# 3D Deformable Object Manipulation using Deep Neural Networks

Zhe Hu[1], Tao Han[1], Peigen Sun[2], Jia Pan[2†], and Dinesh Manocha[3]

***Abstract*—Due to its high dimentionality, deformable object manipulation is a challenging problem in robotics. In this paper, we present a deep neural network based controller to servo-control the position and shape of deformable objects with unknown deformation properties. In particular, a multi-layer neural network is used to map between the robotic end-effector's movement and the object's deformation measurement using an online learning strategy. In addition, we introduce a novel feature to describe deformable objects' deformation used in visual-servoing. This feature is directly extracted from the 3D point cloud rather from the 2D image as in previous work. In addition, we perform simultaneous tracking and reconstruction for the deformable object to resolve the partial observation problem during the deformable object manipulation. We validate the performance of our algorithm and controller on a set of deformable object manipulation tasks and demonstrate that our method can achieve effective and accurate servo-control for general deformable objects with a wide variety of goal settings. Experiment videos are available at https://sites.google.com/view/mso-deep.**

***Index Terms*—Deep Learning in Robotics and Automation; Visual Servoing; Dual Arm Manipulation; RGB-D Perception; Model Learning for Control**

## I. Introduction

COMPARED to manipulating rigid objects, deformable object manipulation is a more challenging task in robotics since it has an extremely high configuration space dimensionality (only 6 dimensions in rigid object manipulation). Though challenging, deformable object manipulation has broad applications in our life [1]–[7].

Our previous work [8] used Gaussian process (GP) based online learning to make manipulation policy adapt to the changing deformation parameters. It has two major limitations. First, GP has limited representation power and thus may not well describe the deformation behavior of soft objects. Second, the learned controller may fail when the soft object is occluded by other obstacles or has self occlusions. Thus, we want to address these limitations to make the deformable object manipulation algorithm more robust and converge faster.

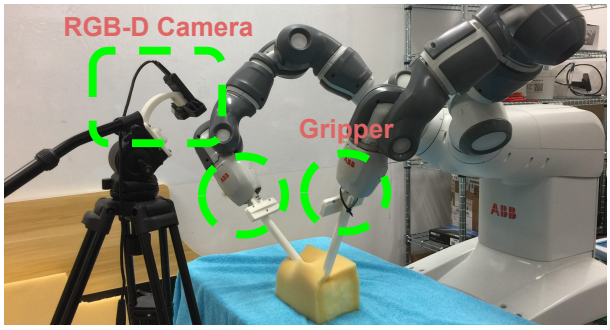



Fig. 1: The robotic and vision system used in our experiment.

**Main Results:** In this paper, we present a novel deformable object manipulation controller which can solve the two limitations in [8] mentioned above. The contribution of this paper is threefold:

- We encode the state of the deformable objects using a novel fixed-length feature that is based on the Fast Point Feature Histogram (FPFH) but extended with PCA. According to our knowledge, this is the first time that a similar feature is used for object manipulation.
- We present a novel data-driven controller based on Deep Neural Networks (DNNs) which can accomplish a better performance than previous works that used linear [5], [9] or nonlinear controllers [8], thanks to the strong representation power of neural networks.
- We design a robust occlusion recovery algorithm which obtains a complete point cloud from the occluded raw data using the real-time tracking and reconstruction technique and thus improves the controller's robustness when meeting occlusions.

## II. Related Work

Existing techniques about deformable object manipulation can mainly be categorized into two types, the traditional methods and the learning-based approaches. Traditional solutions need a (usually over-simplified) deformation model of the target object for deriving an appropriate control policy for manipulation. For instance, [10] characterized the deformation pattern of shell-like objects using an extension of the shell theory. [11] designed an energy function to formulate the bending behavior of planar objects during the grasping operation. The general finite element method (FEM) framework has also been used to systematically describe the deformation behavior of a 3D deformable object when being picked up by a robotic gripper [12].

However, in many scenarios, the deformation model is difficult to obtain due to the complexity of objects. Even if

† denotes the corresponding author.

This work was partially supported by HKSAR Research Grants Council (RGC) General Research Fund (GRF), HKU 17204115, 21203216, NSFC/RGC Joint Research Scheme HKU103/16, and Innovation and Technology Fund (ITF) ITS/457/17FP.

[1]Zhe Hu and Tao han are with the Department of Biomedical Engineering, the City University of Hong Kong, Hong Kong

[2]Peigen Sun and Jia Pan are with the Department of Computer Science, the University of Hong Kong, Hong Kong jpan@cs.hku.hk

[3]Dinesh Manocha is with the Department of Computer Science, the University of Maryland, College Park, MD 20742, USA

Digital Object Identifier (DOI): see top of this page.

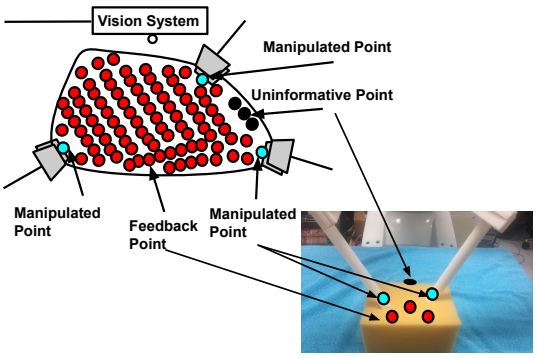



Fig. 2: We model a soft object using three classes of points: manipulated points, feedback points, and uninformative points.

we are able to model the deformation, tuning the deformation model's internal parameters is also difficult. As a result, many recent learning-based methods are proposed to obtain the deformation model or manipulation strategies directly from data. For instance, [13] used reinforcement learning to optimize a controller for haptics-based manipulation, where a high-quality simulator enables the robot to investigate different policies efficiently and thus is critical for the convergence of the learning process. [14] formulated the deformable object manipulation as a model selection problem and introduced a utility metric to measure the performance of a specific model according to the simulation results. These methods focus on deriving a flexible high-level policy but usually lack the capability to achieve accurate operation, which is important for real-world applications. One promising solution to accurate deformation object manipulation is based on the visual servoing. [5], [9], [15] used an adaptive and model-free linear controller to servo-control soft objects, where the object's deformation is described using a spring model [16]. [8] presented a nonlinear servo-controller whose parameters are adaptively adjusted during the manipulation process using the Gaussian process based online learning. In this paper, we follow the general framework of visual-servoing but combine it with deep-learning to accomplish a nonlinear controller that is accurate and robust.

## III. Overview and Problem Formulation

Similar to [8], we discribe the deformable object as a set of discrete points, including the feedback points $\mathbf{p}^f$, the manipulated points $\mathbf{p}^m$ and the uninformative points $\mathbf{p}^u$, as shown in Figure 2. Also, we model the relation between feedback points and manipulated points as

$$\delta\mathbf{p}^m = F(\delta\mathbf{p}^f), \tag{1}$$

where $\delta\mathbf{p}^f = \mathbf{p}^f - \mathbf{p}^f_*$ and $\delta\mathbf{p}^m = \mathbf{p}^m - \mathbf{p}^m_*$ are the displacement relative to the equilibrium for feedback points and manipulated points, respectively.

However, to get $\delta\mathbf{p}^f$, we have to perform tacking during manipulation so as to get the correspondence of feedback points between frames, which is unreliable when the number of points is large. Thus, similar to [8], we extract a low-dimension feature vector $\mathbf{x}$ based on feedback points $\mathbf{p}^f$ and thus we have $\mathbf{x} = Q(\mathbf{p}^f)$ , where $Q(\cdot)$ is the feature extraction function. We expand the function $Q(\cdot)$ at equilibrium state $\mathbf{p}^f_*$ and get $\delta\mathbf{x} = Q'(\mathbf{p}^f_*)\delta\mathbf{p}^f_*$. Thus we can rewrite the Equation 1 as

$$\delta\mathbf{p}^m = F(Q'(\mathbf{p}^f_*)^{-1}\delta\mathbf{x}) \triangleq Z(\delta\mathbf{x}), \tag{2}$$

where the function $Z(\cdot)$ is called the deformation function.

Finally, the goal of the deformable object manipulation is to find a controller which can learn the deformation function $Z(\cdot)$ and use this function to compute desired control velocity through $\delta\mathbf{p}^m = Z(\eta \cdot \Delta\mathbf{x})$, where $\Delta\mathbf{x} = \mathbf{x}_d - \mathbf{x}$ is the gap between the desired state $\mathbf{x}_d$ and the current state $x$, and $\eta$ is the feedback gain.

Note that the target states are not involved in the learning process and they are just used to compute the required control. Also, we assume the given target states are always accessible within some tolerance. This assumption is reasonable since the goal of our servoing algorithm presented here is to accomplish a high accurate manipulation, which needs a relatively correct target configuration in real applications.

## IV. Feature Extraction

In order to describe the deformable object' state, we compute a feature vector based on the original 3D point cloud ($\mathbf{x} = C(\mathbf{p}^f)$). We first extract a 135-dimension histogram based on point cloud and then use PCA (Principle Component Analysis) to reduce its dimension to 30.

### A. Extended FPFH

The extended FPFH [17] is a histogram feature extended based on FPFH [18] and PFH [19]. The Point Feature Histograms are informative pose-invariant local features which represents the surface model's property at point $p$. The histograms are computed based on the combination of certain geometrical relations (like pan, tilt and yaw angles) between $p$'s nearest $k$ neighbors. In detail, first, for each query point $p$, we only consider $p$'s neighbors enclosed in the sphere with radius $r$. Second, for every pair of points $p_i$ and $p_j$ $(i \neq j)$ in point $p$'s $k$-neighbors, we define a Darboux $uvw$ frame $(u = n_i, v = (p_j - p_i) \times u, w = u \times v)$ and compute three feature angles:

$$\begin{aligned} \alpha &= v \cdot n_j, \\ \phi &= (u \cdot (p_j - p_i))/\|p_j - p_i\|, \\ \theta &= \arctan(w \cdot n_j, u \cdot n_j), \end{aligned} \tag{3}$$

where $n_i$ and $n_j$ represent the estimated normals at point $p_i$ and $p_j$ respectively. Third, we divide these angles' space into several bins and create a histogram for the query point $p$. However, the computational complexity of this histogram is $O(n \cdot k^2)$, where $k$ is the number of neighbors for each query point $p$. In order to reduce the computation time, [18] presented a fast version of PFH called FPFH (Fast Point Feature Histogram) that is computed in two steps. First, for each query point $p$ we compute the angles (in Equation 3) between itself and its neighbors and call the feature Simplified Point Feature Histogram (SPFH). Next, the final FPFH is

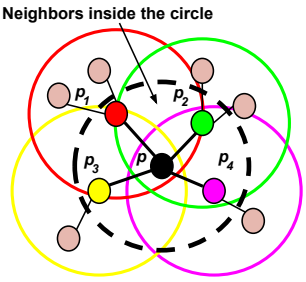



Fig. 3: FPFH feature is computed using the SPFH features of the query point $p$ and its neighbors $p_k$.

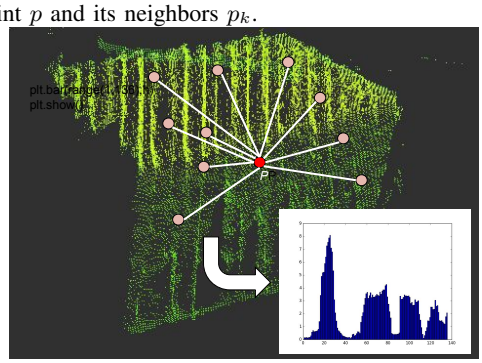


Fig. 4: The extended FPFH is obtained by computing the relation angle between the centroid point and all other points. The red point in Figure 4 represents the centroid point of the whole point cloud.

computed as a weighted sum of the SPFHs of $p$ and its neighbors $p_k$:

$$FPFH(p) = SPFH(p) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{w_k}\cdot SPFH(p_k), \quad (4)$$

where the weight $w_k$ is the distance between query point $p$ and its neighbor point $p_k$. A sketch diagram illustrating these computation processes is shown in Figure 3.

However, FPFH is a histogram at a certain point rather than the whole point cloud. Thus in order to describe the entire deformable object, we finally choose to use an extended version of FPFH which computes the relation angle between the centroid point and all other points (see Equation 3), which is shown in Figure 4. This final histogram contains 135 dimensions (45 for each angle), and it describes the whole surface shape of the deformable object.

### B. Extended FPFH with PCA

Due to the high dimension of extended FPFH, we need more data to fit our controller model. In addition, from experiments, we find that the values of some dimensions change slightly. Thus, we use Principal Component Analysis (PCA) to project the raw extended FPFH to a new space with higher variance and lower dimensions. In particular, for the raw extended FPFH $h$, we first compute the covariance matrix $\Sigma = \frac{1}{n}\sum_i (h_i - \mu)(h_i - \mu)^T$, where $h_i$ represents the $i$-th extended FPFH data in a given data set and $\mu = \frac{1}{n}\sum_i h_i$. Next, we perform the eigen decomposition of the covariance matrix, which means that we find the eigen value $\lambda_i$ and eigen vector $v_i$ that satisfy $\Sigma v_i = \lambda_i v_i$. After we obtain the eigen value $\lambda_i$ and the eigen vector $v_i$, we sort these eigen values and choose $K$ eigen vectors with top $K$ eigen values $\Phi = [v_1, v_2, \cdots, v_K]$. The final feature vector is then

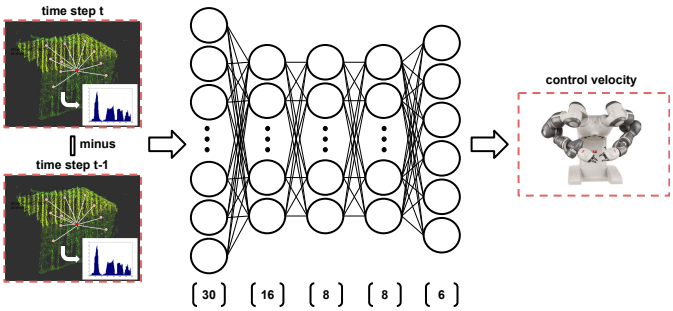



Fig. 5: The architecture of our 5-layered network $H$. The number inside the brackets indicates the neural unit number in each layer.

obtained by projecting raw feature into new space using $\Phi$: $x = \Phi^T(h - \mu)$. The parameter $K$ in our experiment is fixed to 30.

## V. Controller Design

Previous work [5], [8], [9] used a linear model or a GP-based model to learn the deformation function $H$ gradually. However, these simple models have limited representation power and thus may not be able to describe $H$ accurately. Here, we a Deep Neural Network (DNN) as the approximation model to $H$, which can result in a controller that is more accurate and robust due to the strong representation power of DNNs. Actually, both linear model and GP are equivalent to a single layer neural network (possibly with infinite width for GP). Thus, our work extends previous works lengthways.

### A. Network Architecture

In order to learn the deformation function, we build a 5-layered neural network, which is shown in Figure 5. The first layer of the network is an input layer which includes 30 neural units. The network input is feature velocity which is the velocity of extended FPFH after PCA. In training time, this velocity is obtained by subtraction between the FPFH of the current and last time step. In test time, this velocity is obtained by subtraction between the FPFH of the target and current configuration. The second layer is a hidden layer which includes 16 neural units. The third and fourth layers both include 8 neural units. As for the activation function, we test several functions like ReLU and linear. Finally, we choose the linear function as the activation function according to the regression result in the experiment. The last layer is an output layer which includes 6 neural units covering the 6 dimensions of control velocity in the dual-arm robot (ABB Yumi).

To train our neural network, we choose the Mean Square Error (MSE) as loss function. The MSE loss function compute the error between label and estimated value by

$$\text{mse} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2, \quad (5)$$

where $Y_i$ and $\hat{Y}_i$ represent the ground truth and estimated value respectively. The optimizer we choose is called RMSProp which is a method with adaptive learning rate.

We choose to represent $H$ as a small fully-connected multi-layer network rather than a recurrent network (RNN) because

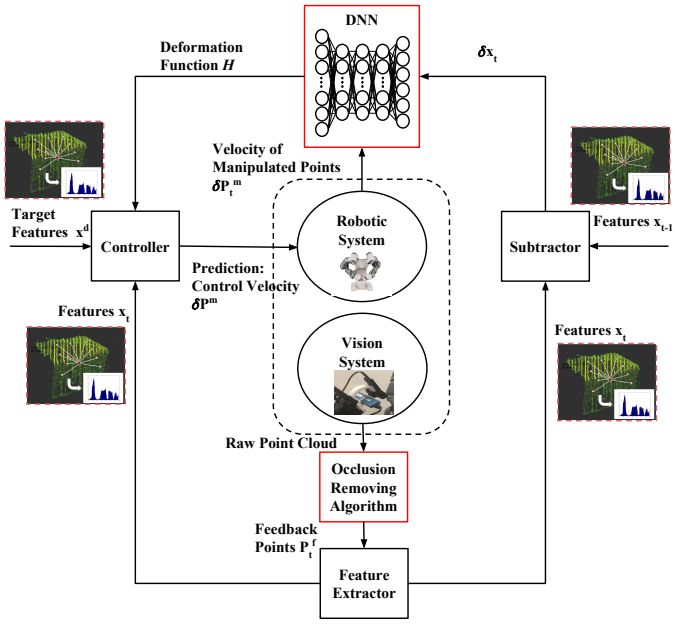



Fig. 6: Our method is an online learning process, which means that the DNN is learned while manipulation.

to avoid overfitting RNN needs more training data which is not feasible during online learning, and RNN has slow training process which also makes it not suitable for the real-time manipulation.

### B. Online Learning Process

In order to improve the model's accuracy and robustness, we follow our previous work [8] which is an online learning method. The online learning process is shown in Figure 6. In detail, we collect data and train the DNN while manipulation. The data is a pair of grippers' and feature's velocities, which can be obtained from the robotic system and vision system respectively. At the same time, the DNN is used to predict the required control velocity given the target and current feature values. The robotic system receives this control velocity and performs manipulation, and then new data are generated.

This online learning process removes the requirement of offline data collection and offline training. Besides, since the data is collected while manipulation, the model will become more and more accurate. In our experiment, we will show that online learning provides a more precise and robust way to perform deformable object manipulation compared to the offline model.

## VI. Occlusion Removing Algorithm

We extract our extended FPFH with PCA from the 3D point cloud provided by an RGB-D camera like the Intel Realsense. However, when some surface parts of the object become invisible due to self-occlusion or occlusion from the robot's gripper, the raw point cloud captured by the camera can not describe the full state of the object. As a result, the feature vector extracted from such point cloud is incomplete for shape servoing.

In this work, we present an occlusion removing algorithm to overcome the problem mentioned above. Our occlusion removing algorithm takes the RGB-D stream as input and serves as the front-end for feature extraction (as shown in Figure 6). One main advantage of the proposed algorithm is that it can generate a complete point cloud to represent all surface state of the object, including the currently observed parts, and the previously observed but currently occluded parts. The point cloud generated by the algorithm will further serve as input for feature extraction, which makes our system more robust to occlusion than previous work [5], [8].

To make the occlusion removing algorithm fulfill our requirement of being model-free, we follow the idea of recent work [20], and formulate the problem into two phases, namely tracking and reconstruction. Based on that, we solve the problem by invoking the two phases in an alternative manner. More precisely, when a new RGB-D image arrives, we activate the tracking phase at first. In this phase, we estimate the deformation field from a reference point cloud to the live point cloud encoded in the depth stream to capture the motion of both the visible and occluded surface parts. We achieve this by firstly performing a non-rigid alignment to match the visible parts of the reference point cloud with the live point cloud. Then we extend the estimated deformation field to the occluded parts according to the as-rigid-as-possible (ARAP) regularization term [21], [22].

Note in the above tracking phase, a reference point cloud which containing the state of both the visible and occluded surface parts is required for non-rigid alignment. However, since we want our pipeline to stay in the budget of being model-free, it is infeasible to introduce any prior surface model for deformation tracking. In our algorithm, we handle this problem in the reconstruction phase based on an incremental image fusion step. To achieve this, we invoke the reconstruction phase after the deformation field is estimated in the tracking phase. According to the estimated deformation field, we integrate the depth image into the reference point cloud to gradually complete and refine its surface details based on the new measurement. After the reference point cloud is updated, we warp it into the configuration of the live point cloud based on the estimated deformation and employ the warped cloud as the reference model for deformation tracking when next new image comes.

### A. Tracking

To model the deformation field in the tracking phase, we follow recent work [20] by representing it as a graph model $\mathcal{G}$. The basic idea of such graph-based representation is to discretize the deformation field into a set of local rigid transformations $\mathcal{G} = \{\mathbf{T}_i \in \mathbf{SE}(3)\}_{i=1}^K$, and assign them to the graph nodes $\{\mathbf{g}_i\}_{i=1}^K$ one-to-one. The graph nodes are uniformly sampled from the reference point cloud to ensure the local transformations' distribution roughly conform to the object's shape. Given the graph model $\mathcal{G}$, we can calculate the deformation of each reference cloud point through interpolation of the local transformations in its nearest graph nodes:

$$\tilde{\mathbf{p}} = \mathcal{W}(\mathbf{p}; \mathcal{G}) = \sum\nolimits_{k \in \mathcal{S}} \omega_k \mathbf{T}_k \mathbf{p}, \tag{6}$$

where $\tilde{\mathbf{p}}$ and $\mathbf{p}$ are the deformed and original reference cloud point respectively; $\mathcal{S}$ denotes the k-nearest neighbor nodes of

the point $\mathbf{p}$; $\omega_k$ is the skinning weight which can be calculated by $\omega_k = \frac{1}{Z}\exp(-\|\mathbf{p}-\mathbf{g}_k\|^2/2\sigma_k^2)$, where $Z$ is a normalization factor, $\sigma_k$ is a predefined parameter.

Based on the graph model $\mathcal{G}$, we formulate the deformation tracking process as the following optimization problem, where we aim at finding the optimal deformation field with smallest energy: $\mathcal{G}^* = \text{argmin}_{\mathcal{G}}\, E(\mathcal{G})$ where

$$E(\mathcal{G}) = \lambda_{\text{data}} \sum_m \|\mathbf{n}_d^\top(\tilde{\mathbf{p}}_m(\mathcal{G}) - \mathbf{p}_d)\|_2^2 + \lambda_{\text{reg}} \sum_{i=1}^{K} \sum_{j\in\mathcal{N}_i} \|\mathbf{T}_i\mathbf{g}_i^t - \mathbf{T}_j\mathbf{g}_i^t\|_2^2. \quad (7)$$

Here $E(\mathcal{G})$ is the energy function. In Equation 7, we use the first term of $E(\mathcal{G})$ to penalize the misalignment between the visible parts of the deformed reference cloud and the live cloud. Here $\tilde{\mathbf{p}}(\mathcal{G}) = \mathcal{W}(\mathbf{p};\mathcal{G})$, and $\{\mathbf{p}_d, \mathbf{n}_d\}$ denote the corresponding point and normal of $\tilde{\mathbf{p}}$ in the live cloud. The second term of Equation 7 is the so called ARAP regularization term, which penalize inconsistent local transformations between nearest graph nodes. Note this term is also essential for inferring the local deformations of the occluded surface part. We use the parameter setting $\lambda_{\text{data}} = 1$ and $\lambda_{\text{reg}} = 30$ in all our experiments, which provides the best performance.

### B. Reconstruction

In order to update the reference point cloud incrementally based on newly observed depth measurement, we activate the reconstruction phase once the deformation field is estimated in the tracking phase. In detail, we follow [23]'s work to employ the Truncated Signed Distance Function (TSDF) for depth image fusion. To achieve this, the algorithm maintains a TSDF volume $\mathbb{V} : \{\mathcal{D}(\mathbf{x}), \Omega(\mathbf{x})\}$ to explicitly describe the state of reference point cloud in the volume voxel $\mathbf{x}$, where $\mathcal{D}(\mathbf{x}) \subseteq [-1, +1]$ encodes the truncated signed distance value for each voxel $\mathbf{x}$, and $\Omega(\mathbf{x}) \subseteq [0, 1]$ is the associated weight. When an new depth image arrives, we first compute the corresponding new TSDF component $d(\mathbf{x})$ of each voxel $\mathbf{x}$ based on the estimated deformation field:

$$d(\mathbf{x}) = \max\left(\min\left(\frac{I(\mathbf{u}) - \lfloor\tilde{\mathbf{x}}\rfloor_z}{\tau}, 1\right), -1\right), \quad (8)$$

where $I(\cdot)$ represents the live depth image. $\tilde{\mathbf{x}}$ represents the voxel deformed from $\mathbf{x}$ using Equation 6 and $\lfloor\tilde{\mathbf{x}}\rfloor_z$ represents the position of point $\tilde{\mathbf{x}}$ along Z-axis. $\mathbf{u}$ represents the projective pixel of voxel $\tilde{\mathbf{x}}$ in image $I$. $\tau$ is the truncated threshold of TSDF value. After that, we integrate the new TSDF component $d(\mathbf{x})$ into the reference volume $\mathbb{V}$ as

$$\mathcal{D}(\mathbf{x}) \leftarrow \frac{\mathcal{D}(\mathbf{x})\Omega(\mathbf{x}) + d(\mathbf{x})\omega(\mathbf{x})}{\Omega(\mathbf{x}) + \omega(\mathbf{x})},$$
$$\Omega(\mathbf{x}) \leftarrow \min\left(\Omega(\mathbf{x}) + \omega(\mathbf{x}), \Omega_{\max}\right),$$

where $\omega(\mathbf{x})$ is the weight we assign to the new components and $\Omega_{\max}$ is the upper threshold of the weight.

From the updated volume $\mathbb{V}$, we extracted an new reference point cloud based on the marching cubes algorithm [24]. Because we integrate the geometry captured by multiple depth images into the TSDF volume $\mathbb{V}$, the extracted point cloud has the advantage of being occlusion-robust.

## VII. Experiments and Results

We test our algorithm on a dual-arm robot called ABB Yumi which has 7 degrees of freedom in each arm. Our vision system includes an RGB-D camera called Intel Realsense which can provide color and depth images with 30 FPS. The entire system setup is shown in Figure 1.

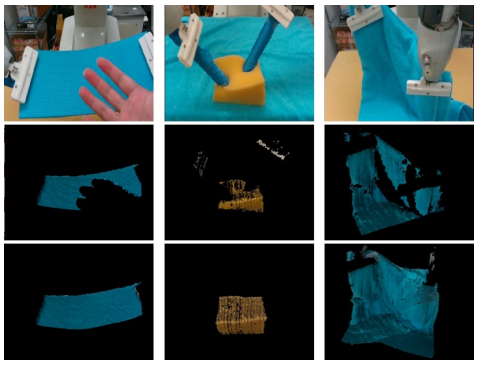


Fig. 7: Results of our occlusion removing algorithm. Row 1 is color images received from the RGB-D camera; Row 2 is raw point clouds generated from RGB-D images; and Row 3 is point clouds generated from our algorithm.

In our experiment, we first test our occlusion removing algorithm and then validate our controller in a set of manipulation tasks and then we compare our controller with several controllers proposed in previous work and discuss the comparison result.

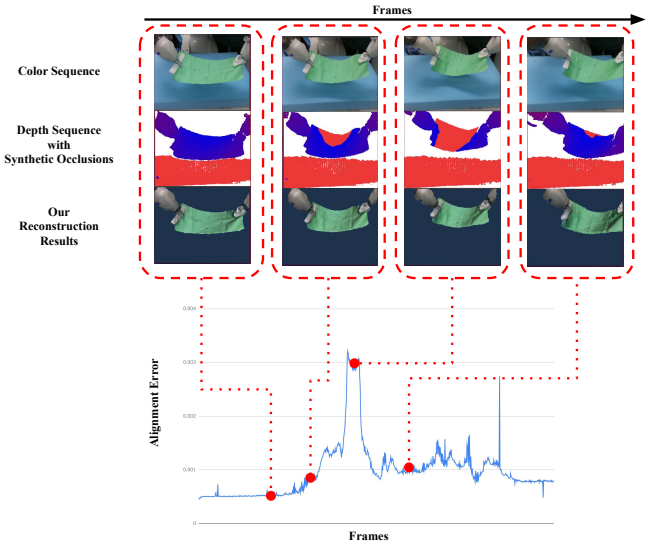



Fig. 8: Top: reconstruction results of a deforming blanket with synthetic occlusions whose masks are highlighted as red regions in the depth sequence. Bottom: the alignment errors between the generated shape models of our method and the raw depth sequence without synthetic occlusions. Our method provides reliable shape estimation during occlusion. Please refer to videos for more details.

### A. Performance of occlusion removing

To show the performance of our method, we compare the occlusion removing algorithm to the raw point cloud in this experiment, with result shown in Figure 7. We can see that these raw point clouds directly received from the RGB-D camera are occluded due to the grippers or objects' deformation. Compared to the raw point cloud, our occlusion

removing algorithm provides a relatively complete point cloud and removes the occlusion parts in the raw point cloud.

As a qualitative evaluation of our occlusion removing algorithm, we further conduct a blanket deforming task with synthetic occlusion masks ( Figure 8). We generate the masks by ignoring in the original depth sequence the depth values greater than a preset threshold and then the corresponding surface parts will become invisible to our algorithm. We evaluate the accuracy of our method, especially for the estimate of the occluded surface parts, by measuring the non-rigid alignment error between the reconstructed shape model and the original depth data without synthetic occlusions, and the results are shown in the bottom part of Figure 8.

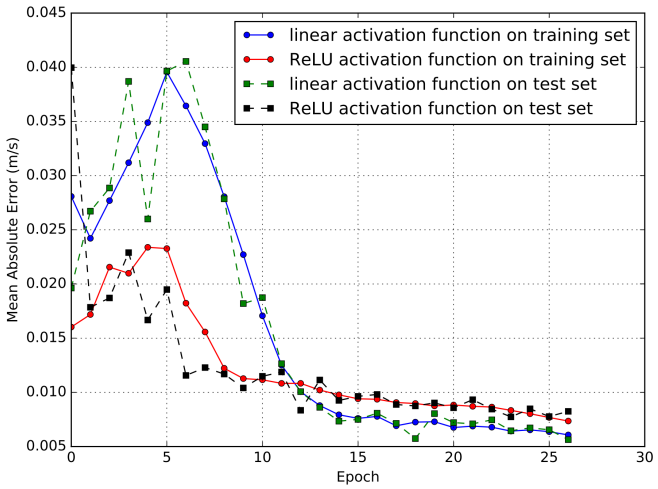



Fig. 9: The Mean Absolute Error (MAE) comparison between linear and ReLU activation function.

### B. Performance of DNN controller

We test our DNN controller in a set of manipulation tasks including rolled towel bending, peg-in-hole, plastic sheet bending, towel folding, and sponge manipulation. We set a target configuration for each task. As shown in Figure 10, our controller accomplished all these tasks. We run our algorithm 10 times starting from a random configuration for each task and the average success rate of these manipulation tasks is about 90%. In particular, the success rate for task 1 is nearly 100% but is only 70% for task 2. Other tasks only fail once during the test. The failure cases in task 2 are the inaccurate fabric assembly due to small holes and random noises from the 3D camera.

To show the performance of our controller quantitatively, we collect an offline data set containing data pairs of the velocity of feature and grippers. We train our controller using 75% data and test the trained controller using the remaining 25% data. We choose the linear function rather than the popular ReLU as the activation function in our DNN, because we found that the linear activation can provide lower Mean Absolute Error (MAE) than ReLU, as shown in Figure 9.

To compare the performance of our DNN with that of the linear model [5], [9], we check the regression results of both models when predicting the two dimensional control velocity (along $x$ and $z$ axis) of the end-effector. As shown in Figure 11, both models can fit the ground truth sufficiently good, but the DNN model is more accurate. We also compare the MAE of our DNN controller with that of the linear model in Table I, and again, DNN controller provides a better result.

| Model | Mean Absolute Error (m/s) | Standard Deviation (m/s) |
|---|---|---|
| Task 1-DNN | 0.0092 | 0.0078 |
| Task 1-LM | 0.0090 | 0.0079 |
| Task 2-DNN | 0.0096 | 0.0082 |
| Task 2-LM | 0.0095 | 0.0083 |
| Task 3-DNN | 0.0059 | 0.0050 |
| Task 3-LM | 0.026 | 0.024 |
| Task 4-DNN | 0.0061 | 0.0049 |
| Task 4-LM | 0.027 | 0.021 |
| Task 5-DNN | 0.0058 | 0.0051 |
| Task 5-LM | 0.025 | 0.024 |

TABLE I: Comparison of the Mean Absolute Error and the Standard Deviation between the DNN model and the linear model (LM).

We further compare our DNN model with the GP model in [8] on task 3. Figure 12 shows the recorded errors between current and target feature values when using controllers based on 3- and 5-layer DNN models and the GP model for manipulation. The 5-Layer DNN controller converges faster than the GP model with a per-iteration running time similar to the GP model.

We also investigate how many data points are necessary for high-quality performance. In task 3, we initialize the DNN model with 100 frames of random movements and then show in Figure 13 the feature error afterwards by running the DNN controller. We can observe that the error quickly decreses given new data frames.

## VIII. Conclusion and Future Work

In this paper, we present a novel controller for deformable object manipulation which is a challenging problem in robotic manipulation. Our controller is based on Deep Neural Network (DNN) and trained from data while manipulation. This online learning process improves the model's robustness and accuracy. In addition, we introduce a novel feature to describe the states of 3D deformable objects while manipulation. This dimension-fixed feature is suitable for our method to train a controller. Furthermore, in order to deal with the occlusion problem occurred in manipulation tasks, we propose an occlusion removing algorithm. Finally, we test our controller and algorithm in a set of deformable object manipulation tasks.

In our future work, we investigate to train our DNN controller using Reinforcement Learning (RL) algorithm to achieve some complicated manipulation tasks like cloth folding and robot-assisted surgery.

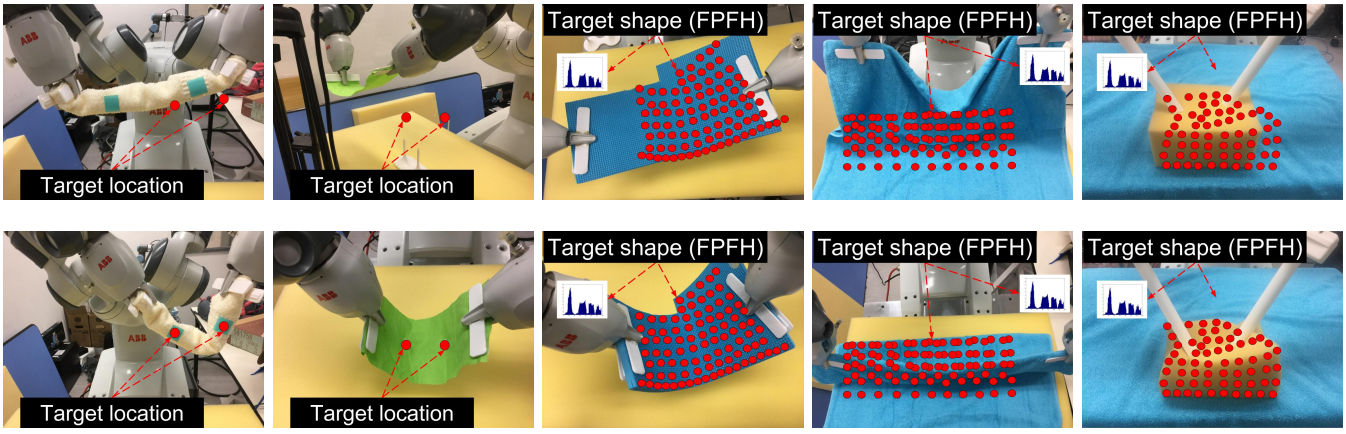
## References

[1] W. Wang, D. Berenson, and D. Balkcom, "An online method for tight-tolerance insertion tasks for string and rope," in *ICRA*, 2015, pp. 2488–2495.

[2] S. Miller, J. van den Berg, M. Fritz, T. Darrell, K. Goldberg, and P. Abbeel, "A geometric approach to robotic laundry folding," *IJRR*, vol. 31, no. 2, pp. 249–267, 2011.

[3] M. Cusumano-Towner, A. Singh, S. Miller, J. F. O'Brien, and P. Abbeel, "Bringing clothing into desired configurations with limited perception," in *ICRA*, 2011, pp. 3893–3900.

[4] D. Kruse, R. J. Radke, and J. T. Wen, "Collaborative human-robot manipulation of highly deformable materials," in *ICRA*, 2015, pp. 3782–3787.




(a) (b) (c) (d) (e)

Fig. 10: The set of tasks used to evaluate the performance of our approach: (a) task 1 - rolled towel bending, (b) task 2 - peg-in-hole for fabric, (c) task 3 - plastic sheet bending, (d) task 4 - towel folding, and (e) task 5 - sponge manipulation. The first row shows the initial state of each object before the manipulation and the second row shows the goal states of the object after the successful manipulation.

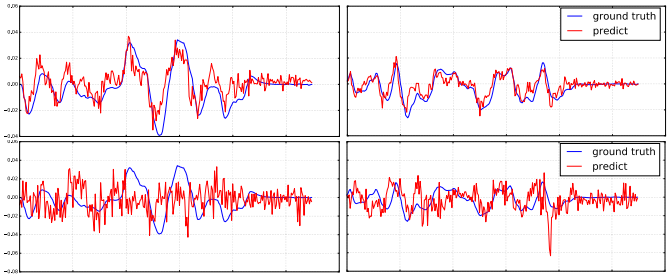



Fig. 11: Regression results of our DNN model (row 1) and linear model (row 2) for predicting the two dimensional control velocity (left and right for $v_x$ and $v_z$ respectively).

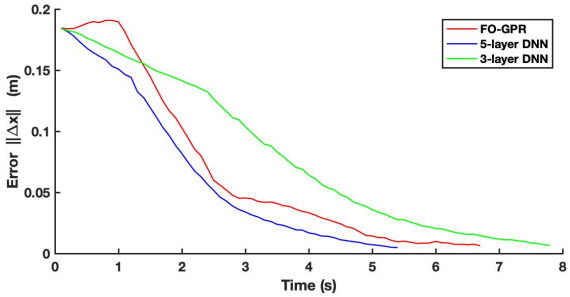



Fig. 12: Comparison between DNN and GP model in task 3. The error means the difference between target and current feature values in each time step while manipulation.

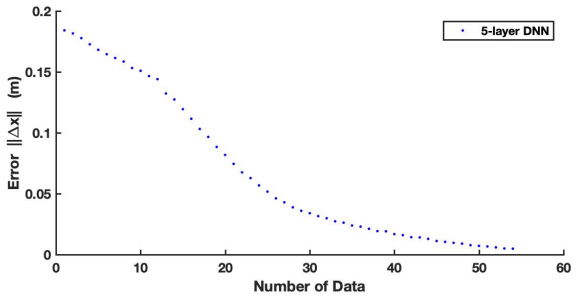



Fig. 13: Relation between the DNN model performance and the number of data frames.

[5] D. Navarro-Alarcon and Y.-H. Liu, "Fourier-based shape servoing: a new feedback method to actively deform soft objects into desired 2-d image contours," *TRO*, vol. 34, no. 1, pp. 272–279, 2018.
[6] S. Patil and R. Alterovitz, "Toward automated tissue retraction in robot-assisted surgery," in *ICRA*, 2010, pp. 2088–2094.
[7] J. Schulman, J. Ho, C. Lee, and P. Abbeel, "Generalization in robotic manipulation through the use of non-rigid registration," in *ISRR*, 2013.
[8] Z. Hu, P. Sun, and J. Pan, "Three-dimensional deformable object manipulation using fast online gaussian process regression," *RAL*, vol. 3, no. 2, pp. 979–986, 2018.
[9] D. Navarro-Alarcon, H. M. Yip, Z. Wang, Y. H. Liu, F. Zhong, T. Zhang, and P. Li, "Automatic 3-d manipulation of soft objects by robotic arms with an adaptive deformation model," *TRO*, vol. 32, no. 2, pp. 429–441, 2016.
[10] J. Tian and Y.-B. Jia, "Modeling deformations of general parametric shells grasped by a robot hand," *TRO*, vol. 26, no. 5, pp. 837–852, 2010.
[11] Y.-B. Jia, F. Guo, and H. Lin, "Grasping deformable planar objects: Squeeze, stick/slip analysis, and energy-based optimalities," *IJRR*, vol. 33, no. 6, pp. 866–897, 2014.
[12] H. Lin, F. Guo, F. Wang, and Y.-B. Jia, "Picking up a soft 3d object by "feeling" the grip," *IJRR*, vol. 34, no. 11, pp. 1361–1384, 2015.
[13] A. Clegg, W. Yu, Z. Erickson, C. K. Liu, and G. Turk, "Learning to navigate cloth using haptics," in *IROS*, 2017, pp. 2799 – 2805.
[14] D. McConachie and D. Berenson, "Bandit-based model selection for deformable object manipulation," *arXiv:1703.10254*, 2017.
[15] D. Navarro-Alarcon, Y. H. Liu, J. G. Romero, and P. Li, "Model-free visually servoed deformation control of elastic objects by robot manipulators," *TRO*, vol. 29, no. 6, pp. 1457–1468, 2013.
[16] S. Hirai and T. Wada, "Indirect simultaneous positioning of deformable objects with multi-pinching fingers based on an uncertain model," *Robotica*, vol. 18, no. 1, pp. 3–11, 2000.
[17] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu, "Fast 3d recognition and pose using the viewpoint feature histogram," in *IROS*, 2010, pp. 2155–2162.
[18] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (FPFH) for 3d registration," in *ICRA*, 2009, pp. 1848–1853.
[19] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz, "Learning informative point classes for the acquisition of object model maps," in *ICARCV*, 2008, pp. 643–650.
[20] R. A. Newcombe, D. Fox, and S. M. Seitz, "Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time," in *CVPR*, 2015, pp. 343–352.
[21] R. W. Sumner, J. Schmid, and M. Pauly, "Embedded deformation for shape manipulation," *TOG*, vol. 26, no. 3, p. 80, 2007.
[22] O. Sorkine and M. Alexa, "As-rigid-as-possible surface modeling," in *SGP*, vol. 4, 2007, pp. 109–116.
[23] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *ISMAR*, 2011, pp. 127–136.
[24] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *SIGGRAPH*, vol. 21, no. 4, 1987, pp. 163–169.