Supplementary Material for Form2Fit: Learning Shape Priors for Generalizable Assembly from Disassembly

Kevin Zakka[1,2], Andy Zeng[2], Johnny Lee[2], Shuran Song[2,3]
[1]Stanford University [2]Google [3]Columbia University
https://form2fit.github.io/

This document contains additional details on the representations, network architecture, model training, data collection and experiments. Our models are trained from scratch in PyTorch with an NVIDIA Titan V on an Intel Xeon E-2136 6-core processor clocked at 3.3GHz.

I. Visual Representation

The PhoXi sensor outputs a 32-bit intensity map and a 32-bit depth map at 1 FPS. We convert the intensity map to an 8-bit grayscale image and normalize the depth map to meters. Grayscale and depth images are converted to heightmaps (Fig. 1) and used as visual representations for our system.

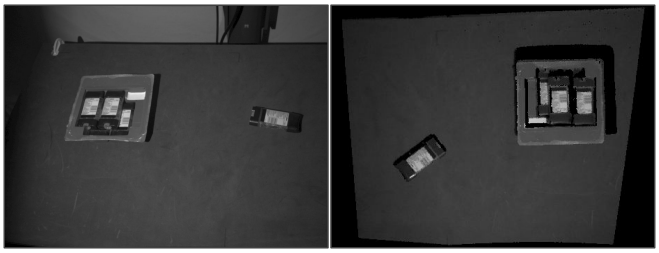

Fig. 1: Example of an intensity image (left) captured by the PhoXi sensor and its equivalent heightmap representation (right).

We perform background subtraction on the resulting heightmaps using the depth image. Empirically, we observe that this helps reduce overfitting.

II. Network Architecture

The three parameterized modules of Form2Fit (*i.e.*, suction, place and matching) use the same neural network architecture: a 20-layer, fully-convolutional, dilated residual network ([1], [2], [3]) as shown in Table I. The residual block consists of two Conv3x3-BatchNorm-ReLU operators with a skip-connection as defined in [1].

The matching network is a Siamese variant with shared weights for both input streams. We feed it a batch of 20 rotations of the kit heightmap I_{kit} for the first stream, and the object heightmap I_{obj} for the second stream. Thus, for a single data point, the matching network has an effective batch size of 21.

III. Loss Functions

Suction and place networks. Suction and place networks are trained with the weighted sum of two loss functions: a binary cross entropy loss with a coefficient of 1 and a dice loss with a coefficient of 5. The dice loss naturally helps account for class imbalance and we observe that it produces more concentrated heatmaps. We note that the data collection process generates single ground truth suction and place pixels (*i.e.*, p and q). We can artificially increase their number by creating a radius r of pixels centered at p and q and passing gradients through the pixels within this radius. All other pixels backpropagate with 0 loss. We set the hyperparameter $r = 6$.

Matching network. The matching module is trained using a pixel-wise contrastive loss, where for every pair of kit and object heightmaps $(I_{\text{kit}}, I_{\text{obj}})$, we sample non-matches from I_{obj} and all 20 rotations of I_{kit} and matches from I_{obj} but only the rotation j of I_{kit} corresponding to the ground-truth angle. The loss function thus encourages descriptors to match solely at the correct rotation of the kit image while non-matches are pushed to be at least a feature distance margin M apart. In our experiments, the margin parameter $M = 8$. At each training iteration, we sample match and non-matches with a 1:5 ratio and compute the loss: $\mathscr{L} = \mathscr{L}_{\text{match}} + \mathscr{L}_{\text{nonmatch}}$ where $\mathscr{L}_{\text{match}}(I_{\text{kit}}, I_{\text{obj}}) = \frac{1}{N_{\text{match}}} \sum_{N_{\text{match}}} \|\mu^j_{\text{kit},i} - \mu_{\text{obj},i}\|_2^2$ and $\mathscr{L}_{\text{nonmatch}}(I_{\text{kit}}, I_{\text{obj}}) = \frac{1}{N_{\text{match}}} \sum_{j \in \mathbb{Z}: j \in [1,20]} \sum_{N_{\text{nonmatch}}} \max\left(0, M - \|\mu^j_{\text{kit},i} - \mu_{\text{obj},i}\|_2\right)^2$. We couple the contrastive loss with hard negative mining and find that it is crucial for successful training.

IV. Data Collection & Training Details

Suction network pretraining. We start by pretraining a suction network for each kit on a small dataset of manually collected demonstrations. We find that it is way more cost-efficient to collect a few demonstrations and bootstrap the learning than starting from a randomly initialized policy. Each demonstration corresponds to one object disassembly (*e.g.* disassembling the deodorant kit once produces 5 demonstrations). One demonstration produces two suction labels (1 when the object is inside the kit, 1 when the object is on the table) effectively doubling the dataset size. We observe that ~ 50 demonstrations, augmented with random rotations and translations and trained for 75 epochs, is sufficient to train a network that can produce accurate suction affordances.

Online learning. After the pretraining, we plug the suction network into the automatic data collection pipeline and it continues to learn from trial and error using the suction action as a reward signal. We store the trajectories in a

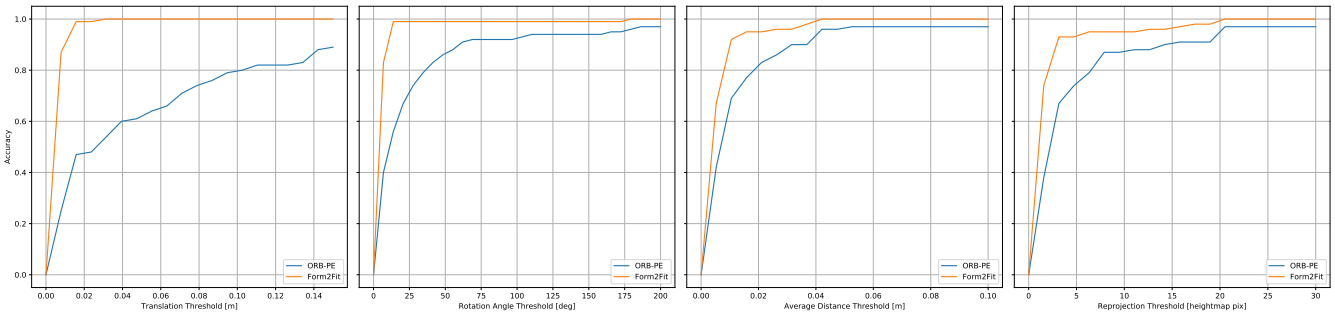

Fig. 2: Our method (Form2Fit) outperforms the baseline alternative.

database and train with prioritized experience replay [4] using stochastic rank-based prioritization, approximated with a power-law distribution.

Matching and placing network training. Once all the data has been gathered, we train the remaining matching and place networks from scratch (i.e., random initialization) for 500 and 200 epochs respectively. All networks are trained with Adam [5] using a fixed learning rate of 10^{-4}, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and a weight decay of 3×10^{-6}. The place network is trained with a batch size of 8 and the matching network with a batch size of 1 (*i.e.*, equivalent to 21).

Data augmentation. We sample a rotation angle uniformly at random from $[0°, 360°]$, then determine the maximum bounds on translation we can apply without transforming the objects or kit outside the heightmap. Once the bounds are computed, we uniformly sample x and y axis translations and construct the corresponding random affine transform. All three networks employ this data augmentation scheme. Note that for the matching network, we only apply random rotations to the kit heightmap and leave the object heightmap untouched.

V. Baseline Details

The ORB-PE baseline method uses ORB descriptors with RANSAC to estimate the ground-truth poses. Specifically, for every image in the training set collected from automatic disassembly, we use OpenCV [6] to detect and compute ORB features which we store in a database. Then, for every query test image in the benchmark, we compute ORB features for the query image and find matches to every descriptor stored in the database using brute force. This allows us to select an image from the training set whose matches are the closest to the query image (where the closeness metric used is the Hamming distance).

Next, we project the matches of the query image and the train image to 3D points in world coordinates, and estimate the rigid transform between both point sets. Finally, we use the train image's ground truth pose in conjunction with the estimated rigid transform to compute the predicted pose $T_{pred} = T_{train} \cdot T_{rigid}$.

We use the Average Distance (ADD) metric to evaluate the performance of ORB-PE and Form2Fit. Given the ground truth rotation R and translation T and the predicted rotation $\hat{R}$ and translation $\hat{T}$, the average distance metric computes the average pairwise distance between the object point cloud

Heightmap $x \in \mathbb{R}^{360\times323\times2}$
Conv3x3(2, 64) - BatchNorm(64)
MaxPool2d(3, 2, 1)
ResBlockDown(64, 128)
MaxPool2d(3, 2, 1)
ResBlockDown(128, 256)
ResBlockDown(256, 512)

(a) Encoder

ResBlockUp(512, 256)
ResBlockUp(256, 128)
Interpolate(2, "bilinear")
ResBlockUp(128, 64)
Interpolate(2, "bilinear")
Conv1x1(64, d)

(b) Decoder

TABLE I: The fully-convolutional encoder-decoder architecture for all three modules. ReLU nonlinearity is omitted for the sake of brevity.

transformed according to the ground truth pose and the estimated one. Concretely,

$$ADD = \frac{1}{m} \sum_{x \in M} ||(Rx + T) - (\hat{R}x + \hat{T})||$$

By comparing the computed ADD to various thresholds, we can plot an accuracy vs threshold curve, visualized in Fig. 2. We observe similar performance in the very low distance tolerance regime (0-0.5 cm) beyond which Form2Fit outperforms the baseline.

VI. Planner Details

The planner integrates information from all three networks to produce the final pose estimate. It does so by considering a large number of suction and place candidate pairs across all kit rotations. Because the Cartesian product of candidate pairs grows exponentially with the number of candidates, a tradeoff between planner speed and accuracy can be controlled through the sampling hyperparameter. In

our experiments, this translates to an execution time that can range from just a few milliseconds to roughly 20 seconds at extremely dense sampling.

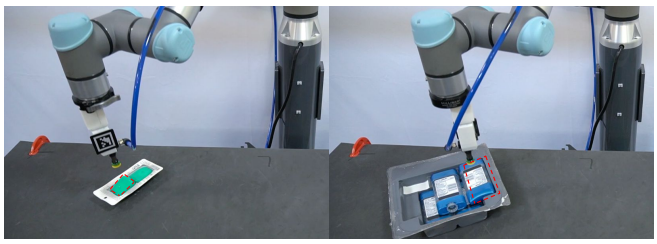

Fig. 3: Example rotational flips that fit in the kit but are counted as unsuccessful assemblies.

VII. Hyperparameters

We performed various hyperparameter sweeps:

- We swept the descriptor dimension d through $[3,5,8,32,64,128]$ and selected 64. We find that for values below 64, the network is over-constrained and learning is crippled. For higher values, while the performance is similar, memory usage is higher and training convergence is slower.
- We swept the contrastive loss margin parameter M through $[1,2,5,8,10]$ and selected 8. Lower values do not produce descriptors that are distinct enough for assembly.
- We swept the batch size for the matching network through $[1,16,32,64]$. We find that increasing the batch size doesn't have a very noticeable effect on final performance.

VIII. Negative Results

We explored various techniques which we ended up discarding due to performance degradation or lack thereof. We report them to give a more complete picture of our attempts.

- We tried to train the matching network with a triplet loss but found that it produced significantly worse matching network performance.
- We found that increasing the spatial resolution of the heightmaps significantly helped performance. We changed it from an initial value of 0.003 to 0.002, *i.e.*, each pixel ended up representing 0.002 meters.
- We tried background randomization and blacking out random objects in the heightmap when training the matching network. We found that it had a negligible effect on performance.
- We tried data augmentation (*i.e.*, random affine transformation) on the object heightmap but found that it degraded performance. Augmenting the kit heightmap did however help.
- We tried adding skip connections from the encoder to the decoder but found that it had little to no effect on performance.
- We tried a SIFT-based baseline instead of ORB but it performed significantly worse. We think this is due to the fact that we use grayscale heightmaps and not RGB images.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[2] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[3] F. Yu, V. Koltun, and T. Funkhouser, "Dilated residual networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 472–480.

[4] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *International Conference on Learning Representations*, vol. 2016, 2016.

[5] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[6] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.