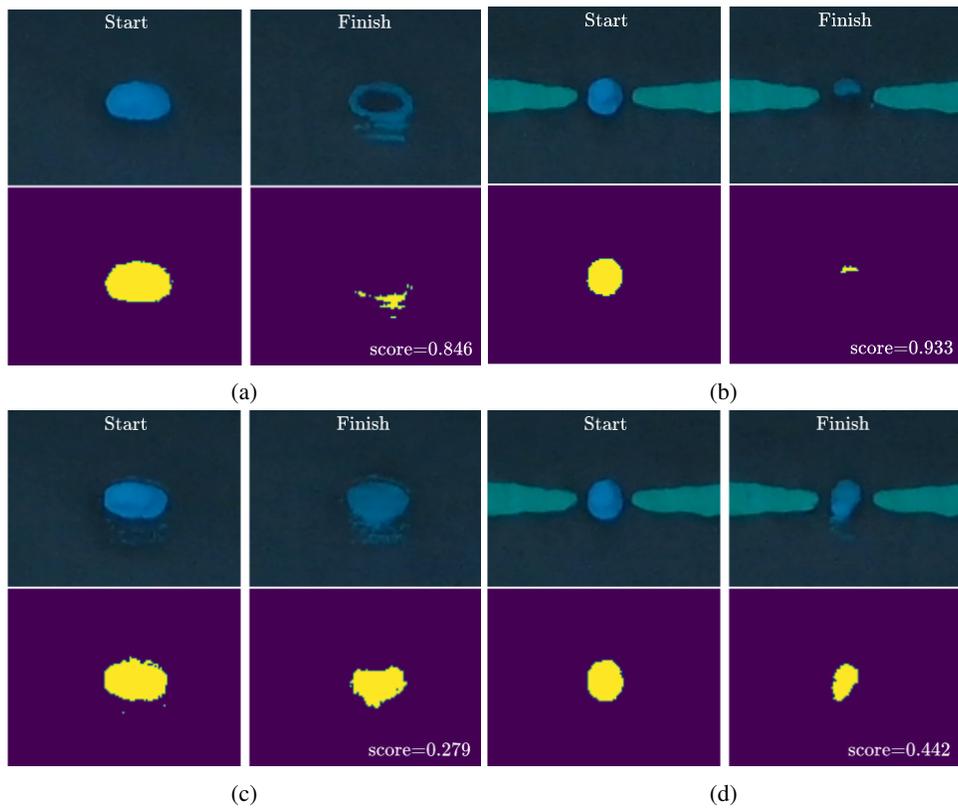


Figure C.3: Qualitative Results showing pre and post-scraper footprint masks, used to generate Percent Footprint Removed metric. The corresponding score is shown for each run.