Unsupervised Learning of 3D Landmarks via Reconstruction

Andrew McMullin and Mark Van der Merwe

I. Introduction

Interacting and understanding the 3D world relies upon robust 3D geometric reasoning. An open question in robotics and computer vision research is discovering the correct representation to solve 3D tasks, such as robotic manipulation or visual question answering. Classical methods often rely upon a fully resolved environment, where all the geometries are known and provided (e.g. as meshes or pointclouds). Meshes or pointclouds, however, are both high dimensional and irregular, with different numbers of points/vertices depending on the object. Voxel-based methods provide regularity but are high-dimensional, making learning difficult. With the introduction of deep learning into robotics, latent representations learned from raw sensory inputs have become popular, and make the representation significantly lower-dimensional, however, the latent representation is often difficult to reason over.

We instead propose to use *keypoints* as the representation of 3D geometry and propose a method to learn these keypoints in an unsupervised fashion from 3D object reconstruction. The advantage to this latent representation is that it has an explicit meaning: as a set of points in 3D space. As such, we can reason about the space of keypoints and manipulate it directly. In fact, we recognize that *any 3D transformation* $\phi(\cdot)$ *(e.g., rotation and scaling) can be applied in keypoint space as well.* This provides us with a new way of interacting with the latent space: 3D transformations. We propose a method to train the keypoints to be equivariant to identical 3D transformations applied to the high-dimensional space we wish to represent. This equivariance is highlighted in Fig 1. By training our keypoints in this way, we can directly manipulate the keypoint representation to represent different outcomes in the high-dimensional space.

In Section II, we cover related work in representation learning, especially for robotics. In Section III, we present a differentiable 3D keypoint encoder (extended from a similar 2D keypoint method from [2]) and introduce losses to train our keypoints to both represent the geometry and to enforce the desired equivariance to 3D transformations. We then demonstrate in Section IV that these keypoints can enable direct manipulation of the latent space, outperforming existing methods on a latent space interpolation test, where we wish to rotate an object 90 degrees via the latent space.

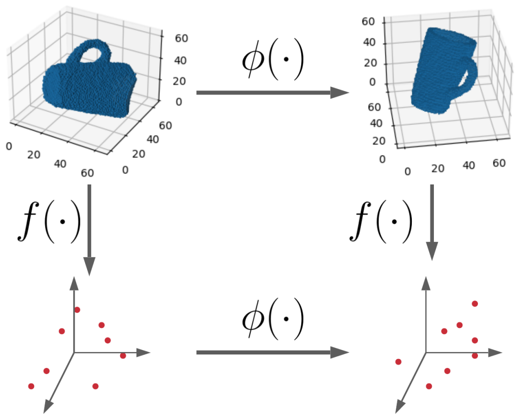
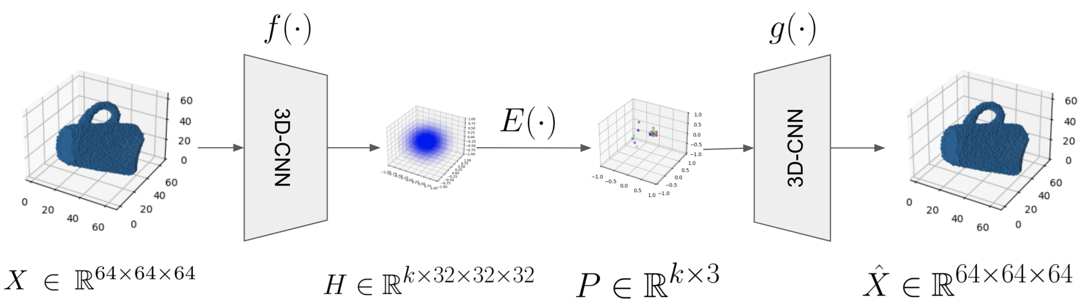


Fig. 1: Keypoints are a compelling latent space due to the fact that their explicit interpretation allows us to apply the same 3D transformations that we apply to our original high dimensional representation. Here $f(\cdot)$ is the mapping to the keypoint latent space, and $\phi(\cdot)$ is a 3D transformation.

II. Related Work

Many representations have been proposed for interacting with the real world. Some works propose to learn directly from high-dimensional sensor inputs using deep learning [7], but need many examples to learn how to act. Lowering the dimensionality of the representation can make it easier to learn how to act. Recent methods seek to first lower the dimensionality of the representation by pretraining a useful representation via a neural network's latent space, then using the lower dimensional latent space as the representation of the world. Previous methods have used 3D voxel reconstruction [8], contrastive learning [11], [12], and multi-modal self-supervision [6] as the pretraining step to learn this lower dimensional representation. However, latent representations are difficult to reason over directly, as they often lack explicit meaning and are unstructured. Recently, methods working in 2D have introduced keypoints as a low-dimensional, structured, and easy to reason over alternative to learned latent representations. To learn these keypoints, methods have been proposed using multi-view consistency [5] and image reconstruction [2]. We propose to extend this notion to 3D, and train the keypoint representation using 3D reconstruction as the supervision.

As mentioned above, reconstruction has been used as a form of supervision for representation learning often, since it requires no additional supervision beyond the data already present. Variants of 3D reconstruction include the output representation, including voxels [3], point clouds [4], and implicit surfaces [9]. While our method should work for any resulting output representation, we use voxels for simplicity.



$f(\cdot)$
3D-CNN
$E(\cdot)$
$g(\cdot)$
3D-CNN
$X \in \mathbb{R}^{64\times64\times64}$
$H \in \mathbb{R}^{k\times32\times32\times32}$
$P \in \mathbb{R}^{k\times3}$
$\hat{X} \in \mathbb{R}^{64\times64\times64}$

Fig. 2: An overview of our differentiable keypoint architecture. We show the input voxels, a single keypoint heatmap (note in reality k heatmaps are predicted), the resulting keypoints wwhen taking the expectation over the heatmaps, the predicted voxels, and the mappings between.

III. Methodology

A. Differentiable 3D Keypoint Encoder

We propose to learn a structured 3D keypoint representation of objects using a differentiable keypoint bottleneck within a 3D reconstruction pipeline. In particular, we propose to extend 2D keypoint bottlenecks to full 3D situations [2].

Our method takes in a voxel representation of the object, $X \in \mathbb{R}^{64\times64\times64}$, where each voxel takes value 1.0 when occupied and 0.0 when unoccupied. Our goal is to predict k 3D keypoints $P = \{P_1, P_2, ...P_k\}$, $P_i \in \mathbb{R}^3$ in a differentiable manner. To do this, we pass our voxel representation through a 3D Convolutional Neural Network (CNN). We output our keypoints as a set of k heatmaps. Each map $H_i \in \mathbb{R}^{32\times32\times32}$ is interpreted as the probability of the ith keypoint appearing at that location in space. To recover the 3D location, we take the expectation over the 3D coordinates, weighted by the given heatmap:

$$\mathbb{E}[P_i^x] = \frac{1}{S}\sum_{x,y,z} x \cdot H_i(x,y,z) \tag{1}$$

$$\mathbb{E}[P_i^y] = \frac{1}{S}\sum_{x,y,z} y \cdot H_i(x,y,z) \tag{2}$$

$$\mathbb{E}[P_i^z] = \frac{1}{S}\sum_{x,y,z} z \cdot H_i(x,y,z) \tag{3}$$

where S is the number of voxels in total.

These k keypoints P then become our latent representation of the input. We can then pass this representation to another 3D CNN which attempts to reconstruct the input voxels. In particular, the keypoints are flattened and passed through a dense layer before being reshaped into a 3D grid and being passed through subsequent 3D CNN layers. We finally get out the voxel reconstruction, $\hat{X} \in \mathbb{R}^{64\times64\times64}$. The architecture is shown in Fig. 2.

For future discussion, we call our differentiable 3D keypoint encoder $f : \mathbb{R}^{64\times64\times64} \rightarrow \mathbb{R}^{k\times3}$ which maps from voxels to keypoints. The decoder which takes in our keypoints and maps back to the voxel space we call $g : \mathbb{R}^{k\times3} \rightarrow \mathbb{R}^{64\times64\times64}$. We modify the encoders and decoder from [10] to our case, changing to the new heatmap outputs in the latent space. We note that our method is not specific to voxels as an input or output representation, and could be extended to various representations in the future.

B. Training the 3D Keypoint Representation

Here we present our training losses and show how we can expect to learn a set of keypoints that respect our desired equivariance to 3D transformations. We use a combination of three losses to train our method: a reconstruction auto-encoding loss, an equivariance consistency loss, and a keypoint separation loss.

Our first form of supervision is to encourage the keypoints to encode the geometry of the object. We do this by encouraging the output from our decoder to look like the input voxels. This is a standard auto-encoding reconstruction loss. We can train this using average Binary Cross Entropy (BCE) loss between the predicted and input voxel geometries.

$$\mathcal{L}_{\text{recon}} = BCE(X, \hat{X}) = BCE(X, g(f(X)))) \tag{4}$$

Next, we must constrain our keypoint representation to respect the desired equivariance to 3D transformations. Without additional supervision, the points may not necessarily respect the transformations $\phi(\cdot)$. To train this, we augment our dataset by applying a random transformation, in the form of anistropic scaling and a random rotation, $\phi(X, \theta)$ where X is the input voxels and θ parameterizes the transformation (i.e., angles of rotation and scaling factors). We apply two separate augmentations, parameterized by θ_1, θ_2, then encode each to keypoints. We then *invert* the transformations, using the provided θ values and apply a loss encouraging the resulting normalized keypoints to be the same.

$$\mathcal{L}_{\text{keypoint}} = MSE(\phi^{-1}(f(\phi(X,\theta_1)),\theta_1), \phi^{-1}(f(\phi(X,\theta_2)),\theta_2) \tag{5}$$

We note that $\phi^{-1}(\cdot)$ for rotation and anistropic scaling are trivial to implement. We show this loss visually in Fig. 3. This loss ensures that our keypoint latent space respects the transformations, and allows us to directly manipulate the latent space using 3D transformations and yield corresponding results in the high-dimensional space. We note that this loss is similar to an augmentation based loss used in [2] used to

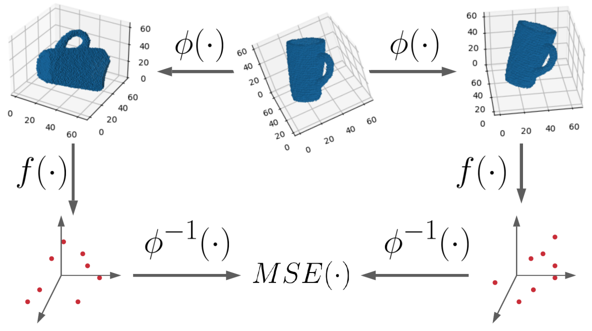


Fig. 3: Visualizing our Keypoint Consistency loss.

make their 2D keypoints represent consistent points in the scene. Ours can be viewed as a more general, 3D version, used to structure our latent space.

Finally, we have a simple loss to prevent the keypoints from collapsing and encourage them to spread out. This is identical to a separation loss used in [2] for the same purpose.

$$\mathcal{L}_{\text{separation}} = \frac{1}{K^2} \sum_{i=1}^{k} \sum_{j=1}^{k} \frac{1}{1 + M * ||P_i - P_j||_2^2} \tag{6}$$

Where M is chosen to be 1000.

The final loss we train our method with is a weighted combination of these three losses.

$$\mathcal{L} = \mathcal{L}_{\text{recon}} + \alpha \cdot \mathcal{L}_{\text{keypoint}} + \beta \cdot \mathcal{L}_{\text{separation}} \tag{7}$$

In our experiments, we use $\alpha = 1.0$ and $\beta = 0.1$.

C. Dataset

As this is an early investigation into equivariant keypoints, we consider only a single mug undergoing various 3D transformations. We use a single mug from the ShapeNet [1] dataset. As discussed in Section III-B, our training requires random rotations and anisotropic scaling. We first randomly sample three scaling terms for each dimension $s_x, s_y, s_z \in [0.75, 1.0]$ and randomly sample an angle to rotate around the z-axis of the mug $\psi \in [0, 2\pi]$. We then apply the scaling and rotation to the mug point cloud and voxelize the result to get our inputs X. We apply this twice randomly to get a pair for training with our consistency loss (Eq. 5). The mug mesh and example voxelized inputs are shown in Fig. 4. Each training epoch consisted of 4000 such pairs of randomly generated voxelizations.

D. Training Details

We implemented our differentiable keypoint method in Pytorch. Our code can be found at https://github.com/mvandermerwe/object_keypoints. We train with the Adam optimizer and train the method for 400 epochs, choosing the best method based on validation error on 4000 randomly generated validation examples from the dataset. We use a batch size of 8 and set the number of keypoints $k = 32$.

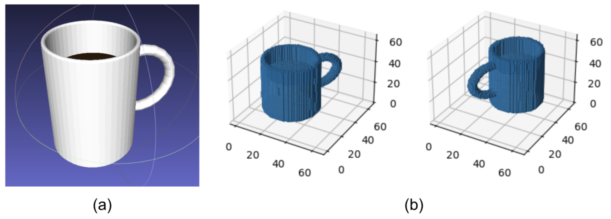


Fig. 4: (a) The mug we used in our experiments and (b) examples of the random augmentations used to train the network.

IV. Experiments

We seek to answer two questions through our experiments. First, what is the representation power of our keypoints? That is, how well can our keypoint bottleneck reconstruct a given geometry. Second, can we directly use the equivariance of the latent space to perform latent interpolation to rotate a mug in the high-dimensional space by adjusting the latent space.

A. Baseline

For our baseline, we use the same network architecture presented in Sec. III, but replace the keypoint encoding with a standard unstructured latent space. That is, the encoder $f(\cdot)$ simply outputs a latent vector $z \in \mathbb{R}^l$ where l is the length of the latent code. This latent vector can then be passed to the decoder. The method is trained with only the reconstruction auto-encoding loss (Eq. 4). We call this method CNN-Net and call our method KP-Net.

For our experiments we set $l = 128$. We note that this means the baseline is given slightly more bits for representing the object than our keypoint method, which uses $3 * 32 = 96$ floating point numbers. We train the method using the same dataset and number of epochs as our method.

B. Metrics

To compare our reconstructions for the representation power and latent interpolation tests, we use three common metrics for reconstruction. First, we use the reconstruction loss used for training (i.e., Eq. 4), which we refer to hereafter as BCE. For the next two metrics, we first threshold the reconstruction $\hat{X}$ to get a binary occupancy prediction at each voxel, using the threshold 0.5 for all methods. The second metric we use is the voxel Intersection over Union (IoU), which sums the intersection of the occupied voxels by the union of the occupied voxels given the predicted and ground truth voxels. Thus, a perfect reconstruction would have IoU of 1.0. Finally, we use the chamfer distance, a common pointcloud distance metric. To do this, we convert our ground truth and predicted voxels to point clouds, sample 1000 points randomly from the resulting cloud, and measure the pointcloud distance. For more details on these common metrics, we refer the reader to [9].

Method	BCE ↓	IoU ↑	Chamfer ↓
CNN-Net	0.006 (0.001)	0.984 (0.003)	9.244 (0.953)
KP-Net (Ours)	0.012 (0.0144)	0.970 (0.011)	9.366 (0.929)

TABLE I: Representation Power Performance

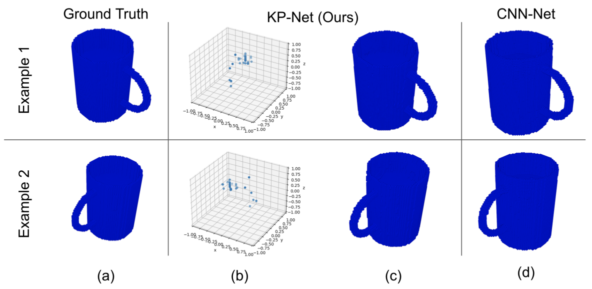


Fig. 5: Qualitative Reconstruction Examples: a) Ground truth voxels, b) keypoint representation, c) KP-Net reconstruction, d) CNN-Net Reconstruction.

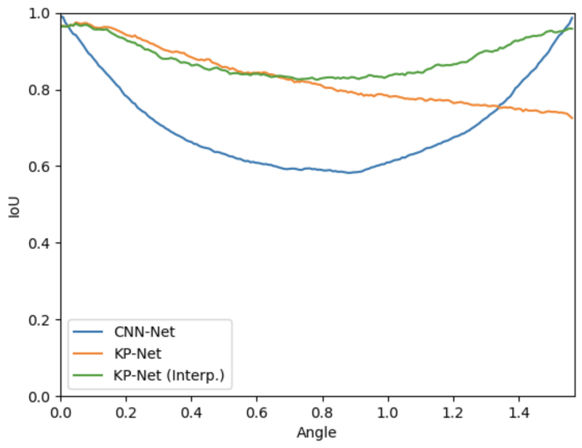


Fig. 6: IoU performance of the various latent interpolation method on the task of interpolating between two rotated mug meshes. We see that our proposed methods outperform the unstructured latent baseline on intermediate states of the mug.

C. Representation Power

To determine the keypoint representation power, we sample 500 random augmentations from our mug dataset and reconstruct each. We compare the resulting reconstructions from our method and from the baseline unstructured network method using the three metrics introduced above. The mean and standard deviation across the dataset for the two methods is shown in Table I. The arrows next to each metric name remind whether we wish to maximize or minimize each metric. Qualitative examples of reconstruction are shown in Fig. 5.

We see that quantitatively, the baseline method CNN-Net method slightly outperforms our method. However, from our qualitative examples we see that the change is visually very small. This is to be expected, as our method has a more complex loss function to optimize. We see that learning with our added losses does not significantly decrease representation power.

D. Latent Interpolation

Here we wish to rotate the mug by 90 degrees, passing through earlier rotation states, by only interacting with the latent space. This sort of task could come in useful when doing robotic manipulation planning or goal specification, where we wish to get intermediate goal states or specify a goal. Success in this task would reconstruct the mug at various intermediate rotations smoothly rotating to the goal configuration - we use our IoU metric as a quantitative measure of success for given intermediate rotations.

To perform the interpolation, our method can exploit the learned equivariance. For a given desired rotation around the z-axis ψ, we can create a 3D rotation transformation $R_z(\psi) \in SO(3)$ and directly apply it to our keypoint space. Thus the latent space for a given ψ would be $R_z(\psi)f(X)$ where X is the mug voxels in its base configuration. We can then reconstruct the resulting keypoints to get an estimate of the voxels at the intermediate rotation and compare to the ground truth. We refer to this approach as KP-Net.

$$\hat{X}_\psi = g(R_z(\psi)f(X)) \tag{8}$$

For our baseline method CNN-Net, the unstructured latent space means we cannot perform this direct reasoning. The closest we can do is assume access to the goal voxel state X^* (e.g., after the rotation), then embed both the starting and ending configuration and linearly interpolate in the unstructured latent space. We weight between the goal and start by using the fraction along the 90 degree rotation we wish to construct. We refer to this approach as CNN-Net.

$$\hat{X}_\psi = g((1.0 - \frac{\psi}{90}) \cdot f(X) + (\frac{\psi}{90}) \cdot f(X^*)) \tag{9}$$

We note that this gives more information to the baseline, since it also has access to the goal configuration and can use it to decide the latent state. We can extend this interpolation to our method as well, by encoding the goal configuration as well, rotating back to the desired angle and performing a weighted sum.

$$\hat{X}_\psi = g((1.0 - \frac{\psi}{90}) \cdot R_z(\psi)f(X) + (\frac{\psi}{90}) \cdot R_z(-90+\psi)f(X^*)) \tag{10}$$

We refer to this method as KP-Net (Interp.).

We show the quantitative results in Fig. 6. We see that our methods significantly outperforms the baselines in intermediate locations, even when not having access to the final configuration information. This indicates that our method has learned to respect 3D transformations in the latent space and so can use it to provide accurate intermediate reconstructions. When we additionally provide our method with the final configuration method (KP-Net (Interp.)), we also improve as we get closer to the final configuration. Representative

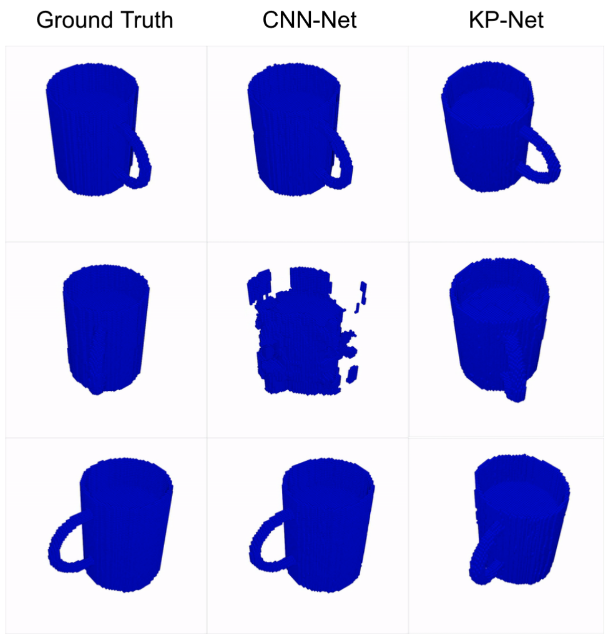


Fig. 7: Qualitative interpolation for our method (KP-Net) and the baseline (CNN-Net).

interpolation reconstructions are shown in Fig. 7. We see that the baseline method does not yield cogent intermediate reconstructions, but rather the object appears to deconstruct and reconstruct at the new configuration. Our method, on the other hand, smoothly rotates as expected.

V. Conclusion

In this work we investigate using a structured keypoint representation to represent high dimensional 3D objects succinctly. We showed that we could train this representation to be equivariant to 3D transformations, allowing a novel way of manipulating the object via the latent space. Our experiments showed that our proposed method has similar representation power to existing unstructured latent models. Additionally, the explicit latent structure allowed our method to outperform unstructured latent methods in executing a desired latent interpolation corresponding to the rotation of our 3D object.

While promising, our results need to be extended in two ways before their integration into robotic manipulation can be realized. First, the method should be extended to handle not just a single object but a category of objects. Second, the method should be used given only partial inputs (e.g., such as the point cloud one receives from a depth sensor). In theory, our presented training paradigm can be used directly on a dataset with these two properties, but it remains to be seen whether the equivariance properties hold meaningful in this new setup.

References

[1] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[2] Boyuan Chen, Pieter Abbeel, and Deepak Pathak. Unsupervised learning of visual 3d keypoints for control. *arXiv preprint arXiv:2106.07643*, 2021.

[3] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European conference on computer vision*, pages 628–644. Springer, 2016.

[4] Haoqiang Fan, Hao Su, and Leonidas J. Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[5] Peter R Florence, Lucas Manuelli, and Russ Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. *arXiv preprint arXiv:1806.08756*, 2018.

[6] Michelle A Lee, Yuke Zhu, Krishnan Srinivasan, Parth Shah, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Jeannette Bohg. Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8943–8950. IEEE, 2019.

[7] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[8] Qingkai Lu, Mark Van der Merwe, Balakumar Sundaralingam, and Tucker Hermans. Multifingered grasp planning via inference in deep neural networks: Outperforming sampling by learning differentiable models. *IEEE Robotics & Automation Magazine*, 27(2):55–65, 2020.

[9] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[10] B Saund and D Berenson. Diverse plausible shape completions from ambiguous depth images. In *Conference on Robot Learning (CoRL)*, 2020.

[11] Wilson Yan, Ashwin Vangipuram, Pieter Abbeel, and Lerrel Pinto. Learning predictive representations for deformable objects using contrastive estimation. *arXiv preprint arXiv:2003.05436*, 2020.

[12] Albert Zhan, Philip Zhao, Lerrel Pinto, Pieter Abbeel, and Michael Laskin. A framework for efficient robotic manipulation. *arXiv preprint arXiv:2012.07975*, 2020.