Learn to Load the Dishwasher

Changxi Zheng
Cornell University, cxzheng@cs.cornell.edu

***Abstract*— We develop an algorithm to enable a personal robot to conduct one of the common tasks in our daily life – moving the objects to a specific place. One typical application scenario of this algorithm is to load a variety of kitchen items such as the plates, mugs, and bowls etc. into a dishwasher. For newly seen objects and their destination container (e.g. the dishwahser), our algorithm features a supervised learning procedure to decide both the position to grasp the objects and the locations to place them. The input of the algorithm includes the image captured by a camera and the point cloud data collected by a depth sensor. A few placing positions and the corresponding object's placing orientations are selected by the trained SVM. Finally our method runs the point-based rigid-body simulation for each of the position/orientation candidates to pick the best placing strategy and proposes an placing order if multiple objects are considered.**

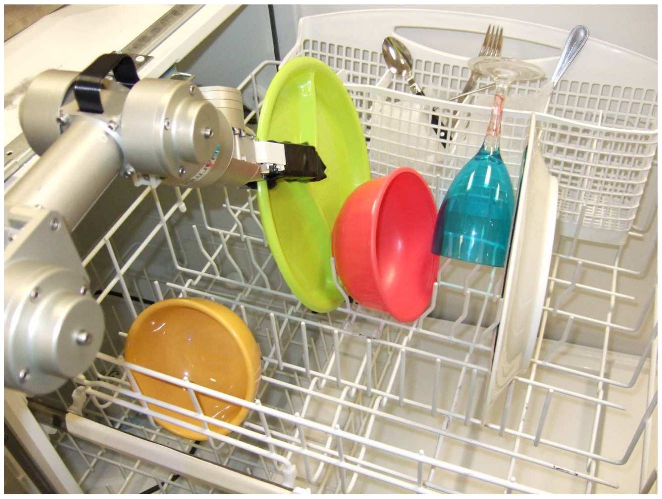


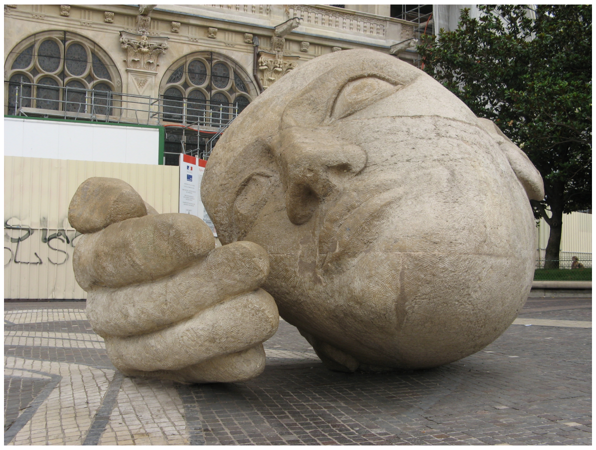
Fig. 1: Load the dishwasher using the robot arm

I. INTRODUCTION

The use of robot is constantly getting improved in recent years to help human beings conduct different kinds of laborious tasks, ranging from assemble complex machinery to balancing a spinning top on the edge of sword [1]. However, most of the robots are programmed or "scripted" for very specific circumstances, such as moving a particular shape of object at particular location to a particular destination. They may run afoul of the tasks or make fatal mistakes if their working context changes. For example, placing the robot in a different environment might fail its jobs. Unfortunately, for personal robot, the uncertainty about the environment is almost inevitable. For instance, consider a personal robot designed for cleaning up the kitchen. It is hard, if not impossible, for the robot to know the layout of the kitchen a priori. Therefore, a versatile personal robot usually requires the learning ability to adapt itself to different unpredictable working situations.

In this project, we focus on developing a learning algorithm for a personal robot to move a variety of objects to specific places. One typical application scenario is to load different shapes of kitchen items such as the plates, mugs, wine glasses and bowls etc. into a dishwasher. Our algorithm does not require the knowledge about the moved objects (e.g. the kitchen items) or the destination place (e.g. the dishwasher). Instead, it features a supervised learning algorithm to decide both the grasping position on the objects and the placing locations. The selected moving strategies are finally validated using a point-based rigid-body simulation before applying to the robot's arm.

To perceive the geometries of both the moved objects and placing destination, our algorithm combines the inputs from a camera and a depth sensor both mounted on the robot. The camera provides captured images for the moved objects (See figure (2.a)). With these images, the grasping points of the objects can be determined using the algorithm proposed in the previous work [2]. The depth sensor collects a "point cloud" data corresponding to the 3D locations that it has found on the front unoccluded surfaces of the objects


(a) Example of camera captured image

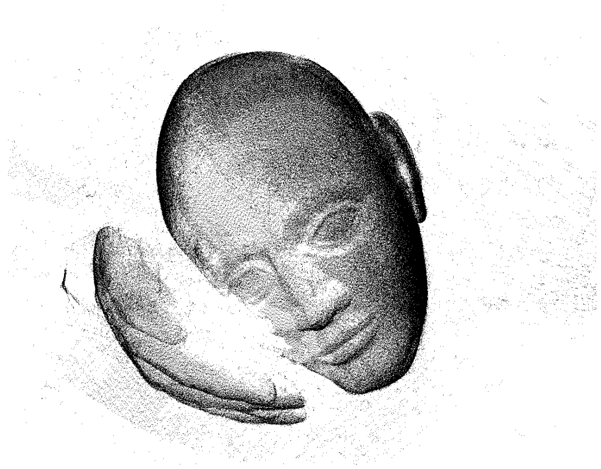

(b) Example of 3D point cloud data

Fig. 2: The example of input data

(See figure (2.b)). Such point clouds are typically incomplete to represent the whole geometry. In order to obtain enough point cloud data, the robot needs scan the object from different view points and employ the registration algorithm such as the *Iterative Closest Point* (ICP) [3], [4], [5] method to align all the input point clouds.

We use the SVM to suggest a loading strategy, which is described by the *placing point* on the destination place and the *placing orientation* of the manipulated object. The placing point is defined as a point where the robot is trying to touch with the grasped object. The placing orientation specifies the orientation of the object when it touches the placing point. Once a moving strategy is specified, the robot can move the object to the pacing point with the given object orientation using any path planning algorithm. After that, the object is released by the robot, experiencing some possible rigid motion due to the gravity or the collisions with the other objects, and finally resting on some place.

Before realizing the proposed moving strategy by the robot, it checks the its feasibility using a rigid body simulation. This could avoid some fatal mistakes (e.g. breaking the dinner plate when loading it into the dishwasher) due to an inappropriate loading decision. However, reconstructing the full 3D models for rigid body simulation is expensive and not robust due to the incompleteness and the noise of the input information. To circumvent this difficulty, we notice that if some place is suitable for holding the object, it is also very likely to be able to hold another geometrically similar object. If the geometric model of the later object is easy to achieve, it can serve as a fitness checker for the original object to test whether or not a particular position is adequate to hold it. Therefore, we first create a database of different shapes of potentially manipulated objects. Our algorithm will match the input geometry with the similar object from the database. For example, in our loading-dishwasher example, the grasped kitchen items are represented by a similar object from a database of kitchen items for which we know the exact 3D model. And that 3D model is used for later fitness checking as well as generating the training examples. We will describe all those details in section IV.

II. Related Work

While there are plenty of previous work about robot manipulation, many of them assumes availability of the 3D model of the manipulated objects. A general survey about this topic can be found in [6], [7]. If the desired location of grasp is available, techniques such as visual servoing that align the gripper to the desired location [8] or haptic feedback [9] can be used to pick up the object. If no grasping point is given, some learning algorithm could be useful to select it. For 2D object, [10], [11] proposed an algorithm to select a 2D position for three-fingered grasps of planar objects. [12] showed that there is a dissociation between recognizing objects and grasping them, i.e., there are seperated neural pathways that recognize objects and that direct spatial control to reach and grasp the object. For 3D object grasping, many of the previous work focus on inferring useful information about objects without 3D model available. [13], [14] showed that given just a single image, it could be possible to extract the (partial) 3D structure of a scene. Later in [2], Saxena and his colleagues proposed a supervised learning algorithm to determine the grasping point from captured images. Utilizing both the camera captured images and point clouds from depth sensor was realized in [15].

III. Robot

The algorithm can be performed on the robot with an arm and other equipment such as cameras, depth sensor and computers. The robot arm could be a position-controlled 5-dof or 7-dof arm with a three-fingered gripper. The depth sensor could be the SwissRanger camera which returns a 144×176 array of depth estimates, the Bumblebee stereo camera with 640×480 array of depth information or a Laser scanner.

IV. Algorithm

In this section, we describe the algorithm details that we have proposed or implemented. We postpone the discussion of the algorithm for the open problems until section VI.

A. Perceive the Geometries

Robot needs to recognize the objects and the container using its vision system. We use the depth sensor to collect the point cloud data representing the objects. Since the point cloud collected from a single sensor scanning is incomplete to recover the full geometry, we need to scan the object from different view points. Therefore an registration algorithm is necessary to align all the input point clouds. The standard point cloud registration algorithm, ICP [3], [4], [5], can be used for this purpose.

However, reconstructing the triangle meshes of the objects from scanned point cloud data is hard, if not impossible. This is because multiple objects might occlude with each other such that there is no way to obtain the completed point cloud data.

We propose to avoid reconstructing the triangle meshes of the objects explicitly by pre-computing a database of different objects and matching the partially scanned data with the database objects to find the closest one. For the application of loading a dishwasher, for instance, the database consists of the triangle meshes of different kitchen items, such as the plates, coffee mugs, and wineglass, etc. The scanned data for a dinner plate should have close matching with some dinner plate in the database up to some scale and rigid transformation. Therefore we can use the matched object in the database to "approximate" the scanned object.

We reuse the ICP algorithm for database matching. In particular, given the partially scanned point cloud data Ω, and an object, Φ, from the database, we define the distance of them as

$$\mathcal{D}(\Omega, \Phi) = \min_{\mathcal{R}(\cdot)} \mathsf{dist}(\mathcal{R}(\Omega), \Phi)$$

where $\mathcal{R}$ is the affine operator (scale, rotation, and linear translation), and dist is the sum of the shortest distance from each point in the point cloud to the surface.

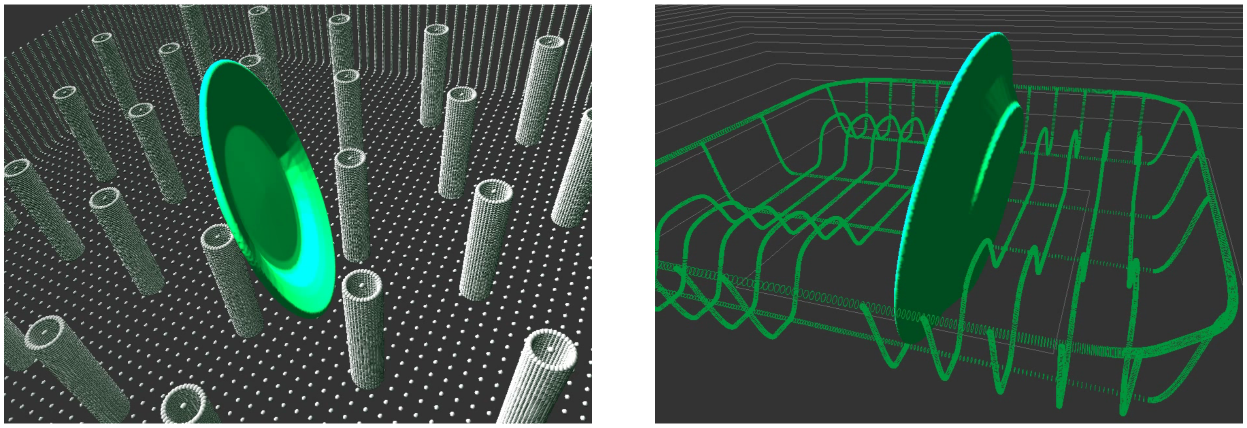
$$\mathsf{dist}(\mathsf{V}, \Phi) = \sum_{p_i \in \mathsf{V}} \psi_\Phi^2(p_i)$$

where ψ_Φ is the level-set function for the object Φ. The dist function can be quickly evaluated at runtime by pre-computing the level-set field of each database object and stored the field in the voxelized grids. The ICP algorithm can be used to find the $\mathcal{D}(\Omega, \Phi)$ iteratively. And the closest matching of the point cloud data is the database object which has the smallest $\mathcal{D}(\Omega, \Phi)$ value,

$$\mathsf{C}(\Omega) = \arg\min_{\Phi} \mathcal{D}(\Omega, \Phi)$$

B. Point-based Rigid-body Simulation

The destination place (e.g. the dishwasher) is scanned and represented by the point cloud. On the other hand, the objects are matched with the existing 3D models in a database. We use the most similar 3D model as a proxy of the original object in the rigid simulation. Therefore, our simulation involves the point cloud data representing the destination place and the triangle meshes approximating the objects. The collision detection can be quickly accelerated by pre-computing a



(a) For dishwasher 1 (b) For dishwasher 2

Fig. 3: Snapshots of the point-based rigid-body simulation

hierarchy of oriented bounding box. The standard linear complementary problem (LCP) arose from the collision events are solved using the projected Gauss-Seidel method. We refer the reader to the paper [16], [17], [18] for detailed introduction of the LCP problem and rigid-body simulation.

We validate a given placing point as follows. First, in the simulation, we set the object's initial position is slightly higher than the placing point. Let the object initial position and rotation expressed as S_0. Then we start the simulation. At each time step, we compute the kinetic energy of the object $E(t_n)$. The simulation runs until the object is resting in the container, i.e. $|E(t_n) - E(t_{n-1})| < \delta$. Let the object ending position and rotation be S_n. And we compute the rigid transformation between the initial state and the termination state, R_s, i.e. $S_n = R_s S_0$. We finally validate the placing point by checking the Frobenius norm of the rigid transformation,

$$\|R_s\|_F < T_s$$

Intuitively, a placing point is valid only when the termination state deviates from the initial state in a threshold, which means the container can hold the object stably at that placing point.

For each placing point, we need to test placing the object with different orientations. Currently in our implementation, we test four orientations, i.e. rotating the object along y-axis 0,45,90,135 degrees.

C. Train the SVM

Using the rigid simulation to validate a placing point is expensive. However, it can be used to generate the training examples for the supervised learning algorithm. We use the SVM to learn whether or not a given placing point is valid.

In particular, given an object Ω and its placing orientation $\boldsymbol{R}$, we model the probability of a point $\boldsymbol{x}$ being a valid placing point as

$$p(\boldsymbol{x}|\Omega, \boldsymbol{R}) = \frac{1}{1 + e^{-\phi(\boldsymbol{x},\Omega,\boldsymbol{R})^T\theta}} \tag{1}$$

where $\phi(\boldsymbol{x}, \Omega, \boldsymbol{R})$ returns the feature vector for the given object at the specified orientation. The prediction parameter θ is estimated using the standard logistic regression.

Right now, we have 36 features for a given object placing at point $\boldsymbol{x}$ with the orientation $\boldsymbol{R}$. To introduce these features, without loss of generality, we first assume the destination place is lying on the X-Z plane. We can place the objects by moving it vertically (along Y direction). First, we compute the bounding box B of the object and project it onto the X-Z plane. Let B_{xz} denote the projection. Next, we enlarge B_{xz} using a constant scalar. i.e. $\tilde{B}_{xz} = \alpha B_{xz}$. If the orientation of the object changes, B_{xz} might rotate accordingly, and so does $\tilde{B}_{xz}$. Then we iterate all the point cloud of the destination place, and add the points whose X-Z coordinates are inside of $\tilde{B}_{xz}$ into the set $\mathcal{I}$. Let $\boldsymbol{p}_{low}$ and $\boldsymbol{p}_{high}$ denote the lowest and highest points in $\mathcal{I}$ in Y direction. To compute the features, we divide $\tilde{B}_{xz}$ into 3×3 grids, such that the center grid is always identical to B_{xz}. For each grid, it has three vertical layers which spans from $\boldsymbol{p}_{low}$ to $\boldsymbol{p}_{high}$. Under this division, we have $3 \times 3 \times 3$ grids, denoted by g_{ijk}, $i, j, k = 1, 2, 3$.

The first 27 features are about the distribution of the points from $\mathcal{I}$ in the grids g_{ijk}. They are simply the ratio of the number of the points in g_{ijk} to the total number of points in $\mathcal{I}$.

The next 9 features are about the hight distribution of the points in $\tilde{B}_{xz}$. If we denote each cell of the 3×3 division of $\tilde{B}_{xz}$ as b_{ij}, $i, j = 1, 2, 3$. The feature for each b_{ij} is the hight of the highest point whose X-Z projection are in b_{ij}.

To generate the training examples, we first randomly select some points in the destination region, and some orientations for the object. We run the rigid body simulation to place the object to the selected point with the given orientation. The simulation runs until either the object arrives its resting state or it moves a certain distance away from the placing point. One placing point is classified as good if the rigid transformation of the object from the beginning state of the simulation to the ending state is small than a given threshold, measured by a weighted norm of the transformation matrix. Otherwise it is considered as bad placing point.

V. Experiments

Our experimental application is to load the kitchen objects into the dishwasher. We have two different dishwashers as shown in figure 3. The kitchen items we tested so far are a dinner plate and a bowl.

In the preliminary experiments, we have trained our SVM using 340 training examples generated by random sampling. All the examples are generated in 2 hours by running the rigid-body simulation. 70% of those training examples are used for training the SVM, which the remaining 30% of them are used for validation. By using the features specified in section IV, for the dinner plate, we can achieve about 7% for the training error and roughly 12% for the prediction error. For the bowl, we achieve about 4.6% of the training error and 8% of the prediction error.

In order to evaluate the quality of our SVM prediction, we again use the rigid-body simulation to validate the selected placing point from SVM, and compute the expected number of placing points we need to choose using SVM to find a valid one. On the other hand, we also randomly select the placing points and validate them using the rigid-body simulation. For the dinner plate, the expected number of placing point selections using SVM is about 2.6, while the expected number of random selection is 5.7. And for the bowl, the expected number of placing point selection using SVM is about 3.1, while the expected number of random selection is 15.4. Clearly, using our SVM prediction can largely accelerate the placing point selection.

VI. Conclusion and Future Work

In this project, we propose an algorithm to enable a personal robot to load the objects to some container. Both the object and the container might not be seen a priori. Our algorithm features a learning process to enable the robot to decide where to put the object and what the loading strategy is. The learning process is supervised by the training examples that are generated using a point-cloud-based rigid-body simulation. In order to avoid reconstructing the triangle meshes of the objects explicitly, we pre-compute a database of different kinds of kitchen items and match the partially scanned point cloud data with the objects in the database. The most geometrically closest object from the database is used in the simulation and SVM prediction to decide the loading strategy.

Designing a versatile algorithm to load different kinds of objects into different containers is a very hard problem. There are significant challenges remaining. Currently, perceiving of the object to object the high quality of point cloud data is still hard. Many of the sensors (e.g. the Swissranger sensor) produce significant noise which makes the database matching hard and ill-posed.

When there are multiple objects considered, not only are the placing positions important, but also the order of placing all the objects become tricky. Currently, we validate the loading sequence using the rigid-body simulation and find the feasible one. This method is hard to scale for a large number of objects where searching for the loading sequence using rigid-body simulation becomes the performance bottleneck. Searching for an optimal and feasible loading sequence for a large set of objects is also an interesting future work.

References

[1] T. Shin-ichi and M. Aatoshi, “Living and working with robots,” *Nipponia*, 2000.

[2] A. Saxena, J. Driemeyer, and A. Y. Ng., “Robotic grasping of novel objects using vision,” *International Journal of Robotics Research (IJRR)*, vol. 27, no. 2, pp. 157–173, Feb. 2008.

[3] S. Rusinkiewicz and M. Levoy, “Efficient variants of the icp algorithm,” in *3DIM*, 2001.

[4] N. J. Mitra, N. Gelfand, H. Pottmann, and L. J. Guibas, “Registration of point cloud data from a geometric optimization perspective,” in *Geometry Processing*, 2004, pp. 23–32.

[5] N. Gelfand, N. J. Mitra, L. J. Guibas, and H. Pottmann, “Robust global registration,” in *Eurographics Symposium on Geometry Processing*, 2005.

[6] A. Bicchi and V. Kumar, “Robotic grasping and contact: a review,” in *International Conference on Robotics and Automation (ICRA)*, 2000.

[7] K. Shimoga, “Robot grasp synthesis: a survey,” *International Journal of Robotics Research (IJRR)*, vol. 15, pp. 230–266, Feb. 1996.

[8] D. Kragic and H. Christensen, “Robust visual servoing,” *International Journal of Robotics Research (IJRR)*, vol. 22, pp. 923–939, 2003.

[9] A. Petrovskaya, O. Khatib, S. Thrun, and A. Ng, “Bayesian estimation for autonomous object manipulation based on tactile sensors,” in *International Conference on Robotics and Automation (ICRA)*, 2006.

[10] A. Morales, P. Sanz, and A. del Pobil, “Vision-based computation of three-finger grasps on unknown planer objects,” in *IEEE/RSJ Intelligent Robots and Systems Conference*, 2002.

[11] D. Bowers and R. Lumia, “Manipulation of unmodeled objects using intelligent grasping schemes,” *IEEE Transactions on Fuzzy Systems*, vol. 11, no. 3, 2003.

[12] M. A. Goodale, A. D. Milner, L. S. Jakobson, and D. P. Carey, “A neurological dissociation between perceiving objects and grasping them,” *Nature*, vol. 349, pp. 154–156, 1991.

[13] A. Saxena, S. Chung, and A. Ng, “Learning depth from single monocular images,” *Neural Information Processing Systems (NIPS)*, vol. 18, 2005.

[14] A. Saxena, M. Sun, and A. Ng, “Learning 3-d scene structure from a single still image,” in *ICCV workshop on 3D Representation for Recognition*, 2007.

[15] A. Saxena, L. Wong, and A. Ng, “Learning grasp strategies with partial shape information,” *AAAI*, 2008.

[16] E. Guendelman, A. Selle, F. Losasso, and R. Fedkiw, “Coupling water and smoke to thin deformable and rigid shells,” *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 973–981, Aug. 2005.

[17] D. Baraff, “Coping with friction for non-penetrating rigid body simulation,” *Computer Graphics (Proc. SIGGRAPH 91)*, pp. 31–40, 1991.

[18] D.Baraff, “Fast contact force computation for nonpenetrating rigid bodies,” in *ACM SIGGRAPH*, 1994, pp. 23–34.