2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Acquiring Mechanical Knowledge from 3D Point Clouds

Zijia Li, Kei Okada and Masayuki Inaba

Abstract— **We consider the problem of acquiring mechanical knowledge through visual cues to help robots use objects in new situations. In this work, we propose a novel deep learning approach that allows a robot to acquire mechanical knowledge from 3D point clouds. This presents two main challenges. The first challenge is that a robot needs to infer novel objects' functions from its experience. Secondly, the robot should also need to know how to manipulate these novel objects. To solve these problems, we present a two-branch deep neural network. The first branch detects function parts from the point clouds while the second branch predicts offset poses. Fusing the results from these two branches, our approach can not only detect what functions the novel objects may have but also generate key object states which can be used to guide a robot to manipulate these objects. We show that even though most of the training samples are synthetic data, our model still learns useful features and outputs proper results. Finally, we evaluate our approach on a real robot to run a series of tasks. The experimental results show that our approach has the capability to transfer mechanical knowledge in new situations.**

I. INTRODUCTION

Through daily experiences and intuition, humans can easily recognize various kinds of objects and their functions at first glance. For example, a button is associated with the action press and a door with handle implies that the handle can be used to open the door. This kind of intuition is known as **affordance** in the psychology field, developed by Gibson [1]. The concept of affordance is widely implemented in product and web design to improve user experience. In terms of robots, this knowledge allows them to perform manipulation tasks in unknown situations.

Since the idea presented by Gibson is abstract, modeling object affordances in the real world becomes a difficult problem. In robotics, many previous works treat affordance as a property of an object [2], similar to color, shape, etc. Furthermore, different parts of an object may have different affordances [3, 4, 5, 6]. For example, a hammer's handle affords "*grasping*" and the hammerhead affords "*striking*". However, in neuroscience, Osiurak et al. [7] argue that the term "affordance" should be redefined in a more precise way, otherwise it will become progressively useless and eventually meaning everything and its opposite. According to their definition, affordances are animal-relative, hand-centered properties. For example, a cup affords push-ability, grasp-ability or throw-ability, while actions such as pouring liquid into a cup and using a cup to support an apple are not driven by affordance, because these skills are tool-centered and human acquires these skills by interacting with environmental objects. The authors in [7] call these skills as **mechanical actions**, and the relevant knowledge is called **mechanical knowledge**.

Z. Li, K. Okada and M. Inaba are with Department of Mechano-Infomatics, The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-8656, Japan; zijia@jsk.t.u-tokyo.ac.jp

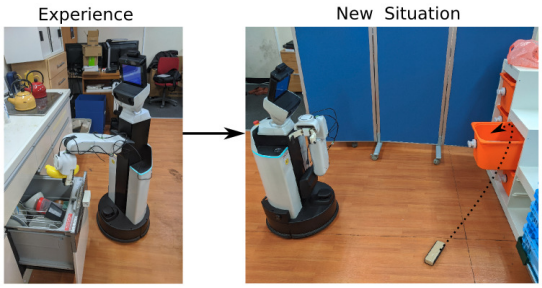

Fig. 1. An example of transferring mechanical knowledge to a new situation.

The capability of acquiring mechanical knowledge from the environment is necessary for robots to handle new situations. For example, if a robot has the experience of opening a kitchen drawer and putting things into the drawer, the robot should also know how to use other drawers to store things without further human demonstrations (Fig. 1).

We consider mechanical knowledge should contain information about: (1) what function an object may have; (2) which part of the object has this function; (3) how to use the object to achieve this function. In this work, we present a two-branch deep neural network for acquiring mechanical knowledge from training data and adapting this knowledge to novel scenarios. Our network takes 3D point clouds as inputs, the outputs of the network are point labels and key object states. The point labels contain information about what functions the input point clouds have and where the functions locate, while the object states are used to generate motion trajectories for robotic manipulation.

In this paper, our primary contributions are as follows:

- We present a two-branch deep neural network for mechanical knowledge acquisition from object pair 3D point clouds.
- We show that our network is able to learn useful features even though most of the training samples are synthetic data.
- We show how to use a pre-trained model to learn a new task with a few demonstration data.

II. RELATED WORKS

The concept of affordance is considered as a key for developing intelligent agents [8] because it reveals how an animal may interact with the environment. In some researches, affordance is viewed as a hint to reveal object

functions [3, 4, 5, 6]. For example, a knife's handle affords "grasping" and the knife blade affords "cutting". However, Osiurak et al. [7] pointed out that the concept of affordance is generating confusion due to different interpretations. They formulated a more conservative definition of affordance and proposed the concept of mechanical knowledge to describe how tools and objects can be used together mechanically. When interacting with new tools, humans can use mechanical knowledge to infer a potential utilization by reasoning the physical properties of the tool. Therefore, an intelligent agent should also be able to use mechanical knowledge to detect novel tools' functions by recognizing their visual features such as shapes and geometries.

Detecting objects' visual features for robotic manipulation has been studied for a long time and many insightful approaches have been proposed. In robotic grasp detection, Jiang et al. [10] used the support vector machine (SVM) to learn hand-crafted features and applied the trained model to detect grasping rectangles on 2D images. As deep learning methods become popular in the past several years, authors in [11, 12, 13] replaced the SVM with deep neural networks to learn visual features for grasping rectangle detection. In object function detection or affordance detection, Nguyen et al. [9] presented a method to identify object affordances via a deep convolutional neural network, their model successfully identified object affordance at the pixel level. Similarly, [5] and [6] improved the deep neural network structure and reached a higher accuracy. Although these approaches achieved success in detecting what functions the input image contains and where these functions locate, the outputs of these approaches do not tell a robot how to do.

On the contrary, deep reinforcement learning methods can tell robots how to manipulate objects by outputting robotic control signals such as joint angles. Levine et al. [14] used a convolution neural network to learn visuomotor policies. Their network takes RGB images as input and output motor torques on a real robot, According to the results, their model enables a real robot to finish some complex tasks such as hanging a coat hanger on clothes rack, screwing a cap onto a bottle, etc. Matas et al. [15] achieved the goal of teaching a robot to manipulate deformable objects. They employed domain randomization in a simulation environment to train a deep learning agent and successfully transferred the policy from simulation to the real world. However, deep reinforcement learning methods can not explicitly provide information about the objects' functions because the learned visual features are encoded in the network. Furthermore, when implementing reinforcement learning methods, users are difficult to know about the goal at the beginning, due to the fact that deep reinforcement learning methods only output one step in each time.

Beyond these methods, we propose a mechanical knowledge acquisition network. Given object pair point cloud inputs, our network is able to output information of mechanical knowledge, including what functions the point clouds have, which parts contain these functions, and how to achieve these functions.

III. Acquiring Mechanical Knowledge from 3D Point Clouds

A. Problem Statement

Given an input 3D point cloud that contains an activated object $\mathcal{O}_1$, its associated object $\mathcal{O}_2$, our network aims at acquiring mechanical knowledge via visual features. The network has two outputs, a classification output that predicts every point's function label; a regression output that consists of multiply states (waypoints) $\{\mathcal{S}_s, \mathcal{S}_{m1}, ...\mathcal{S}_g\}$ to form a motion trajectory, where $\mathcal{S}_s$ is the start state, $\mathcal{S}_g$ is the goal state and others are middle states, each state contains 6D pose information. According to the detected function, a trajectory may represent the states of either the activated object or the associated object. For example, given a point cloud that contains a plate (activated) and a cup, the desired function of this object pair is "support" and the corresponding trajectory represents the cup's states. On the other hand, if a point cloud contains a hammer (activated) and a walnut, the desired function of this object pair is "pound" and the corresponding trajectory represents the hammer's states. On the contrary, if the walnut is activated, we may want the network outputs "no function" because a walnut can not smash a hammer. Besides, the network also accepts one object as input, in this case, the output function is always "grasp" and all the output states are the same, which represent one potential grasp pose of that object.

B. Two-Branch Network Architecture

Our deep neural network is built upon the deep learning architecture described in [16, 17]. Specifically, two main components allow our network to handle unordered point data. The first one is the Set Abstraction module, which is responsible for encoding hierarchical point set features. a complete Set Abstraction module is composed of a sampling layer, a grouping layer, and a PointNet layer. The sampling layer is designed to gather a subset from an input point cloud using the iterative farthest point sampling algorithm, elements in the subset are considered as centroids of local regions. The function of the grouping layer is to construct local region sets by searching neighboring points around the centroids. The PointNet layer then encodes these local region patterns into feature vectors. The second component is the feature propagation module, similar to the transposed convolution layer in CNNs, this component can be seen as a decoder network, and it achieves the feature propagation function by interpolating feature values.

An overview of our network is shown in Fig. 2. The network takes N$\times$(3+1) matrixes as input, where N is the number of points in the input point cloud, each point has a 3-dimensional value (x, y, z) plus an extra activated channel. The activated channel indicates whether a point belongs to the activated object, we set this value to one for points from the activated object, otherwise set to zero. The feature encoding part consists of three set abstraction modules. In the first set abstraction module, we set the number of points in the sampling layer (npoint) to 512, the number of

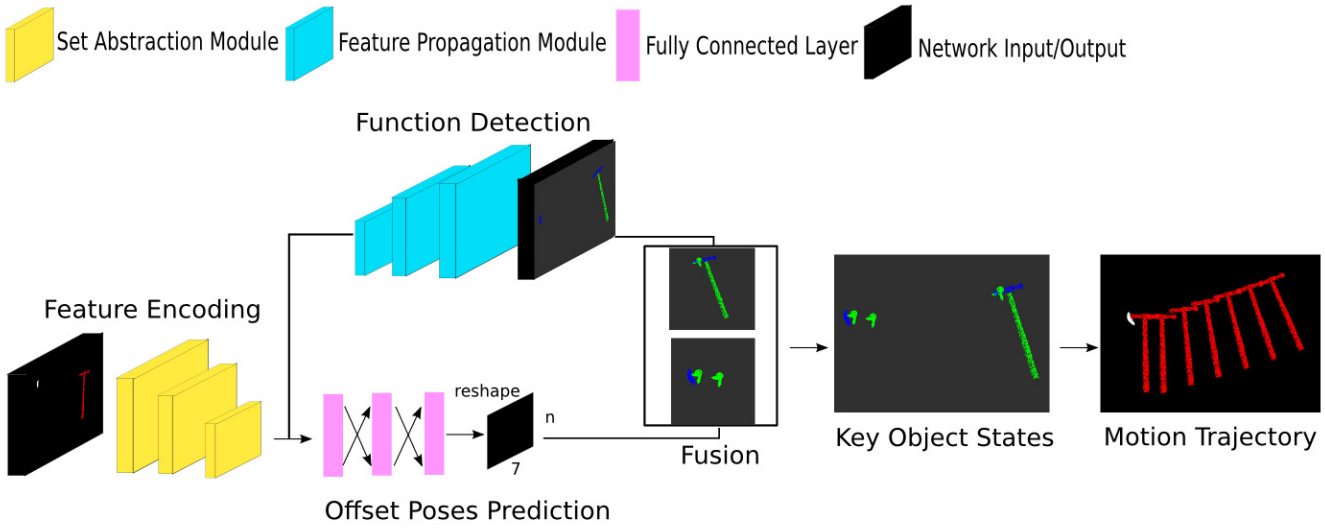

Fig. 2. An overview of our mechanical knowledge acquisition network.

neighboring points in the grouping layer (nsample) to 32 with the radius of 0.02, and the neurons of multi-layer perceptron (MLP) network inside the PointNet layer are set to 32, 32, 64 separately. The second set abstraction module has the parameters (npoint = 256, radius = 0.05, nsample = 64, MLP = [64, 64, 128]), and the parameters in the third one are set to (npoint = None, radius = None, nsample = None, MLP = [128, 256, 512]).

The function detection branch is composed of three feature propagation modules, the MLP parameters that we use in each layer are [128, 128], [128, 64], [64, 64, 64] separately. After that is a 1D convolutional layer which has 128 neurons and a dropout layer with the dropout rate of 0.5, the last layer is another 1D convolutional layer which outputs a tensor with the shape (2400, 5), where 2400 is the number of points and 5 is the number of classes ("no function", "grasp", "pour", "pound" and "support"). The second branch consists of three fully connected layers to predict offset poses, the numbers of neurons in these layers are [256, 128, 3×7], where 7 represents an offset pose (3 values for position and 4 values for orientation in quaternion form), 3 is the number of poses. We can use the following equation to recover the offset poses to the original poses:

$$P_{original}(x, y, z, qx, qy, qz, w) = P_{offset}(x + c_x, y + c_y, z + c_z, qx, qy, qz, w) \quad (1)$$

$$C(x, y, z) = \frac{\sum_i^N Point_i(x, y, z)}{N} \quad (2)$$

Where N is the number of points in the corresponding point set, $C(x, y, z)$ is the centroid of the point set. Using the offset position instead of original position can be viewed as a kind of normalization that helps the network learn better.

C. Fusing Network Outputs

The fusion module will not be trained because it is a logic module. According to the function detection output, the fusion module will decide which object should be manipulated and recover the offset poses to generate a motion trajectory. For example, the output function in Fig. 2 is "pounding" because the hammer is activated, which means a robot can use the hammer to pound the apple. Therefore the fusion module assigns the start state on the hammer and the rest states on the apple, notice that the position of the start state is calculated by adding up the position of the first offset pose and the centroid of the hammer's function part (the blue part), similarly, the second and third states are calculated by adding up the rest offset poses and the centroid of the apple. When there is only one activated object in the input point cloud, the output function is "grasp". The grasp pose is calculated by adding up the first offset pose and the centroid of the corresponding point set.

D. Network Training

During the training phase, we apply the sparse softmax cross entropy to measure the classification loss, while the regression loss is measured by the mean squared error, the total loss is given by 0.9 * classification loss + 0.1 * regression loss because the classification loss is much larger than the regression loss. The training label for offset pose is computed by subtracting the corresponding part centroids from the state labels, and the training label for function detection is a (1, 2400) tensor which indicates each point's class. The number of epochs is set to 200. In each epoch, we randomly shuffle the training set then loop over all the training data with the batch size of 32. We use Adam optimizer with an initial learning rate of 0.001 and decay exponentially by a factor of 0.7. It takes us about 5 hours to train the network on an NVIDIA GTX 1080Ti GPU until convergence.

E. Training Data Collection, Augmentation and Annotation

Since our goal is helping robots manipulate tools in real scenes, the objects that we use to train our network ought to

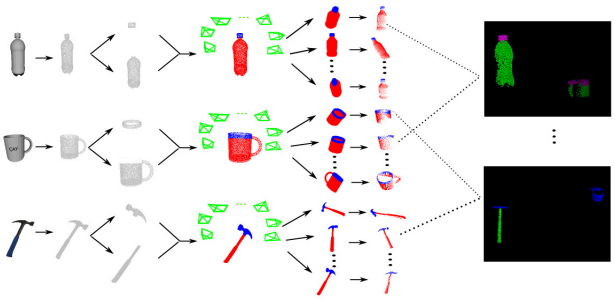

Fig. 3. A demonstration of how to generate training data from CAD models.

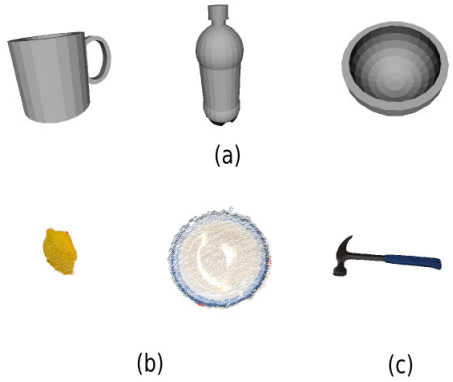

Fig. 4. Example of the objects in our training dataset. (a) objects from the ModelNet40 dataset [19]. (b) objects from the RGB-D object dataset [18]. (c) objects that we collected from the Internet.

be commonly seen in daily life and manipulatable by general robots. Utilizing the existing resources, we have found an RGB-D object dataset [18] that provides RGB-D scans of real objects. However, only several categories are suitable for our experiments and the number of samples in each category is small. To train a robust network, we need more categories and samples.

Inspired by the work presented in [20], we introduced the CAD models from the ModelNet40 dataset [19] to augment the training dataset. Generally, most of the models that we obtain from 3D depth sensors are partly occluded, but CAD models from the ModelNet40 dataset are fully visible. If we train the network using the fully visible models and test it on the partially occluded models, the network's performance will become very poor. To solve this problem, we generated partly occluded point clouds from fully visible point clouds via different viewpoints, then used these partly occluded point clouds to train our network. The concrete steps are as follows:

1) Firstly, we sample 100000 points uniformly from a CAD model's mesh surface.
2) The sampled point cloud will then be scaled to real object size and translated to zero-centered.
3) We assign labels to every point in the point cloud by segmenting function parts of the object.
4) We put virtual cameras on various viewpoints to generate partly occluded point clouds from a fully visible point cloud using the z-buffer algorithm [21] and the OpenGL library. For every fully visible point cloud, we generate 10 partly occluded point cloud samples.

A demonstration is shown in Fig. 3, notice that invisible points are removed after step 4. On the other hand, samples from the RGB-D object dataset are partly occluded point clouds, these samples will not be processed, we also pick 10 samples from every instance.

In particular, the CAD models of bottles, bowls, cups and some plates (which are mixed in the "bowl" class) are from the ModelNet40, the scenes of plates and foods (lemon, orange, peach, pear, potato, tomato) are from the RGB-D object dataset. Besides, we also collect CAD models of hammers from the internet. Some examples are shown in Fig. 4. The bottle, bowl and cup class in our training set contain 30 instances separately, the plate, hammer, and food class contain 15, 8 and 33 instances separately, the total number of instances in our dataset is 146 and the number of samples is 1,460. We then randomly pick three instances from each category to build a validation set and keep the remains for training.

Given these objects, we have five function classes: "no function", "grasp", "pour", "pound" and "support". Each time we randomly pick two samples from the corresponding object classes to generate an object pair point cloud. Specifically, the objects in the "grasp" class are bottles, cups, hammers, and food samples. The object pairs in the "pour" class are (bottle, cup), (bottle, bowl), and (cup, bowl). The object pairs in the "pound" class are (hammer, cup), (hammer, bottle), (hammer, bowl), and (hammer, food). The object pairs in the "support" class are (plate, cup), (plate, bottle), and (plate, food). The object pairs in the "no function" class are (other objects, hammer), (other objects, plate). Here the first object is the activated object and the second object is the associated object. Notice that in the "pour" class, only points in the mouth part of the activated object and associated object have the "pour" label, and the label of other points are set to "no function". This means the robot should focus on the mouth part of the two objects when performing the pouring task. Similarly, we put the "pound" label on the hammerhead and the whole associated object to present that the robot can pound anywhere of the associated object.

When processing state labels, we first sort out an object category's corresponding scenes, for example, the food class may appear in the "pound", "support" and "no function" scenes, then we manually attach the corresponding states separately for all food samples. Besides, state labels in the "no function" class are set to zeros. Finally, we combine these states when generating object pair point clouds. For functions such as "pour", we also consider whether the other object locates on the left-hand side or right-hand side, therefore we create "pouring from left" and "pouring from right" states (see Fig. 5 (a)). Since there are countless valid trajectories for each function, to get better performance, we

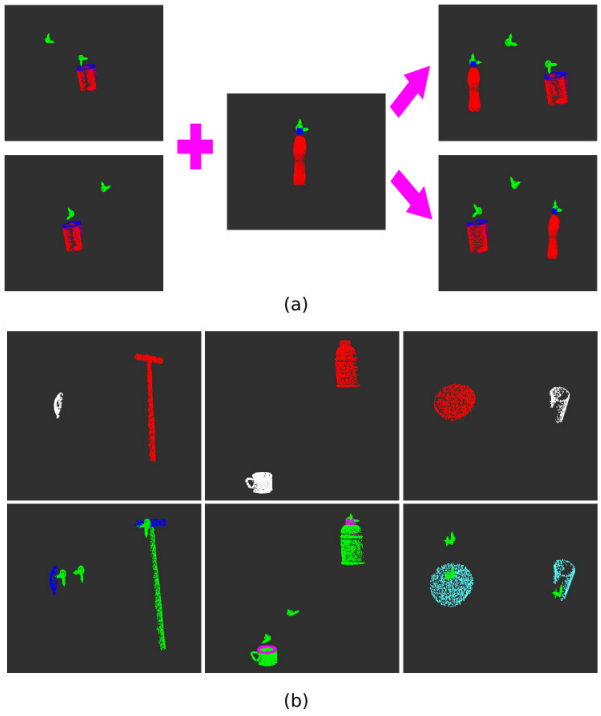

Fig. 5. (a) A demonstration of how to generate state labels. (b) Example of point clouds and labels in our training dataset. Top: The object in red color is the activated object and the object in white color is the associated object. Bottom: Blue points have the "pound" label, magenta points have the "pour" label, sky blue points have the "support" label and green points have the "no function" label.

keep the trajectory similar to each other in the same function class.

In order to augment the dataset, when generating point clouds, we randomly assign a translation factor between [-1, 1] and a scale factor between [0.8, 1.2] to each object to change their positions and scales. To avoid overlapping, we compute the distance between the centroid of two objects, for a distance less than 0.2 (or 20cm), we will reassign the translation factor. Finally, we resize the point clouds to the size of 2,400 through uniform sampling (for point clouds that have points less than 2,400, we pad them with zeros). For each object pair mentioned above, we generate 500 point clouds for training and 50 point clouds for validation (some examples are shown in Fig. 5 (b)). The total number of point clouds for training is 12,000 (2,000 for "grasp", 1,500 for "pour", 2,000 for "pound", 1,500 for "support" and 5,000 for "no function") and the number of point clouds for validation is 1,200.

IV. Experiments

Through our experiments, we seek to answer the following questions: 1) Given the synthetic data, can our approach effectively learn useful features? 2) How well does the trained model adapt the knowledge to novel objects? 3) How well does a robot run manipulation tasks following the generated trajectories? To answer these questions, we conduct a series of experiments on a TOYOTA's HSR robot [22].

Fig. 6. Objects used in our test set and robotic experiments.

A. Pouring, Pounding, and Placing

Based on our training data, we consider three different tasks: pouring, pounding, and placing. During the evaluation, we used the following metrics to define success for each task: success if the robot grasps an activated object and the mouth part of the activated object is put within the mouth part of the associated object for the task pouring; success if the robot grasps a hammer and uses the hammerhead to hit the associated object for the task pounding; success if the robot grasps the associated object and places it on the plate for the task placing.

B. Test Data Collection

In order to evaluate the performance of our approach, we prepared five bottles, five bowls, five cups, five plates, two hammers, and three plastic food models, so the number of test objects is 25 (see Fig. 6). We captured five samples for every object through the HSR robot's head-mounted RGB-D camera, thus the number of samples is 25 * 5 = 125. During the collection, we put the objects in different poses and scanned them from different viewpoints to avoid repetition. Similar to the training set creation process, we randomly selected two samples from two object classes to generate a test point cloud. Finally, we generated 400 point clouds for the 'grasp' class, 300 point clouds for the "pour" class, 400 point clouds for the "pound" class, 300 point clouds for the "support" class, and 1,000 point clouds for the "no function" class. Therefore, the size of our test set is 2,400.

C. Results

Table I shows the results of using different point cloud types as training data. The fully visible point clouds were generated by sampling points from the CAD models directly then combine different samples to generate object pair point clouds. The results clearly show that using the processed partly occluded point clouds as training data significantly improves the system performance.

In Table II, we apply the Mean Absolute Error (MAE) to measure the error between the positions of predicted states and the positions of ground truth states. Notice that we add state labels to the test set in the same way as what we do to

TABLE I

FUNCITON CLASSIFICATION ACCURACY WHEN USING DIFFERENT POINT CLOUD TYPES AS TRAINING DATA

	Fully Visible	Partly Occluded
Pour	84.38%	91.29%
Pound	49.22%	93.69%
Support	100.00%	100.00%

TABLE II

COMPARE THE MEAN ABSOLUTE ERROR (CM) BETWEEN DIFFERENT METHODS

	Validation Set		Test Set	
	Offset	Origin	Offset	Origin
Grasping	0.9	2.8	1.6	18.8
Pouring	1.5	3.3	2.9	32.4
Pounding	2.1	3.9	3.1	23.7
Placing	2	3.5	2.7	13.7

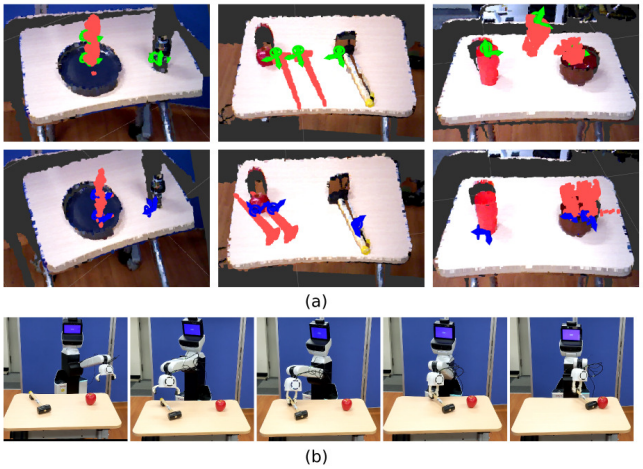

Fig. 7. (a) Examples of output motion trajectories for different tasks using different approaches. Top: Our method. Bottom: The ICP estimation method. (b) The HSR robot is following the trajectory to perform a pounding task.

the training set, although the state labels are not unique, a valid prediction should not deviate far from our annotation. From the results, we can see that benefit from the data normalization, using offset poses as training labels achieves much higher accuracy than using original poses as training labels on the test set.

D. Evaluation on Real Robot

We evaluate our system's performance in real-world scenarios on the HSR robot platform. The robot has an RGB-D camera on its head so that we can obtain point clouds from the environment. To segment target points from the background, a Mask-RCNN model [23] is trained to get the target objects' masks from RGB images, the corresponding depth images are then used to unproject pixels within the masks into 3D space to generate input point clouds, at the same time, activation signals are assigned to corresponding points. Remember that we assume the input point cloud of our model has no more than two objects. When more than two candidates are detected, the system will generate different object pair combinations and send them to our model one by one to get corresponding results. The predicted function class is determined by the number of function points in each class (function classes other than "no function"), the function class with the largest number is the output function class. If the number of function points is less than 100, the object pair is considered no function.

According to the predicted function, the system will decide which object is manipulatable. The manipulatable object will then be separated from the point cloud and sent to the trained model again to predict a grasp pose. In our case, the grasp pose is consistent with the tool frame of the HSR robot's end effector. The system will also calculate a transformation between the grasp pose and the start state, then apply the transformation on other states to get the robot motion trajectory. In this paper, we assume that the test objects are put in free space, therefore we adopt linear interpolation to complete the whole motion trajectory. In more complex situations, users can combine motion planning approaches with state labels to avoid collisions.

During the evaluation, we ran 30 trials for each task. In each trial, object pairs were randomly picked from the test objects. To test how well our method learns the mechanical knowledge, we compare its performance against the ICP estimation method. The ICP estimation method is built upon the ICP (iterative closest point) algorithm [24], which is widely used for estimating the rigid registration of 3D point sets. The implementation of the ICP estimation approach is stated as follows: (1) Loop over all the relevant training samples and apply the ICP algorithm to compute the transformation as well as fitness score (sum of squared distances from the source point cloud to the target point cloud) between the input object point cloud and the training samples. (2) Find out the sample with the lowest fitness score, then apply the transformation to its corresponding state labels to generate a motion trajectory for the task. Table III shows the comparison results, we can see that our approach is much better than the ICP estimation approach when dealing with novel objects. The reason is that the ICP estimation approach requires a training sample that has a similar shape and size with the test object, while our approach can learn the visual and geometry features from the point cloud. The output states (the green and blue markers) of the two approaches are shown in Fig. 7 (a) and Fig. 7 (b) shows how to use the predicted trajectory to guide our robot to perform a pounding task. Since the robot motion trajectory is calculated based on the transformation between the grasp pose and the start state, the difference

TABLE III

RESULTS ON ROBOTIC EXPERIMENTS

	Our	ICP Estimation
Pouring	93.33%	30.0%
Pounding	80.0%	6.7%
Placing	96.67%	83.67%

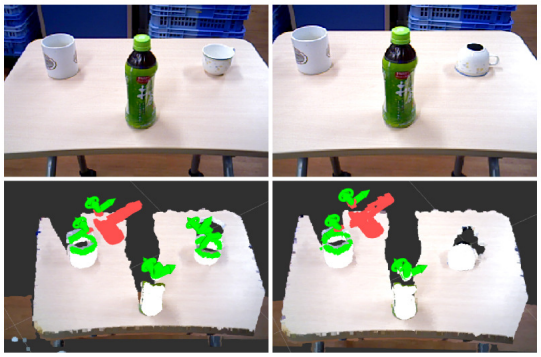

Fig. 8. Left: When there are multiple candidates, our system will predict multiple trajectories and randomly select one trajectory to execute. Right: One of the cups is turned upside-down. Our system detects this cup does not have the pour function in this state. Green points are function points and white points are no function points.

between the real grasp pose and the predicted grasp pose may cause failures. Take the pounding task as an example, during grasping, the hammer's orientation may be changed due to the predicted grasp pose is imperfectly aligned with the hammer's current direction. A large change may lead to the real motion trajectory deviate from the excepted trajectory and cause failures.

Beyond the aforementioned robotic experiments, we also conducted a more complex experiment to verify whether our model learns useful geometry patterns. In this experiment, we first put two different cups and a bottle on the desk to detect the pour function, then we turned one of the cups upside-down and executed the program again to observe the outputs. The experimental results are shown in Fig.8. In this figure, function points are colored in green for visualization. From the figure in the bottom right corner, we can see that points of the upside-down cup are labeled with "no function", which means our model successfully connects the mouth shape pattern to the function "pour". We also tried other cups within the test objects, finally, we got 9 successes in 10 trials.

E. Learning Mechanical Knowledge on New Task

Finally, we implemented our model on a new scenario using a few training data to estimate the scalability of our approach. Specifically, we collected 14 kitchen drawer point clouds (7 samples for closed state and 7 samples for opened state) and 15 banana case point clouds from different viewpoints (Fig. 9 (a)). Then we manually labeled function points and attached the state labels to every scene (grasping the kitchen drawer handle, grasping the banana case, and putting the banana case into the opened state kitchen drawer). Following the aforementioned training process, we generated 50 samples for each scene and used these training data to fine-tune our pre-trained model.

In the test scenario, we prepared some drawers placed in the stair structure, the shape and location of the drawer handle are different from the training object. We also put an eyeglass case on the floor. In this task, the robot is

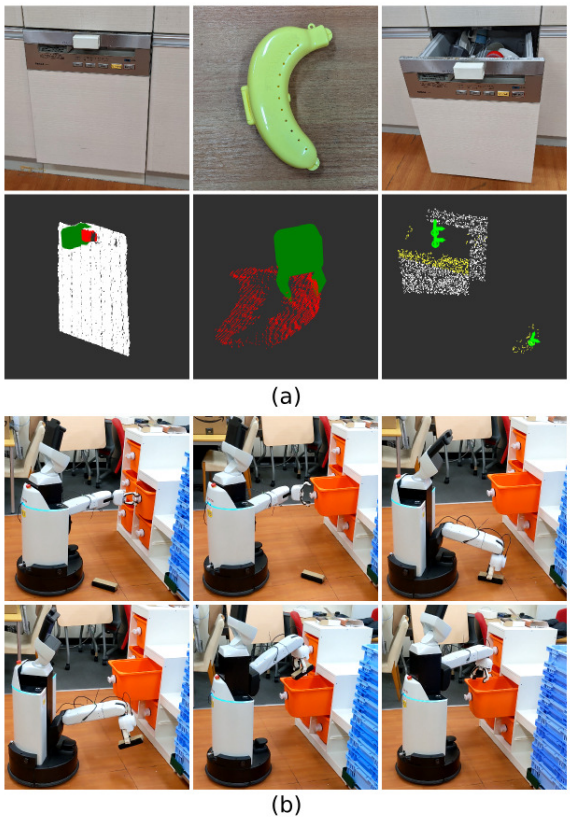

Fig. 9. (a) Top: Training objects for the new task. Bottom: Training data for the new task. Red points have the "grasp" label, white points have the "no function" label and yellow points have the "contain" label. The marker in the first two columns represents a grasping pose and markers in the last column represent object states. (b) The HSR robot is performing a new task by leveraging mechanical knowledge.

required to open a drawer, pick up the eyeglass case, put it into the drawer, and finally close the drawer. To finish this task, our network needs to predict three key actions: grasping the drawer handle to open the drawer; grasping the eyeglass case in random orientation; placing the eyeglass case into the drawer. We also trained a Mask-RCNN to detect the banana case and the drawer (closed state) to obtain the corresponding point cloud. The task starts with detecting grasping pose on a drawer's handle, if the robot successfully opened a drawer, the robot has to go back to the starting point because it only has a single arm. At the same time, the system will record the handle as well as the drawer's location. Given the drawer's point cloud and the robot's moving distance, the system computes a 3D bounding box and use this 3D bounding box to crop the points of the opened drawer. After that, the system puts the opened drawer points and eyeglass case points together to get an input point cloud. This input point cloud is then passed to our model to get a trajectory. Finally, the robot grasps the eyeglass case, puts it into the drawer following this trajectory, and closes the drawer. An example of the task is shown in Fig. 9 (b). We have run 10 trials and got 8 successes. It shows that our network has the ability to learn mechanical knowledge from a few

demonstration data and transfer this knowledge to a novel situation.

V. Conclusion

In this paper, we introduced a novel approach for acquiring mechanical knowledge from 3D point clouds. Different from the previous approaches, our approaches can not only detect object functions and their location but also predict trajectories to guide a robot to perform tasks. Since the available 3D data is lacking, we used the generated partially occluded point clouds to train our deep neural network and tested the trained model on real-world scenes. Experimental results showed that given most of the training data are synthetic data, our model still learned useful features and generated valid motion trajectories. Furthermore, we implemented our system on the HSR robot. Following the predicted trajectory, the robot performed various manipulation tasks at a high success rate. Finally, we conducted a new experiment and the experimental results showed that our approach has the capability of acquiring mechanical knowledge from a few demonstration data and adapting the knowledge to a new situation.

References

[1] J. J. Gibson, *The Ecological Approach to Visual Perception*. classic edition. Psychology Press, 2014.

[2] T. Hermans, J. M. Rehg and A. Bobick, "Affordance prediction via learned object attributes," in *IEEE International Conference on Robotics and Automation (ICRA): Workshop on Semantic Perception, Mapping, and Exploration*, 2011, pp. 181-184.

[3] A. Myers, C. L. Teo, C. Fermuller and Y. Aloimonos, "Affordance detection of tool parts from geometric features," in 2015 *IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 1374-1381.

[4] A. Nguyen, D. Kanoulas, Darwin G. Caldwell and Nikos G. Tsagarakis, "Object-based affordances detection with Convolutional Neural Networks and dense Conditional Random Fields," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2017. pp. 5908-5915.

[5] T.-T. Do, A. Nguyen, I. Reid, D. G. Caldwell and N. G. Tsagarakis, "Affordancenet: An end-to-end deep learning approach for object affordance detection," *arXiv preprint arXiv:1709.07326,* 2017.

[6] K Chaudhary, K Okada, M Inaba and X Chen, "Predicting Part Affordances of Objects Using TwoStream Fully Convolutional Network with Multimodal Inputs," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[7] F. Osiurak, Y. Rossetti and A. Badets, "What is an affordance? 40 years later," in *Neuroscience & Biobehavioral Reviews,* pp. 403-417, 2017.

[8] N. Yamanobe, W. Wan, I G. Ramirez-Alpizar, D. Petit, T. Tsuji, S. Akizuki, M. Hashimoto, K. Nagata and K. Harada, "A brief review of affordance in robotic manipulation research," in *Advance Robotics,* pp. 1086-1101, 2017.

[9] A. Nguyen, D. Kanoulas, D. G. Caldwell and N. G. Tsagarakis, "Detecting object affordances with convolutional neural networks," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, Oct 2016, pp. 2765-2770.

[10] Y. Jiang, S. Moseson and A. Saxena, "Efficient grasping from rgbd images: Learning using a new rectangle representation," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3304-3311, 2011.

[11] I. Lenz, H.Lee and A. Saxena, "Deep learning for detecting robotic grasps," in *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013.

[12] J. Redmon, and A. Angelova, "Real-time grasp detection using convolutional neural networks," in *IEEE International Conference on Robotics and Automation (ICRA).* IEEE, pp. 1316-1322, 2015.

[13] S. Kumra and C. Kanan, "Robotic Grasp Detection using Deep Convolutional Neural Networks" in *Intelligent Robots and Systems (IROS)*. IEEE, pp. 769-776, 2017.

[14] S. Levine, C. Finn, T. Darrell and P. Abbeel, "End-to-End Training of Deep Visuomotor Policies," in *Journal of Machine Learning Research (JMLR)*, 2016.

[15] J. Matas, S. James and A. J. Davison, "Sim-to-Real Reinforcement Learning for Deformable Object Manipulation," in *Conference on Robot Learning (CoRL)*, 2018.

[16] C. R. Qi, L. Yi, H. Su and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," *arXiv preprint arXiv:1706.02413*, 2017.

[17] C. R. Qi, H. Su, K. Mo and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *ArXiv preprint arXiv:1612.00593,* 2016.

[18] K. Lai, L. Bo, X. Ren and D. Fox, "A Large-Scale Hierarchical Multi-View RGB-D Object Dataset," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[19] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pages 19121920, 2015.

[20] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani. C. Anil, T. To, E. Cameracci S. Boochoon and S. Birchfield, "Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization" in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2018.

[21] E. Catmull, "A Subdivision Algorithm for Computer Display of Curved Surfaces," PhD Thesis, Dept of Computer Science, University of Utah, Salt Lake City, Utah, U.S.A., 1974.

[22] T. Yamamoto, K. Terada, A. Ochiai, F. Saito, Y. Asahara and K. Murase, "Development of Human Support Robot as the research platform of a domestic mobile manipulator," in *ROBOMECH Journal*, 2019.

[23] K. He, G. Gkioxari, P. Dollar and R. Girshick, "Mask R-CNN," in *IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[24] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," in *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI),* 1992.