Online 3D Deformable Object Classification for Mobile Cobot Manipulation

Khang Nguyen Tuan Dang Manfred Huber

***Abstract*—Vision-based object manipulation in assistive mobile cobots essentially relies on classifying the target objects based on their 3D shapes and features, whether they are deformed or not. In this work, we present an auto-generated dataset of deformed objects specific for assistive mobile cobot manipulation using an intuitive Laplacian-based mesh deformation procedure. We first determine the graspable region of the robot hand on the given object's mesh. Then, we uniformly sample handle points within the graspable region and perform deformation with multiple handle points based on the robot gripper configuration. In each deformation, we identify the orientation of handle points and prevent self-intersection to guarantee the object's physical meaning when multiple handle points are simultaneously applied to the mesh at different deformation intensities. We also introduce a lightweight neural network for 3D deformable object classification. Finally, we test our generated dataset on the Baxter robot with two 7-DOF arms, an integrated RGB-D camera, and a 3D deformable object classifier. The result shows that the robot is able to classify real-world deformed objects from point clouds captured at multiple views by the RGB-D camera. The source code is available at https://github.com/mkhangg/deformable_cobot.**

***Index Terms*—3D vision, deformable object manipulation**

I. Introduction

Manipulating everyday objects based on three-dimensional (3D) visual information is crucial for assistive robot applications but enormously depends on the 3D shapes of the objects. Everyday objects include a wide range of things made from various materials with different levels of softness and usually have simple topological forms. However, manipulating these objects, especially deformable objects (*e.g.*, soda cans, toy balls, and disposal cups), with different levels of forces at each time results in different deformed shapes, challenging the robotic vision to re-classify deformed objects efficiently in later manipulation. Indeed, robots might eventually lose the features of the target objects and can no longer detect those objects in deformed shapes. For this reason, an auto-generated dataset of deformable everyday objects by introducing an intuitive mesh deformation procedure is essential to facilitate robust, vision-based deformable object manipulation for assistive robotics development.

Significant effort has been spent in previously on generating datasets of 3D deformable objects, but focused primarily on humanoid shapes embedded with human skeleton models [1]–[3]. These humanoid shapes are helpful in unmanned vehicle system applications regarding classifying and detecting pedestrians. However, they are inapplicable for deformable object manipulation as assistive mobile cobots are highly likely to deal with everyday objects, such as cans, balls, and cups made from homogeneous materials like tin, foam, and paper.

All authors are with the Learning and Adaptive Robotics Laboratory, Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX 76013, USA. (emails: khang.nguyen8@mavs.uta.edu, tuan.dang@uta.edu, huber@cse.uta.edu)

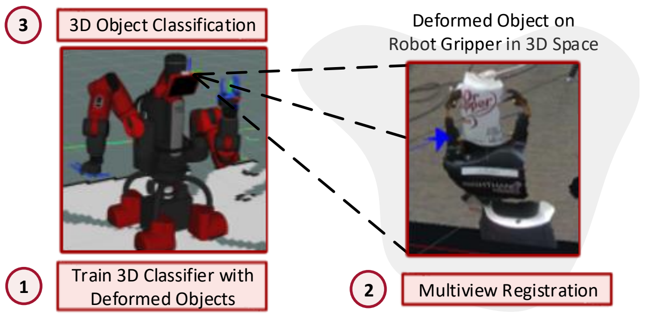

3
3D Object Classification
Deformed Object on Robot Gripper in 3D Space
1
Train 3D Classifier with Deformed Objects
2
Multiview Registration

Fig. 1: Overview of online 3D deformable object classification for assistive mobile cobots with an integrated RGB-D camera.

Recent works in deformable object manipulation mainly focus on folding towels and clothes [4]–[6] or ropes [7] of various materials. However, a weakness of these techniques is assuming that robots have prior knowledge of the target-deformed object. Thus, these techniques only enable robots to deal with garment-like objects and preclude themselves from learning diverse deformed shapes of everyday objects.

To fill this gap in deformable object manipulation, we propose an auto-generated dataset of deformable everyday objects using an intuitive mesh deformation procedure. Our dataset includes tin cans, foam balls, and paper cups of various sizes in undeformed and deformed shapes. To generate deformed objects, we identify the graspable region of the robot gripper on the object, uniformly sample the handle points within the graspable region, and deform the object's mesh with different deformation intensities. Using this data, the mobile cobot can learn to re-recognize everyday objects in both undeformed and deformed shapes after manipulating them (Fig. 1).

To generate such a dataset, we have to ensure that the generated deformed objects have physical meaning by overcoming the following challenges: (1) sampling a set of handle points that could physically express grasp poses, (2) identifying the orientation of handle points, (3) preventing self-intersection when multiple handle points are simultaneously applied at multiple levels of deformation intensities, and (4) teach the robot to recognize those deformed objects efficiently.

In this work, we make the following contributions: (1) presenting a deformation procedure to auto-generate a dataset of deformable everyday objects, (2) verifying our dataset with convolutional neural network (CNN) classifiers with a spatial transformer network for 3D objects, and (3) performing experiments on the Baxter robot with two 7-DOF arms and an Intel RealSense D435i RGB-D camera with real-world objects.

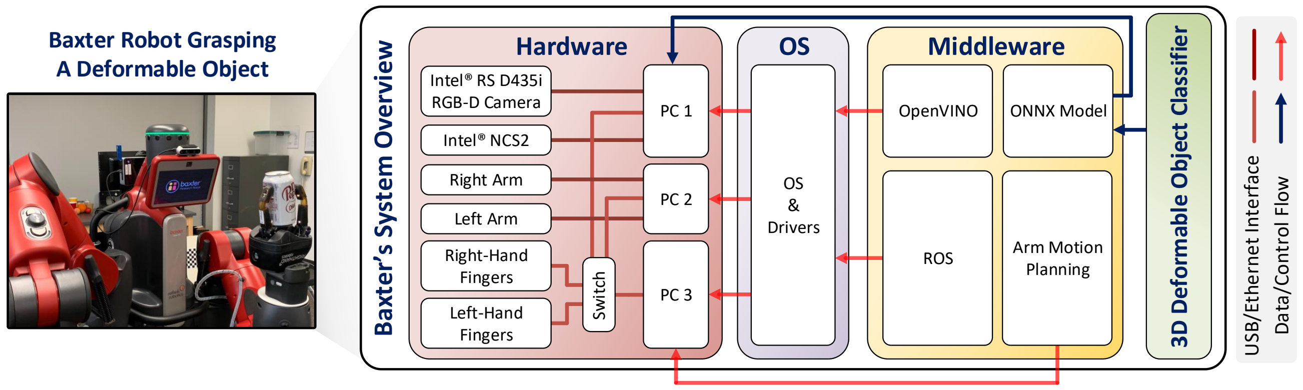


Fig. 2: The robot system includes a hardware layer, an OS layer, a middleware layer, and a 3D deformable object classifier.

II. Related Work

3D Mesh Deformation: Mesh deformation and editing techniques in 3D have been a well-researched area in computer graphics, including fitting to Laplacian coordinates [8]–[10], computing Laplacian energy [11], and optimizing Dirichlet surface energy [1]. These traditional shape deformation frameworks well-handle detail-preserving standards when a smooth large-scale deformation is applied to the original shapes, which matches users' intuition and expectation of shape deformation. Recent incremental development in deep learning (DL) also introduces networks that serve object deformation purposes which output either a target-deformed mesh or point cloud. As a matter of fact, these DL techniques conceptually differ from the traditional mesh deformation techniques as they require an RGB image of a deformed target object [12]–[14], a common intermediate template representation [15], or user-defined high-level deformation intentions [16] as an additional input along with the given point cloud to deform the original objects as desired. Nevertheless, these DL techniques are not well-suited for generating a dataset of everyday objects that are deformed by robot manipulation as it is impossible to capture all possibilities of deformations with RGB images, as well as to find the common template representation for all objects with different shapes, and it is generally ambiguous to define user-defined high-level deformation intentions.

In the context of robotic applications, especially robot manipulation, both traditional and DL deformation techniques lack the intuition of (i) where handle points for deformation are to efficiently cover all possibilities, (ii) how much deformation intensity should be applied to the surface, (iii) which direction with respect to the object's centroid should the deformation be applied in, and (iv) how to ensure the physical meaning of deformed objects (*e.g.*, preventing self-intersection on deformed meshes when multiple handle points are applied). In this work, we present an intuitive technique followed by the traditional Laplacian-based mesh deformation [8] that can sufficiently address the above-mentioned intuition of robot manipulation based on the robot gripper configurations.

3D Object Classification: To classify objects in the form of point clouds, hand-crafted feature engineering, and DL-based feature extractions are common methods to represent point clouds in feature space. Hand-crafted feature engineering can work with some simple objects but fails to generalize objects' representations. Furthermore, traditional CNNs can not be applied directly to point clouds as with images because of their irregular structures. To enable point clouds to work with CNNs, we need to convert the point cloud into ordered structure data, a so-called feature representation. The projection-based methods [17], [18] project multiple views of a point cloud into multiple images so we can directly apply CNNs on top of these images. Alternative to the projection-based method, researchers exploit the voxel-based method [19], [20] to organize a point cloud into 3D grids of voxels. By voxelizing a point cloud, we can apply 3D CNNs to extract its features. In this work, we use a point-based method [21], [22] to extract features from a point cloud, directly processing point clouds as input and avoiding extra computation like in projection-based and voxel-based methods.

System Architecture: As our end goal is to build a robotic system that is able to efficiently recognize objects in 3D space, we first organize software components, then connect them to hardware components, and synchronize between them, which is the first step to localizing objects in a rich representation. Organizing software components in a constrained environment is challenging since the robot system has to enable multiple sensor modalities, actuators, and communication stacks simultaneously. To maintain system reliability, we design our software architecture under modularization, layerization [23]–[25], and high synchronization requirements [26].

III. Overview of Robot System

We customize the Baxter robot to fit our application, as illustrated in Fig. 2. The Baxter robot was originally equipped with one computer controlling two 7-DOF arms. We then add two grippers, controlled by another computer. An Intel RealSense D435i RGB-D camera is attached to the third computer to reconstruct 3D scenes. These three computers are connected to an Ethernet switch and exchange messages through Robot Operating System (ROS) messages. Note that ROS messages are built on top of TCP/IP supported by the native OS: Ubuntu 20.04 and ROS Noetic. To reduce the system's complexity, we employ an onboard Intel Graphic Processor Unit (GPU) to execute our deformable object classifier while the training stage of the classifier takes place on another computer with

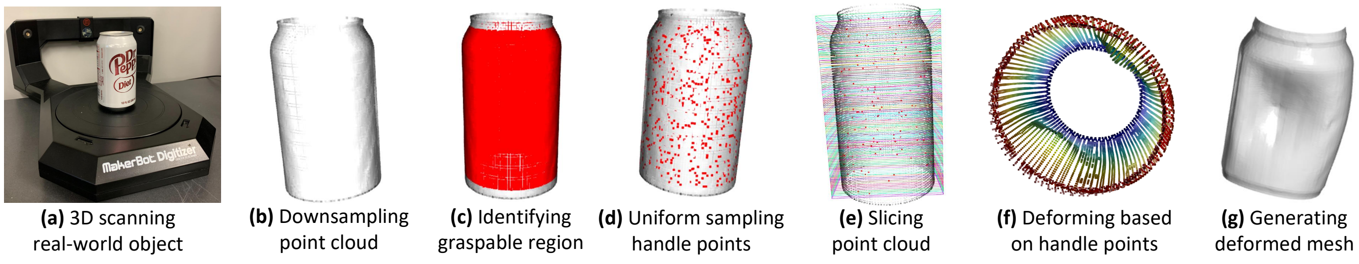

(a) 3D scanning real-world object
(b) Downsampling point cloud
(c) Identifying graspable region
(d) Uniform sampling handle points
(e) Slicing point cloud
(f) Deforming based on handle points
(g) Generating deformed mesh

Fig. 3: The deformation procedure includes **(a)** 3D scanning real-world objects, **(b)** voxel downsampling scanned point clouds, **(c)** identifying the graspable region of the robot on point clouds, **(d)** uniform sampling handle points, **(e)** slicing point clouds, **(f)** performing deformation on handle points based on robot gripper's configuration, and **(g)** generating deformed meshes.

a dedicated NVIDIA GPU. To run the classifier on the Intel GPU, we use the native library developed by Intel, OpenVINO, to implement our software. The model resulting from the training process is then converted into an ONNX model and eventually loaded on a native GPU. When the application layer takes a multiview of RGB-D images, it reconstructs the 3D view, and converts the view into a point cloud. Herein, the application layer sends the point cloud to the native GPU, where the loaded classifier is ready to classify the point cloud. We train the classifier on our server and convert it into ONNX format so that we run it on the existing robot hardware, such as the onboard Intel GPU, using the OpenVINO library. Further details in training and testing the classifier are discussed in Sec. V, and the deployment stage on the robot system with real-world objects is evaluated in Sec. VI-C.

IV. 3D Deformable Object Generation

In this section, we denote a given mesh with n vertices as $\mathcal{M} := (E, V, F)$, where E, V, and F are sets of edges, vertices, and faces in $\mathcal{M}$, respectively, and denote a given point cloud with n points as $\mathcal{P} := \{\mathbf{p}_i\}$, for $i = 1, 2, ..., n$, where $\mathbf{p}_i = [x_i, y_i, z_i]^T$ represent points in $\mathcal{P}$ in Cartesian coordinates. The set of vertices V in $\mathcal{M}$ contains $\{\mathbf{v}_i\}$, similar to the set of points $\{\mathbf{p}_i\}$ in $\mathcal{P}$, for $i = 1, 2, ..., n$. Likewise, we denote the deformed mesh as $\mathcal{M}' := (E', V', F')$.

A. *Smooth-Regularization ARAP Deformation*

Let $\mathbf{v}_i \in V$ denote the original vertices in a mesh $\mathcal{M}$, and $\mathbf{v}_i' \in V'$ denote the transformed vertices on the deformed mesh $\mathcal{M}'$. Without loss of generality, in the set of transformed vertices V', we arrange vertices $\mathbf{v}_i'$ as follows:

$$\begin{cases} \mathbf{v}_i' = \mathbf{c}_i = \mathbf{v}_i, & \text{for } i = 1, 2, ..., m \\ \mathbf{v}_i' = T_i(\mathbf{v}_i)\ , & \text{for } i = m+1, m+2, ..., n \end{cases} \tag{1}$$

where $T_i(\cdot)$ represents the transformation from the original vertex $\mathbf{v}_i$ to the transformed vertex $\mathbf{v}_i'$, $\mathbf{c}_i$ denotes a constraint point in $\mathcal{M}'$ that is invariant to vertex $\mathbf{v}_i$ in $\mathcal{M}$, and $m < n$ with m being the number of invariant vertices, $\mathbf{c}_i$, and n being the total number of vertices, $\mathbf{v}_i$.

The difference between one vertex with respect to its neighbors is approximated by the differential (Laplacian or δ) of Cartesian coordinates [27] as follows:

$$\delta(\mathbf{v}_i) = \frac{1}{|\Omega_i|} \sum_{j \in \mathcal{N}(\mathbf{v}_i)} \frac{(\cot\alpha_{ij} + \cot\beta_{ij})}{2} (\mathbf{v}_i - \mathbf{v}_j) \tag{2}$$

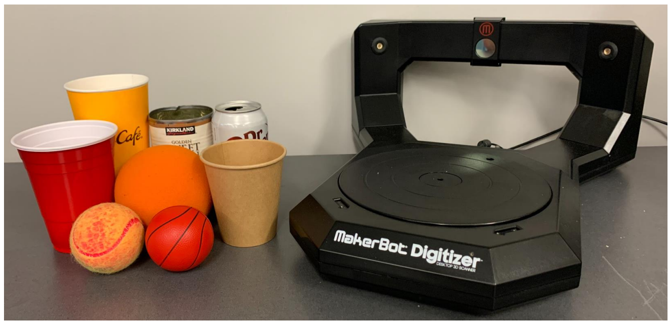


Fig. 4: Real-world objects (tin cans, foam balls, and paper cups) of various sizes (left) and the MakerBot Digitizer 3D scanner (right).

where $\delta(\mathbf{v}_i)$ represents the δ-coordinates of vertex $\mathbf{v}_i$, $\mathbf{v}_j$ represent a one-ring neighboring vextex of vertex $\mathbf{v}_i$, $|\Omega_i|$ is the area of the Voronoi cell of $\mathbf{v}_i$, $\mathcal{N}(\mathbf{v}_i)$ is the set of points that are one-ring neighbours of $\mathbf{v}_i$, and α_{ij}, β_{ij} are two opposite angles of the edge connecting $\mathbf{v}_i$ and $\mathbf{v}_j$.

We deform the mesh $\mathcal{M}$ by using the smooth-regularization As-Rigid-As-Possible (ARAP) deformation. Using Eq. 1 and Eq. 2, the deformation technique is formalized as follows:

$$\begin{aligned} \mathcal{E}(V') = &\sum_{i=1}^{n} \left|\left| \frac{1}{|\Omega_i|} \sum_{j \in \mathcal{N}(\mathbf{v}_i')} \omega_{\cot} \left(\mathbf{v}_i' - \mathbf{v}_j'\right) - R(\mathbf{v}_i') \cdot \delta(\mathbf{v}_i) \right|\right|_2^2 \\ &+ \sum_{i=m+1}^{n} \left|\left| \mathbf{v}_i' - \mathbf{c}_i \right|\right|_2^2 \\ &+ \sum_{i=1}^{n} \sum_{j \in \mathcal{N}(\mathbf{v}_i')} \alpha_{smooth} \cdot S(\mathcal{M}) \left|\left| R(\mathbf{v}_i') - R(\mathbf{v}_j') \right|\right|_2^2, \end{aligned} \tag{3}$$

$$\text{with } \delta(\mathbf{v}_i) = \frac{1}{|V|} \sum_{i=1}^{|V|} \frac{1}{|\Omega_i|} \sum_{j \in \mathcal{N}(\mathbf{v}_i)} \omega_{\cot}(\mathbf{v}_i - \mathbf{v}_j)$$

$$\text{and } \omega_{\cot} = (\cot\alpha_{ij} + \cot\beta_{ij})/2$$

where $\mathcal{E}(V')$ represents the error function for the set of transformed vertices V' in $\mathcal{M}'$, $R(\mathbf{v}_i')$ represents the rotation matrix for vertex $\mathbf{v}_i'$, α_{smooth} is the regularization for surface smoothness, and $S(\mathcal{M})$ denotes the surface area of $\mathcal{M}$.

We also use the average of cotangents of two adjacent angles as weights for its compensation for non-uniform shapes [28] and its better geometric representation in terms of the query vertex, one-ring neighboring vertices, and adjacency angles.

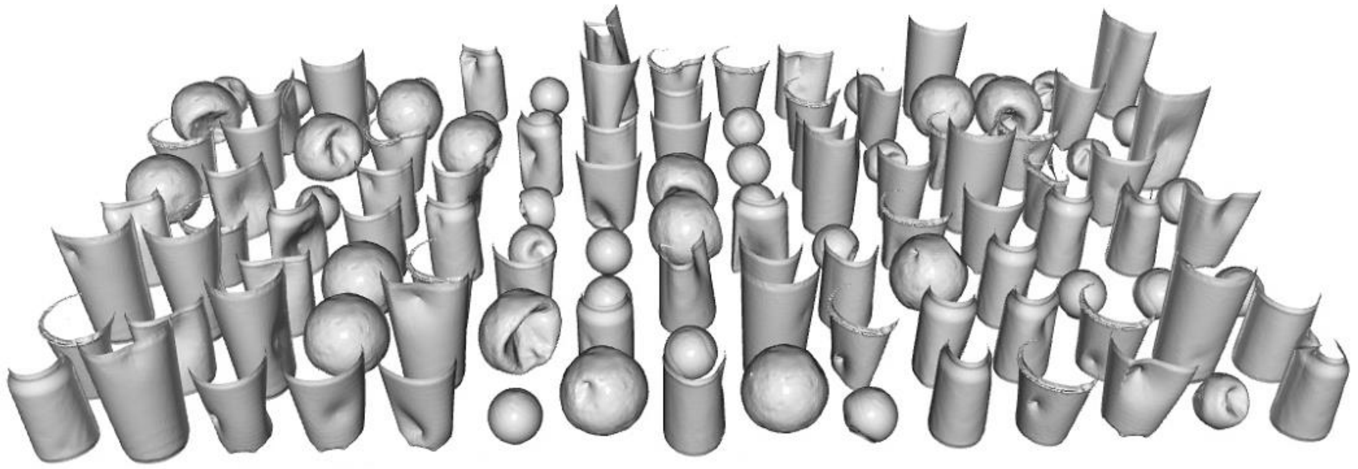


Fig. 5: Demonstration of a subset of our auto-generated 3D deformable object dataset.

B. 3D Deformable Object Generation

The deformation procedure is shown in Fig. 3, from scanning real-world objects to generating deformed meshes.

1) 3D Scanning Real-World Objects: We use the MakerBot Digitizer 3D scanner to scan real-world objects (Fig. 4) as the scanner handles the point cloud registration procedure when performing multiple scans. Our raw scans include various-sized tin cans, foam balls, and paper cups.

2) Downsampling Point Clouds: Raw point clouds given by the 3D scanner contain approximately 50,000 points each. We voxelize to downsample the point cloud and reconstruct the mesh from the downsampled point cloud using the Poisson surface reconstruction method [29] with the maximum tree depth of 9. The number of points in downsampled point clouds is about 10,000 points each (Fig. 3b), and each downsampled point cloud is less than 1 MB. In fact, this step is necessary to save computational time in later steps as the number of points has reduced sufficiently but still maintains the object's details.

3) Identifying Graspable Region: To identify the graspable region on the raw scan of the objects, we prioritize the graspable region along the long side of the objects based on the intuition of grasp poses [30], [31]. An example of identifying the graspable region of a tin can is illustrated in Fig. 3c.

4) Sampling Handle Points: Instead of brute-forcing all points in the graspable region to be handle points, we uniformly sample them from the points in the graspable region. By doing so, we can also represent all possibilities of grasping poses using a uniformly distributed subset. An example of sampling handle points of a tin can is shown in Fig. 3d.

5) Handle Points for Multiple Deformations: The robot's hand is usually in the form of a gripper; we identify at least two grasping points (or handle points) for the grasp pose in each deformation. As illustrated in Fig. 3e, for each point in the sampling of handle points in the previous step, we slice the object into slices with the thickness of 1 and take the point that has the farthest Euclidean distance to the sampled point as the second point for the grasp pose.

6) Orientation of Handle Points: Note that Eq. 1 reveals the fact that the transformation for one vertex, $T_i(\mathbf{v}_i)$, could be an arbitrary transformation, which does not guarantee the physical meaning of the deformed object under grasping conditions. Thus, we define our transformation as $T_{-\mathbf{n}_i}(\mathbf{v}_i)$, where $T_{-\mathbf{n}_i}(\cdot)$ denotes the transformation that is in the opposite direction of the normal vector, $\mathbf{n}_i$, of the vertex, $\mathbf{v}_i$, as shown in Fig. 3f.

7) Intensities of Deformation: To perform deformation with different intensities, we define the intensity at $\mathbf{v}_i$, I_i, as:

$$I_i = ||T_{-\mathbf{n}_i}(\mathbf{v}_i) - \mathbf{v}_i|| \tag{4}$$

where $||\cdot||$ represents the vector's magnitude. If I_i exceeds the calculated maximum intensity, $I_i^{\max}$, at that configuration, I_i is set to $I_i^{\max}$, preventing self-intersection on deformed meshes.

8) Generating Deformed Meshes: We use Eq. 3 and Eq. 4 to perform the deformation of multiple handle points with appropriate orientations and deformation intensities as discussed in the previous steps. An example of the deformed can is shown in Fig. 3g. The deformation procedure is applied to each 3D scan of real-world objects in Fig. 4, and our dataset is generated on an Intel Core i7-12700K CPU, taking approximately 23 to 28 minutes for each class. Each class contains roughly 1,600 to 2,000 deformed meshes. A subset of our 3D deformable object dataset is shown in Fig. 5.

V. 3D Deformable Object Classification

To classify an everyday object efficiently in terms of runtime, whether in original or deformed shapes, on the Baxter robot, we design a 3D deformable object classifier as illustrated in Fig. 6. The deformable object classifier takes either the full view of the object (in the testing stage) or the partial view of the object (in the deployment stage) and decides which class the object belongs to.

The input layer takes the input size of $N \times 3$ or $N \times 6$, where N is the number of sampling points from the downsampled point cloud of n points ($n \gg N$). Note that the input size depends on whether the input contains normal vectors or not. The model then passes the input to the spatial transformer network [32] to embed spatial information that is invariant to permutations, rotations, and translations in point clouds. Then, the spatial features are multiplied with the point coordinates $N \times 3$ and re-appended to the normal vector coordinates if

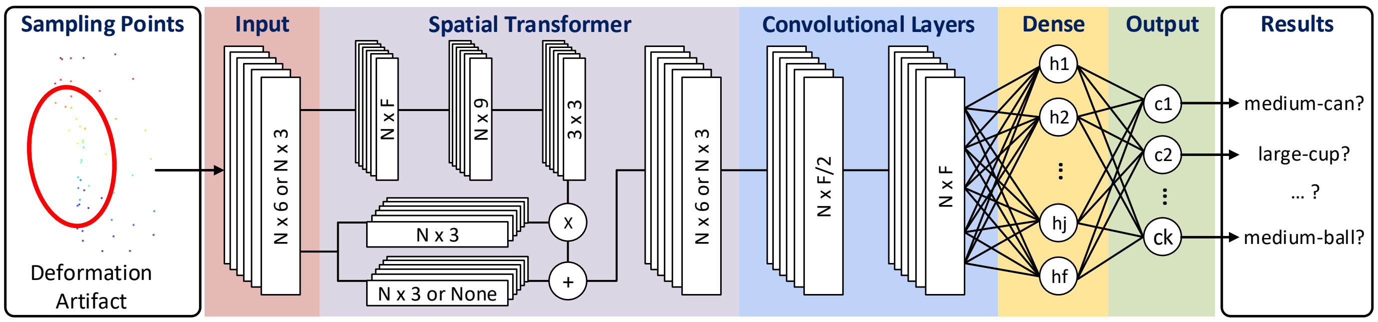


Fig. 6: The 3D deformable object classifier takes uniformly sampling points of a point cloud with deformation artifacts as input and returns probabilities of all classes at the output layer. Hidden layers include a spatial transformer, two convolutional layers, and one fully-connected layer, with $\otimes$ and $\oplus$ indicating matrix multiplication operation and appending operation on batches, respectively.

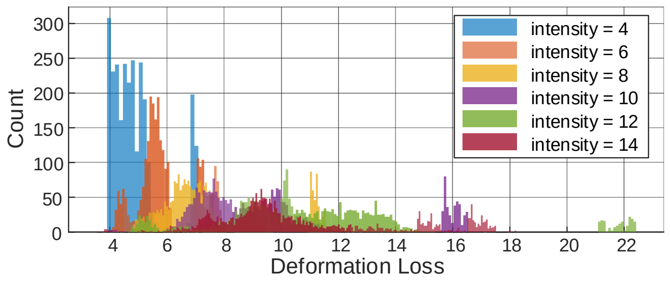


Fig. 7: Distribution of deformation loss of the generated dataset at deformation intensities of 4, 6, 8, 10, 12, and 14, respectively.

needed. Next, two convolutional layers are applied to the input embedded with spatial feature information to extract F features for each input, which results in the size of $N \times F$. Then the feature maps are flattened to a fully-connected layer, and the classes' probabilities are returned at the output layer.

The idea behind using spatial transformers for 3D object classification is to find the different representation that is invariant to permutations, rotations, and translations in point clouds. Afterward, we can apply conventional CNNs on top of these representations to perform the classification task. Therefore, finding the features that represent for aforementioned constraints is the crucial step to classifying a 3D object-based point cloud. To implement this representation, we first transform 3D space information from each point to higher-dimension feature spaces. Next, we select the most representative features among N sample points for each feature in the new feature space. These features represent spatial relationships between points, the so-called spatial feature transformer. Since we extract these features using symmetric functions such as `maxpool()`, we maintain the invariance on the unordered pointset of the point cloud. We then project N points into this feature transform to get a new representation, which is an ordered representation, so that further CNNs or MLP can be applied to classify 3D objects in the form of point clouds.

We evaluate different combinations of numbers of features, F, and numbers of sampling points, N, to see which parameters contribute to learning processing. Through testing different combinations, we can select the most appropriate pair of (N, F) for a specific application which depends on the objects' complexity. The evaluation of the classifier's performance with different design choices is discussed in Sec. VI-B.

VI. Evaluation & Experimental Results

A. Deformation Loss for Auto-Generated Dataset

To validate the deformation of one object with respect to the original mesh, we compute the total loss, $\mathcal{L}$, as a weighted sum of (1) Chamfer distance loss, (2) mesh edge loss, (3) mesh normal loss, and (4) Laplacian smoothing loss, as described:

$$\begin{aligned}\mathcal{L} = w_{dist} \cdot \mathcal{L}_{dist} + w_{edge} \cdot \mathcal{L}_{edge} \\ + w_{normal} \cdot \mathcal{L}_{normal} + w_{smooth} \cdot \mathcal{L}_{smooth}\end{aligned} \tag{5}$$

where $\mathcal{L}_{dist}$, $\mathcal{L}_{edge}$, $\mathcal{L}_{normal}$, and $\mathcal{L}_{smooth}$ are losses for Chamfer distance, mesh edge, mesh normal, and Laplacian smoothing, respectively, and w_{dist}, w_{edge}, w_{normal}, and w_{smooth} are their corresponding weights.

1) Chamfer Distance Loss: To estimate the similarity between the original and the deformed objects in the form of point clouds, we use the Chamfer distance, $\mathcal{L}_{dist}$, between two clouds, which is written as follows:

$$\begin{aligned}\mathcal{L}_{dist} = \frac{1}{|\mathcal{P}_1|}\left[\sum_{\mathbf{p}_i \in \mathcal{P}_1} \min_{\mathbf{p}_j \in \mathcal{P}_2} ||\mathbf{p}_i - \mathbf{p}_j||_2^2\right] \\ + \frac{1}{|\mathcal{P}_2|}\left[\sum_{\mathbf{p}_j \in \mathcal{P}_2} \min_{\mathbf{p}_i \in \mathcal{P}_1} ||\mathbf{p}_j - \mathbf{p}_i||_2^2\right]\end{aligned}$$

where $\mathcal{P}_1$ and $\mathcal{P}_2$ represent the two point clouds, respectively, $\mathbf{p}_i$ and $\mathbf{p}_j$ are i^{th} and j^{th} points in $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively, and $|\mathcal{P}_i|$ indicates the number of points in the point cloud $\mathcal{P}_i$.

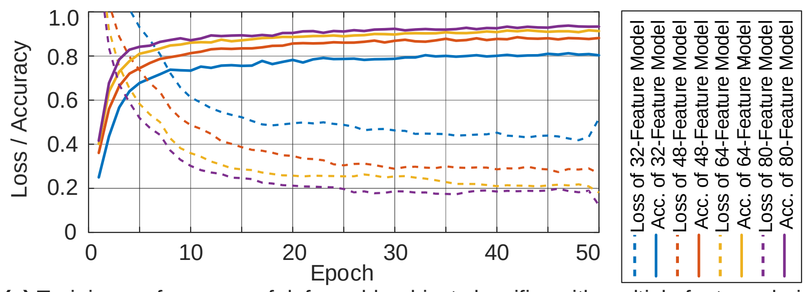
2) Mesh Edge Loss: In deformed meshes, we intuitively expect to preserve the object's details; therefore, we compute the mesh edge loss, $\mathcal{L}_{edge}$, that penalizes long edges:

$$\mathcal{L}_{edge} = \frac{1}{|E|}\left[\sum_{i=1}^{|V|} \sum_{j \in \mathcal{N}(\mathbf{v}_i)} \left(||\mathbf{v}_i - \mathbf{v}_j||_2^2 - L_{target}\right)^2\right]$$

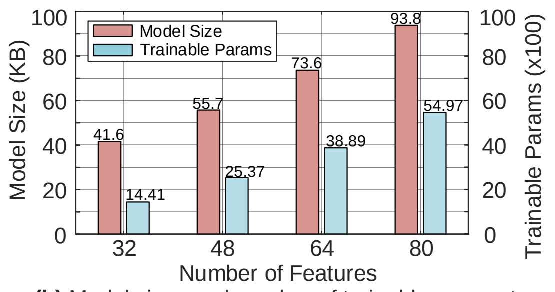
where L_{target} is the target penalized length and $|E|$ represents the number of edges in the deformed mesh $\mathcal{M}'$.

3) Mesh Normal Loss: Due to the fact that the mesh is deformed, each pair of adjacent surfaces are displaced differently. Thus, the mesh normal consistency computes the mesh normal loss, $\mathcal{L}_{normal}$, across the mesh surface as follows:

$$\mathcal{L}_{normal} = \frac{1}{|F|}\left[\sum_{i=1}^{|V|} \sum_{j \in \mathcal{N}(\mathbf{v}_i)} (1 - \cos(\mathbf{n}_i, \mathbf{n}_j))\right]$$



(a) Training performance of deformable object classifier with multiple feature choices



(b) Model sizes and number of trainable parameters

Fig. 8: (a) Training performance with accuracy and loss over the training stage of deformable object classifiers with 32, 48, 64, and 80 features, respectively, and their corresponding **(b)** model sizes (in kilobytes) and numbers of trainable parameters (in hundred).

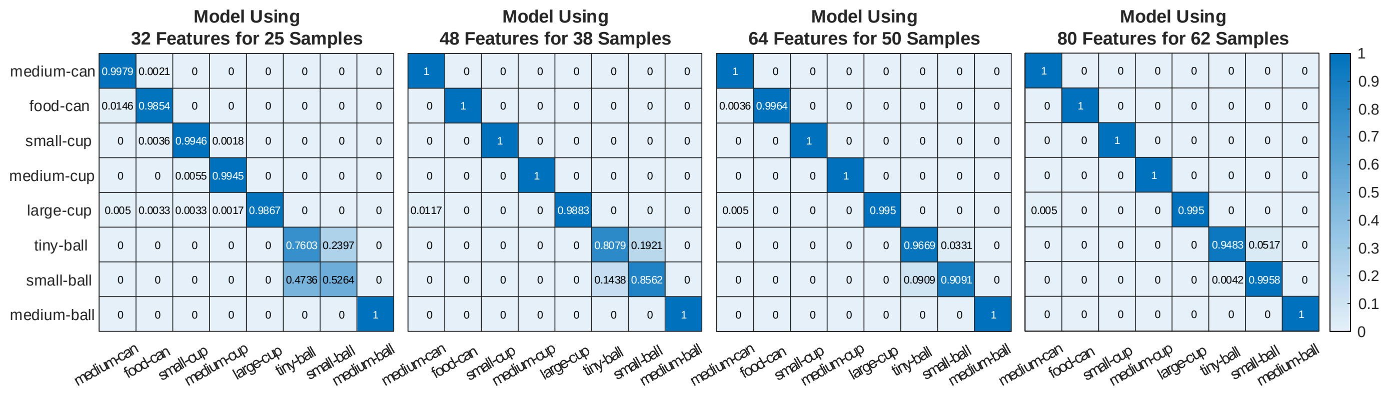


Fig. 9: Confusion matrices of deformable object classifiers with 25, 38, 50, and 62 sample points, respectively.

where $\mathbf{n}_i$ and $\mathbf{n}_j$ denote normal vectors of two shared surfaces of $\mathbf{v}_i$ and $\mathbf{v}_j$, and $|F|$ indicates the number of faces in $\mathcal{M}'$.

4) Laplacian Smoothing Loss: Deforming a given mesh results in a different surface that can be evaluated using the Laplacian smoothing loss, $\mathcal{L}_{smooth}$, for mesh in terms of geometric-related entities to points on the mesh surface:

$$\mathcal{L}_{smooth} = \frac{1}{|V|}\sum_{i=1}^{|V|}\delta(\mathbf{v}_i)$$

The distribution of deformation loss, $\mathcal{L}$, of the generated dataset calculated using Eq. 5 well-presents and generalizes deformations at multiple intensities, as shown in Fig. 7.

B. Performance of Classifier with Different Design Choices

1) Data Splitting: To verify the performance of our network with different design choices of (N, F) on the deformable objects dataset, we split our auto-generated dataset of 3D deformable objects into training (70%) and test sets (30%).

2) Training Performance of Classification Network: We train our 3D deformable object classifier on the NVIDIA GTX 4090 (24 GB) GPU with 70% of the generated dataset for 50 epochs for each of four different numbers of features: 32, 48, 64, and 80 features. The trained models start to converge at the 10^{th} epoch and finally converge at the 30^{th} epoch. As shown in Fig. 8a, the more features extracted from the input point cloud, the more accurate the model obtained. With 80 features, the model achieves the highest accuracy of 93% with the lowest loss of 19%. Meanwhile, the model with 32 features obtains an accuracy of 80% with a loss of 41%. The model with 48 features and 64 features obtains accuracies of 88% and 91% and losses of 28% and 21%, respectively.

3) Model Sizes & Numbers of Trainable Parameters: As we aim to deploy our classifiers on the Baxter robot for fast interpretation on the native onboard Intel GPU with the OpenVINO library, we limit the model sizes not to exceed 100 KB. The number of trainable parameters and model sizes at each number of features are shown in Fig. 8b.

4) Testing Performance of Classification Network: We test our trained classifiers with the remaining 30% of the generated dataset and create the confusion matrices for each of them, as shown in Fig. 9. The classifiers perform better as more sampling points are used in models trained with more features. The model with 32 features underperforms as it is highly likely to confuse tiny and small balls with other classes, such as large and small cups. The model with 48 features still misclassifies between tiny balls and small balls and between large cups and medium cans. The same results occur with the model with 64 features, but this model has a lower likelihood of misclassification. Lastly, the model using 80 features for 62 sampling points outperforms others; however, the failure remains when it still confuses between tiny balls and small balls, but with lower misclassification rates than other models.

C. Deployment on Baxter Robot

1) Experimental Setup: We first mount the Intel RealSense D435i RGB-D camera on the display of the Baxter robot. We then let the Baxter robot grasp one of the deformed objects (cans, cups, and balls) in the dataset using its hand gripper (Fig. 10a). The Baxter robot rotates its wrist to take point

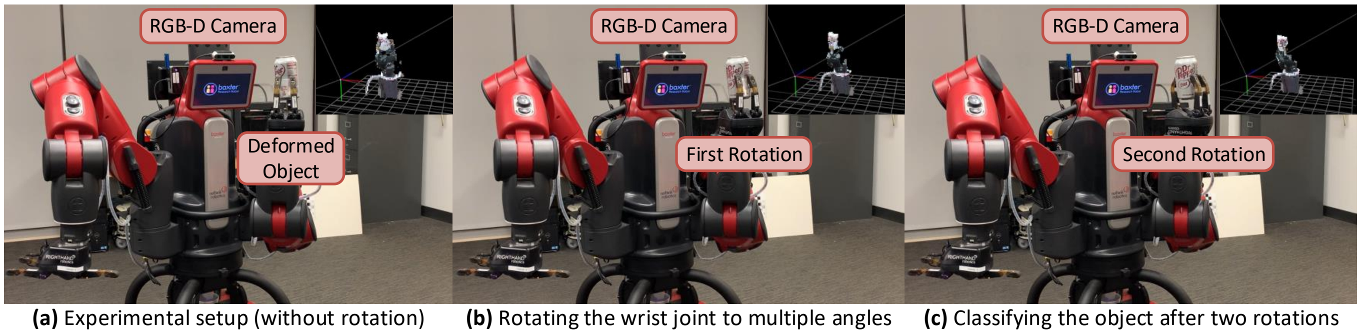

RGB-D Camera
Deformed Object
RGB-D Camera
First Rotation
RGB-D Camera
Second Rotation
(a) Experimental setup (without rotation)
(b) Rotating the wrist joint to multiple angles
(c) Classifying the object after two rotations

Fig. 10: **(a)** Experimental setup for the Baxter robot grasping a deformed object (a deformed can), **(b)** rotating the wrist to take point clouds at multiple views, and **(c)** multiview registration and classifying the deformed object using the 3D classifier operating on onboard Intel GPU.

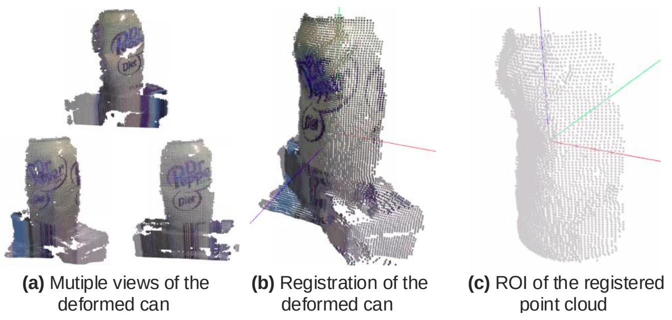

(a) Mutiple views of the deformed can
(b) Registration of the deformed can
(c) ROI of the registered point cloud

Fig. 11: The deployment stage includes **(a)** capturing multiple views of the object, **(b)** multiview point cloud registration, and **(c)** selecting the region of interest of the registered point cloud.

clouds at multiple angles (Fig. 10b). Finally, the Baxter robot classifies the object after taking the region of interest (ROI) of the registration of multiview point clouds (Fig. 10c).

2) Capturing Point Cloud at Multiple Views: To take multiple views of the object, we first set the RGB-D camera with minimum and maximum depth thresholds to focus on the grasping object and to avoid capturing non-interested background. Next, we rotate the wrist joint, W_2, of the robot's arm by different angles. The process occurs on two computers, including one controlling the robot's arm and the main computer, with synchronization between them. For each angle, once the computer controlling the robot arm receives a motion command from the main computer, it executes motion planning and notifies the main computer when the motion command is done. To this point, the main computer captures a point cloud from the RGB-D camera and saves it into an array. The process is repeated for other angles, resulting in an array of point clouds captured at multiple views (Fig. 11a).

3) Mutiview Point Cloud Registration: After capturing multiview point clouds of the grasping object, we register them using the point-to-point Iterative Closest Point (ICP) algorithm [33] to reconstruct a partial view of the object (Fig. 11b).

4) Region of Interest of Registered Point Cloud: As the registered point cloud may contain non-interested objects, we select the ROI from the registered point cloud. To solve this, we leverage the relative position of the wrist joint, W_2, of the robot's arm to get rid of the lower part in the point cloud. The ROI of the registered point cloud is shown in Fig. 11c.

5) Object Classification using OpenVINO: Since our goal is to run our classifier without any dedicated NVIDIA GPUs, we leverage the existing onboard Intel GPU using the OpenVINO library. Herein, the ROI of the registered point cloud at the previous step serves as the input of the classifier operating on the onboard GPU of the Intel NUC5i7RYH PC.

6) Time Complexity on Conventional Hardware: The inference time for the 3D classifier on the main computer varies between 18 and 22 milliseconds (ms), depending on the number of sampling points from the ROI of the registered point cloud that the classifier takes in as input.

D. Demonstration

The demonstration video takes the scenario of the Baxter robot (1) grasps a deformed soda can on its gripper, (2) captures multiple views of the soda can by rotating its wrist joint to multiple angles, (3) performs the pre-processing procedure (Sec. VI-C), and (4) recognizes the deformed object by utilizing the 3D deformable object classifier running on the onboard Intel GPU: https://youtu.be/qkgi3T6xYzI.

VII. Limitations & Future Works

The main limitation of the current robotic system is that the robot does not grasp deformed objects with sufficient force. When the grasp does not guarantee force closure, the object in the robot's gripper may fall off its hand when the robot is moving around for service tasks. Additionally, in the scope of this work, the elasticities of the target objects are not learned by the robot. Therefore, we reserve the tasks of learning the elasticities of 3D deformable objects and analyzing multifingered hand kinematics for future work.

VIII. Conclusions

This work presents hand-eye coordination for deformable object manipulation on assistive mobile cobots by introducing an auto-generated 3D deformable object dataset and a 3D deformable object classifier. The dataset is generated from scans of real-world objects taken from the MakerBot Digitizer 3D scanner using an intuitive Laplacian-based mesh deformation procedure with smooth regularization. Meanwhile, the classifiers are designed specifically for deformable object classification and evaluated with different design choices. We test our classifiers on the generated dataset on the Baxter robot with two 7-DOF arms and a mounted Intel RealSense D435i RGB-D camera with real-world objects. The result shows that the robot can still recognize the deformable object on its hand.

References

[1] R. Magnet, J. Ren, O. Sorkine-Hornung, and M. Ovsjanikov, “Smooth non-rigid shape matching via effective dirichlet energy optimization,” *arXiv preprint arXiv:2210.02870*, 2022.

[2] F. Bogo, J. Romero, M. Loper, and M. J. Black, “Faust: Dataset and evaluation for 3d mesh registration,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 3794–3801.

[3] A. M. Bronstein, M. M. Bronstein, and R. Kimmel, *Numerical geometry of non-rigid shapes*. Springer Science & Business Media, 2008.

[4] C. Kampouris, I. Mariolis, G. Peleka, E. Skartados, A. Kargakos, D. Triantafyllou, and S. Malassiotis, “Multi-sensorial and explorative recognition of garments and their material properties in unconstrained environment,” in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 1656–1663.

[5] D. Seita, N. Jamali, M. Laskey, A. K. Tanwani, R. Berenstein, P. Baskaran, S. Iba, J. Canny, and K. Goldberg, “Deep transfer learning of pick points on fabric for robot bed-making,” in *Robotics Research: The 19th International Symposium ISRR*. Springer, 2022, pp. 275–290.

[6] A. Doumanoglou, J. Stria, G. Peleka, I. Mariolis, V. Petrik, A. Kargakos, L. Wagner, V. Hlaváč, T.-K. Kim, and S. Malassiotis, “Folding clothes autonomously: A complete pipeline,” *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1461–1478, 2016.

[7] M. Zuern, M. Wnuk, A. Schneider, A. Lechler, and A. Verl, “Localization and tracking of deformable linear objects with self organizing maps,” in *ISR Europe 2022; 54th International Symposium on Robotics*. VDE, 2022, pp. 1–9.

[8] O. Sorkine and M. Alexa, “As-rigid-as-possible surface modeling,” in *Symposium on Geometry processing*, vol. 4, 2007, pp. 109–116.

[9] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel, “Laplacian surface editing,” in *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 2004, pp. 175–184.

[10] O. Sorkine, “Laplacian mesh processing,” *Eurographics (State of the Art Reports)*, vol. 4, no. 4, 2005.

[11] A. Jacobson, I. Baran, J. Popovic, and O. Sorkine, “Bounded biharmonic weights for real-time deformation.” *ACM Trans. Graph.*, vol. 30, no. 4, p. 78, 2011.

[12] A. Kurenkov, J. Ji, A. Garg, V. Mehta, J. Gwak, C. Choy, and S. Savarese, “Deformnet: Free-form deformation network for 3d shape reconstruction from a single image,” in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 858–866.

[13] D. Jack, J. K. Pontes, S. Sridharan, C. Fookes, S. Shirazi, F. Maire, and A. Eriksson, “Learning free-form deformations for 3d object reconstruction,” in *Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part II 14*. Springer, 2019, pp. 317–333.

[14] W. Wang, D. Ceylan, R. Mech, and U. Neumann, “3dn: 3d deformation network,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1038–1046.

[15] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry, “3d-coded: 3d correspondences by deep deformation,” in *Proceedings of the european conference on computer vision (ECCV)*, 2018, pp. 230–246.

[16] M. E. Yumer and N. J. Mitra, “Learning semantic deformation flows with 3d convolutional networks,” in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VI*. Springer, 2016, pp. 294–311.

[17] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, “Multi-view convolutional neural networks for 3d shape recognition,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 945–953.

[18] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, “Multi-view 3d object detection network for autonomous driving,” in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 1907–1915.

[19] D. Maturana and S. Scherer, “Voxnet: A 3d convolutional neural network for real-time object recognition,” in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2015, pp. 922–928.

[20] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, “Pointpillars: Fast encoders for object detection from point clouds,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 12 697–12 705.

[21] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[22] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, “Point transformer,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 16 259–16 268.

[23] K. M. Svore, A. V. Aho, A. W. Cross, I. Chuang, and I. L. Markov, “A layered software architecture for quantum computing design tools,” *Computer*, vol. 39, no. 1, pp. 74–83, 2006.

[24] T. Dang, K. Nguyen, and M. Huber, “Perfc: An efficient 2d and 3d perception software-hardware framework for mobile cobot,” in *The International FLAIRS Conference Proceedings*, vol. 36, 2023.

[25] ——, “Extperfc: An efficient 2d and 3d perception hardware-software framework for mobile cobot,” *arXiv preprint arXiv:2306.04853*, 2023.

[26] T. Dang, T. Tran, K. Nguyen, T. Pham, N. Pham, T. Vu, and P. Nguyen, “iotree: a battery-free wearable system with biocompatible sensors for continuous tree health monitoring,” in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, 2022, pp. 769–771.

[27] M. P. Do Carmo, *Differential geometry of curves and surfaces: revised and updated second edition*. Courier Dover Publications, 2016.

[28] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr, “Discrete differential-geometry operators for triangulated 2-manifolds,” in *Visualization and mathematics III*. Springer, 2003, pp. 35–57.

[29] M. Kazhdan, M. Bolitho, and H. Hoppe, “Poisson surface reconstruction,” in *Proceedings of the fourth Eurographics symposium on Geometry processing*, vol. 7, 2006, p. 0.

[30] H.-S. Fang, C. Wang, M. Gou, and C. Lu, “Graspnet-1billion: A large-scale benchmark for general object grasping,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 444–11 453.

[31] R. Wang, J. Zhang, J. Chen, Y. Xu, P. Li, T. Liu, and H. Wang, “Dexgraspnet: A large-scale robotic dexterous grasp dataset for general objects based on simulation,” *arXiv preprint arXiv:2210.02697*, 2022.

[32] M. Jaderberg, K. Simonyan, A. Zisserman *et al.*, “Spatial transformer networks,” *Advances in neural information processing systems*, vol. 28, 2015.

[33] P. J. Besl and N. D. McKay, “Method for registration of 3-d shapes,” in *Sensor fusion IV: control paradigms and data structures*, vol. 1611. Spie, 1992, pp. 586–606.