Learning to Place New Objects

Yun Jiang, Changxi Zheng, Marcus Lim and Ashutosh Saxena

***Abstract*— The ability to place objects in an environment is an important skill for a personal robot. An object should not only be placed stably, but should also be placed in its preferred location/orientation. For instance, it is preferred that a plate be inserted vertically into the slot of a dish-rack as compared to being placed horizontally in it. Unstructured environments such as homes have a large variety of object types as well as of placing areas. Therefore our algorithms should be able to handle placing *new* object types and *new* placing areas. These reasons make placing a challenging manipulation task.**

In this work, we propose using a supervised learning approach for finding good placements given point-clouds of the object and the placing area. Our method combines the features that capture support, stability and preferred configurations, and uses a shared sparsity structure in its the parameters. Even when neither the object nor the placing area is seen previously in the training set, our learning algorithm predicts good placements. In robotic experiments, our method enables the robot to stably place known objects with a 98% success rate and 98% when also considering semantically preferred orientations. In the case of placing a new object into a new placing area, the success rate is 82% and 72%.[1]

I. Introduction

In several manipulation tasks of interest, such as arranging a disorganized kitchen, loading a dishwasher or laying a dinner table, a robot needs to pick up and place objects. While grasping has attracted great attention in previous works, placing remains under-explored. To place objects successfully, a robot needs to figure out where and in what orientation to place them—even in cases when the objects and the placing areas may not have been seen before.

Given a designated placing area (e.g., a dish-rack), this work focuses on finding good placements (which includes the location and orientation) for an object. An object can be placed stably in several different ways—for example, a plate could be placed horizontally on a table or placed vertically in the slots of a dish-rack, or even be side-supported when other objects are present (see Fig. 1). A martini glass should be placed upright on a table but upside down on a stemware holder. In addition to stability, some objects also have 'preferred' placing configuration that can be learned from prior experience. For example, long thin objects (e.g., pens, pencils, spoons) are placed horizontally on a table, but vertically in a pen- or cutlery-holder. Plates and other 'flat' objects are placed horizontally on a table, but plates are vertically inserted into the 'slots' of a dish-rack. Thus there are certain common features depending on the shape of objects and placing areas that indicate their preferred placements. These reasons make the space of potential placing configurations in indoor environments very large. The situation is further exacerbated when the robot has not seen the object or the placing area before.

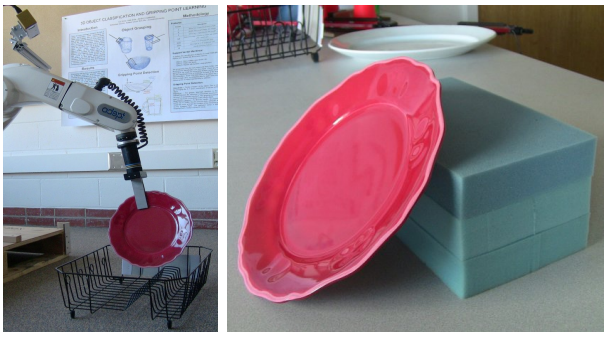


Fig. 1: How to place an object depends on the shape of the object and the placing environment. For example, a plate could be placed vertically in the dish rack (left), or placed slanted against a support (right). Furthermore, objects can also have a 'preferred' placing configuration. E.g., in the dish rack above, the preferred placement is inserting the plate vertically into the rack's slot, not horizontally.

In this work, we compute a number of features that indicate stability and support, and rely on supervised learning techniques to learn a functional mapping from these features to good placements. We learn the parameters of our model by maximizing the margin between the positive and the negative classes (similar to SVMs [2]). However we note that while some features remain consistent across different objects and placing areas, there are also features that are specific to particular objects and placing areas. We therefore impose a shared sparsity structure in the parameters while learning.

We test our algorithm extensively in the tasks of placing several objects in different placing environments. (See Fig. 2 for some examples.) The scenarios range from simple placing of objects on flat surfaces to narrowly inserting plates into dish-rack slots to hanging martini glasses upside down on a holding bar. We perform our experiments with a robotic arm that has no tactile feedback (which makes good predictions of placements important). Given the point-clouds of the placing area and the object with pre-defined grasping point, our method enabled our robot to stably place previously unseen objects in new placing areas with 82% success rate where 72% of the placements also had the preferred orientation. When the object and the placing area are seen in the training set and their 3D models are available, the success rate of stable as well as preferred-orientation placements both increased to 98% in further experiments.

The contributions of this work are as follows:

Y. Jiang, C. Zheng, M. Lim and A. Saxena are with the Department of Computer Science, Cornell University, Ithaca, NY 14850, USA {yunjiang,cxzheng}@cs.cornell.edu, mkl65@cornell.edu, asaxena@cs.cornell.edu

[1] This work was first presented at [1].

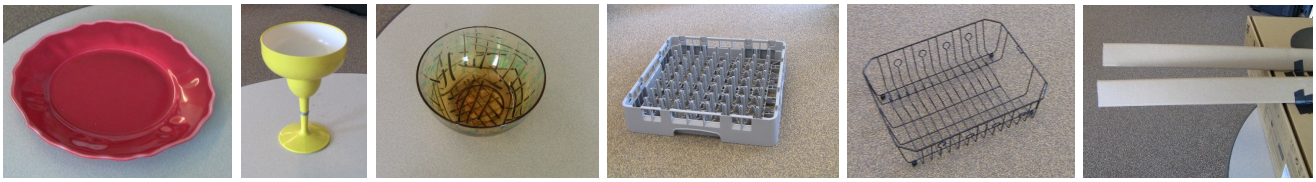


Fig. 2: Examples of a few objects (plate, martini glass, bowl) and placing areas (rack-1, rack-3, stemware holder). Full list is shown in Table II.

- While some there is some prior work in finding 'flat' surfaces [3], we believe that our work is the first one that considers placing new objects in complex placing areas.
- Our learning algorithm captures features that indicate stability of the placement as well as the 'preferred' placing configuration for an object.

II. Related Work

There is little previous work in placing objects, and it is restricted to placing objects upright on 'flat' surfaces. Edsinger and Kemp [4] considered placing objects on a flat shelf. The robot first detected the edge of the shelf, and then explored the area with its arm to confirm the location of the shelf. It used passive compliance and force control to gently and stably place the object on the shelf. This work indicates that even for flat surfaces, unless the robot knows the placing strategy very precisely, it takes good tactile sensing and adjustment to implement placing without knocking the object down. Schuster et al. [3] recently developed a learning algorithm to detect clutter-free 'flat' areas where an object can be placed. While these works assumes that the given object is already in its upright orientation, some other related works consider how to find the upright or the current orientation of the objects, e.g. Fu et al. [5] proposed several geometric features to learn the upright orientation from an object's 3D model and Saxena et al. [6] predicted the orientation of an object given its image. Our work is different and complementary to these studies: we generalize placing environments from flat surfaces to more complex ones, and desired configurations are extended from upright to all other possible orientations that can make the best use of the placing area.

Planning and rule-based approaches have been used in the past to move objects around. For example, Lozano-Perez et al. [7] considered picking up and placing objects by decoupling the planning problem into parts, and tested on grasping objects and placing them on a table. Sugie et al. [8] used rule-based planning in order to push objects on a table surface. However, these approaches assume known full 3D model of the objects, consider only flat surfaces, and do not model preferred placements.

In a related manipulation task, grasping, learning algorithms have been shown to be promising. Saxena et al. [9], [10], [11], [12] used supervised learning with images and partial point clouds as inputs to infer good grasps. Later, Le et al. [13] and Jiang et al. [14], [15] proposed new learning algorithms for grasping. Rao et al. [16] used point cloud to facilitate segmentation as well as learning grasps. Li et al. [17] combined object detection and grasping for higher accuracy in grasping point detection. In some grasping works that assume known full 3D models (such as GraspIt [18]), different 3D locations/orientations of the hand are evaluated for their grasp scores. Berenson et al. [19] consider grasping planning in complex scenes. Goldfeder [20] recently discussed a data-driven grasping approach. In this paper, we consider learning placements of objects in cluttered placing areas—which is a different problem because our learning algorithm has to consider the object as well as the environment to evaluate good placements. To the best of our knowledge, we have not seen any work about learning to place in complex situations.

III. System Overview

We frame the manipulation task of placing as a supervised learning problem. Given the features computed from the point-clouds of the object and the environment, the goal of the learning algorithm is to find a good placing hypothesis. As outlined in Fig. 4, the core part of our system is the placement classifier with supervised learning (yellow box in Fig. 4), which we will describe in detail in section IV-B. In this section, we briefly describe the other parts.

Our input to the system is the point-cloud of the object and the placing area obtained from a Microsoft Kinect sensor.[2]

A. Sampling of potential placements

Given the point-cloud of both the object and placing area, we first randomly sample several locations (100 in our experiments) in the bounding box of the placing area as the placement candidates. For each location, we have 18 orientations (45 degrees rotation along each dimension) of the object. For each of the 1800 candidate, we compute some features that capture stability and semantic preference (see Section IV-A) and then use a learning algorithm to estimate the score of it being a good placement.

B. Physics-based Simulation

In this work, we use rigid-body simulation to generate training data for our supervised learning algorithm (see Fig. 6 for some simulated placing tasks).

[2]Our learning algorithm presented in this paper is designed to work both with raw point-cloud obtained from a sensor (which is often noisy and incomplete) as well as full 3D models. When 3D models of the object are available, we can optionally register the raw point-cloud to the database of the 3D models, using the Iterative Closest Point (ICP) algorithm [21], [22]. The best matching object from the database is then used to represent the completed geometry of the scanned object. We test both approaches in our experiments in Section V-F.

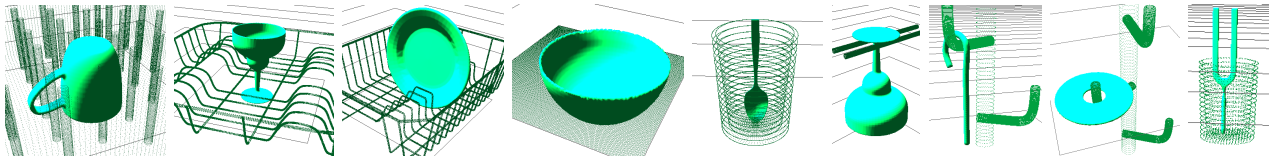


Fig. 3: Some snapshots from our rigid-body simulator showing different objects placed in different placing areas. (Placing areas from left: rack-1, rack-2, rack-3, flat surface, pen holder, stemware holder, hook, hook and pen holder. Objects from left: mug, martini glass, plate, bowl, spoon, martini glass, candy cane, disc and tuning fork.)

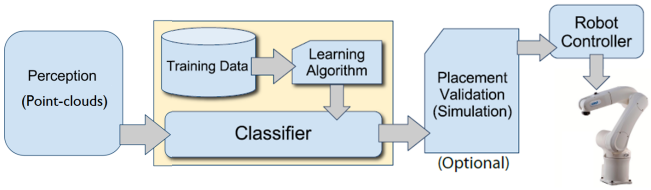


Fig. 4: **System Overview:** The core part is the placement classifier (yellow box) which is trained using supervised learning to identify a few candidates of placement configurations based on the perceived geometries of environments and objects. When we have access to 3D models, we perform 3D reconstruction on the raw point-cloud before classification and we validate predicted placements via a rigid-body simulation before realization. However, these two steps are not compulsory, as demonstrated in our robotic experiments.

A placement defines the location $\boldsymbol{T}_0$ and orientation $\boldsymbol{R}_0$ of the object in the environment. Its motion is computed by the rigid-body simulation at discrete time steps. At each time step, we compute the kinetic energy change of the object $\Delta E = E_n - E_{n-1}$. The simulation runs until the kinetic energy is almost constant ($\Delta E < \delta$), in which a stable state of the object can be assumed. Let $\boldsymbol{T}_s$ and $\boldsymbol{R}_s$ denote the stable state. We label the given placement as a valid one if the final state is close enough to the initial state, i.e. $\|\boldsymbol{T}_s - \boldsymbol{T}_0\|_2^2 + \|\boldsymbol{R}_s - \boldsymbol{R}_0\|_2^2 < \delta_s$.

Since the simulation itself has no knowledge of placing preferences, when creating the ground-truth training data, we manually labeled all the stable (as verified by the simulation) but un-preferred placements as negative examples.

We can also use the physics-based simulation to verify the predicted placements when 3D models are available (see the second robotic experiment in Section V-F). This simulation is computationally expensive,[3] but we only need to do it for a few top placements.

C. Realization

To realize a placement decision, our robotic arm grasps the object, moves it to the placing location with desired orientation, and releases it. Our main goal in this paper is to learn how to place objects well, therefore in our experiments the kinematically feasible grasping points are pre-calculated. However, the motion plan that realizes the object being placed in the predicted location and orientation is computed using an inverse kinematics solver with a simple path planner [23]. The robot executes that plan and opens the gripper in order to release the object for placing.

[3]A single stability test takes 3.8 second on a 2.93GHz dual-core processor on average.

IV. Learning Approach

A. Features

In this section, we describe the features used in our learning algorithm. We design the features to capture the following two properties:

- **Supports and Stability.** The object should be able to stay still after placing, and even better to be able to stand small perturbations.
- **Preferred placements.** Placements should also follow common practice. For example, the plates should be inserted into a dish-rack vertically and glasses should be placed upside down on a stemware holder.

We group the features into three categories. In the following description, we use O' to denote the point-cloud of the object O after being placed, and use B to denote the point-cloud of a placing area B. Let $\boldsymbol{p}_o$ be the 3D coordinate of a point $o \in \mathsf{O}'$ from the object, and $\boldsymbol{x}_t$ be the coordinate of a point $t \in \mathsf{B}$ from the placing area.

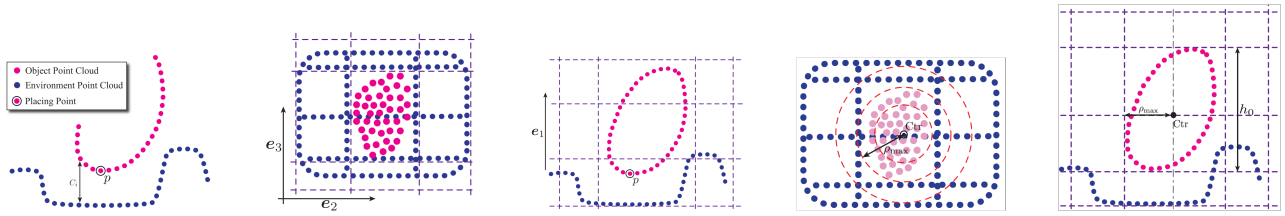
Supporting Contacts: Intuitively, an object is placed stably when it is supported by a wide spread of contacts. Hence, we compute features that reflect the distribution of the supporting contacts. In particular, we choose the top 5% points in the placing area closest to the object (measured by the vertical distance, c_i shown in Fig. 5a) at the placing point. Suppose the k points are $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$. We quantify the set of these points by 8 features:

1) Falling distance $\min_{i=1\ldots k} c_i$.
2) Variance in XY-plane and Z-axis respectively, $\frac{1}{k}\sum_{i=1}^{k}(\boldsymbol{x}'_i - \bar{\boldsymbol{x}}')^2$, where $\boldsymbol{x}'_i$ is the projection of $\boldsymbol{x}_i$ and $\bar{\boldsymbol{x}}' = \frac{1}{k}\sum_{i=1}^{k}\boldsymbol{x}'_i$.
3) Eigenvalues and ratios. We compute the three Eigenvalues ($\lambda_1 \geq \lambda_2 \geq \lambda_3$) of the covariance matrix of these k points. Then we use them along with the ratios λ_2/λ_1 and λ_3/λ_2 as the features.

Another common physics-based criterion is the center of mass (COM) of the placed object should be inside of (or close to) the region enclosed by contacts. So we calculate the distance from the centroid of O' to the nearest boundary of the 2D convex hull formed by contacts projected to XY-plane, $\mathcal{H}_{con}$. We also compute the projected convex hull of the whole object, $\mathcal{H}_{obj}$. The area ratio of these two polygons $S_{\mathcal{H}_{con}}/S_{\mathcal{H}_{obj}}$ is included as another feature.

Two more features representing the percentage of the object points below or above the placing area are used to capture the relative location.

Caging: There are some placements where the object would not be strictly immovable but is well confined within the



(a) supporting contacts (b) caging (top view) (c) caging (side view) (d) histogram (top view) (e) histogram (side view)

Fig. 5: Illustration of features that are designed to capture the stability and preferred orientations in a good placement.

placing area. A pen being placed upright in a pen-holder is one example. While this kind of placement has only a few supports from the pen-holder and may move under perturbations, it is still considered a good one. We call this effect 'gravity caging.'[4]

We capture this by partitioning the point-cloud of the environment and computing a battery of features for each zone. In detail, we divide the space around the object into $3 \times 3 \times 3$ zones. The whole divided space is the axis-aligned bounding box of the object scaled by 1.6, and the dimensions of the center zone are 1.05 times those of the bounding box (Fig. 5b and 5c). The point-cloud of the placing area is partitioned into these zones labelled by Ψ_{ijk}, $i, j, k \in \{1, 2, 3\}$, where i indexes the vertical direction $\boldsymbol{e}_1$, and j and k index the other two orthogonal directions, $\boldsymbol{e}_2$ and $\boldsymbol{e}_3$, on horizontal plane.

From the top view, there are 9 regions (Fig. 5b), each of which covers three zones in the vertical direction. The maximum height of points in each region is computed, leading to 9 features. We also compute the horizontal distance to the object in three vertical levels from four directions ($\pm\boldsymbol{e}_2, \pm\boldsymbol{e}_3$) (Fig. 5c). In particular, for each $i = 1, 2, 3$, we compute

$$\begin{aligned} d_{i1} &= \min_{\substack{\boldsymbol{x}_t \in \Psi_{i11} \cup \Psi_{i12} \cup \Psi_{i13} \\ \boldsymbol{p}_o \in \mathsf{O}'}} \boldsymbol{e}_2^T (\boldsymbol{p}_o - \boldsymbol{x}_t) \\ d_{i2} &= \min_{\substack{\boldsymbol{x}_t \in \Psi_{i31} \cup \Psi_{i32} \cup \Psi_{i33} \\ \boldsymbol{p}_o \in \mathsf{O}'}} -\boldsymbol{e}_2^T (\boldsymbol{p}_o - \boldsymbol{x}_t) \\ d_{i3} &= \min_{\substack{\boldsymbol{x}_t \in \Psi_{i11} \cup \Psi_{i21} \cup \Psi_{i31} \\ \boldsymbol{p}_o \in \mathsf{O}'}} \boldsymbol{e}_3^T (\boldsymbol{p}_o - \boldsymbol{x}_t) \\ d_{i4} &= \min_{\substack{\boldsymbol{x}_t \in \Psi_{i13} \cup \Psi_{i23} \cup \Psi_{i33} \\ \boldsymbol{p}_o \in \mathsf{O}'}} -\boldsymbol{e}_3^T (\boldsymbol{p}_o - \boldsymbol{x}_t) \end{aligned} \tag{1}$$

and produce 12 additional features.

The degree of gravity-caging also depends on the relative height of the object and the caging placing area. Therefore, we compute the histogram of the height of the points surrounding the object. In detail, we first define a cylinder centered at the lowest contact point with a radius that can just cover O'. Then points of B in this cylinder are divided into $n_r \times n_\theta$ parts based on 2D polar coordinates. Here, n_r is the number of radial divisions and n_θ is the number of divisions in azimuth. The vector of the maximum height in each cell (normalized by the height of the object), $H = (h_{1,1}, \ldots, h_{1,n_\theta}, \ldots, h_{n_r,n_\theta})$, is used as additional caging features. To make H rotation-invariant, the cylinder is always rotated to align polar axis with the highest point, so that the maximum value in H is always one of $h_{i,1}, i = 1...n_r$. We set $n_r = 4$ and $n_\theta = 4$ for our experiments.

Histogram Features: Generally, a placement depends on the geometric shapes of both the object and the placing area. We compute a representation of the geometry as follows. We partition the point-cloud of O' and of B radially and in Z-axis, centered at the centroid of O'. Suppose the height of the object is h_O and its radius is ρ_{max}. The 2D histogram with $n_z \times n_\rho$ number of bins covers the cylinder with the radius of $\rho_{max} \cdot n_\rho/(n_\rho - 2)$ and the height of $h_O \cdot n_z/(n_z - 2)$, illustrated in Fig. 5d and 5e. In this way, the histogram (number of points in each cell) can capture the global shape of the object as well as the local shape of the environment around the placing point.

We also compute the ratio of the two histograms as another set of features, i.e., the number of points from O' over the number of points from B in each cell. For numerical stability, the maximum ratio is fixed to 10 in practice. For our experiments, we set $n_z = 4$ and $n_\rho = 8$ and hence have 96 histogram features.

In total, we generate 145 features: 12 features for supporting contacts, 37 features for caging, and 96 for the histogram features. They are used in the learning algorithm described in the following sections.

B. Max-Margin Learning

Under the setting of a supervised learning problem, we are given a dataset of labeled good and bad placements (see Section V-E), represented by their features. Our goal is to learn a function of features that can determine whether a placement is good or not. As support vector machines (SVMs) [2] have strong theoretical guarantees in the performance and have been applied to many classification problems, we build our learning algorithm based on SVM.

Let $\phi_i \in \mathbb{R}^p$ be the features of i^{th} instance in the dataset, and let $y_i \in \{-1, 1\}$ represent the label, where 1 is a good placement and -1 is not. For n examples in the dataset, the soft-margin SVM learning problem [27] is formulated as:

$$\begin{aligned} &\min \frac{1}{2} \|\theta\|_2^2 + C \sum_{i=1}^{n} \xi_i \\ &\text{s.t. } y_i(\theta^T \phi_i - b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \ \forall 1 \leq i \leq n \end{aligned} \tag{2}$$

where $\theta \in \mathbb{R}^p$ are the parameters of the model, and ξ are the slack variables. This method finds a separating hyperplane

[4]This idea is motivated by previous works on force closure [24], [25] and caging grasps [26].

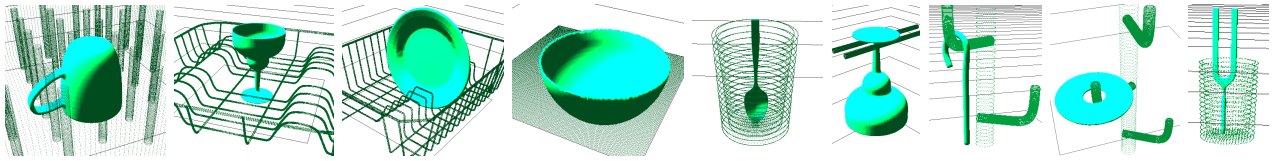


Fig. 6: Some snapshots from our rigid-body simulator showing different objects placed in different placing areas. Placing areas from left: rack1, rack2, rack3, flat surface, pen holder, stemware holder, hook, hook and pen holder. Objects from left: mug, martini glass, plate, bowl, spoon, martini glass, candy cane, disc and tuning fork. Rigid-body simulation is only used in labeling the training data and in half of the robotic experiments when 3D object models are used (Section V-F).

that maximizes the margin between the positive and the negative examples.

C. Shared-sparsity Max-margin Learning

If we look at the objects and their placements in the environment, we notice that there is an intrinsic difference between different placing settings. For example, it seems unrealistic to assume placing dishes into a rack and hanging martini glasses upside down on a stemware holder share exactly the same hypothesis, although they might agree on a subset of attributes. While some attributes may be shared across different objects and placing areas, there are some attributes that are specific to the particular setting. In such a scenario, it is not sufficient to have either one single model or several completely independent models for each placing setting that tend to suffer from over-fitting. Therefore, we propose to use a shared sparsity structure in our learning.

Say, we have M objects and N placing areas, thus making a total of $r = MN$ placing 'tasks' of particular object-area pairs. Each task can have its own model but intuitively these should share some parameters underneath. To quantify this constraint, we use ideas from recent works [28], [29] that attempt to capture the structure in the parameters of the models. [28] used a shared sparsity structure for multiple linear regressions. We apply their model to the classic soft-margin SVM.

In detail, for r tasks, let $\Phi_i \in \mathbb{R}^{p \times n_i}$ and Y_i denote training data and its corresponding label, where p is the size of the feature vector and n_i is the number of data points in task i. We associate every task with a weight vector θ_i. We decompose θ_i in two parts $\theta_i = S_i + B_i$: the self-owned features S_i and the shared features B_i. All self-owned features, S_i, should have only a few non-zero values so that they can reflect individual differences to some extent but would not become dominant in the final model. Shared features, B_i, need not have identical values, but should share similar sparsity structure across tasks. In other words, for each feature, they should all either be active or non-active. Let $\|S\|_{1,1} = \sum_{i,j} |S_i^j|$ and $\|B\|_{1,\infty} = \sum_{j=1}^{p} \max_i |B_i^j|$. Our new goal function is now:

$$\begin{aligned} \min_{\theta_i, b_i, i=1,\ldots,r} \quad & \textstyle\sum_{i=1}^{r} \left(\frac{1}{2} \|\theta_i\|_2^2 + C \sum_{j=1}^{n_i} \xi_{i,j} \right) + \\ & \lambda_S \|S\|_{1,1} + \lambda_B \|B\|_{1,\infty} \\ \text{subject to} \quad & Y_i^j (\theta_i^T \Phi_i^j + b_i) \geq 1 - \xi_{i,j}, \quad \xi_{i,j} \geq 0 \\ & \forall 1 \leq i \leq r, 1 \leq j \leq n_i \\ & \theta_i = S_i + B_i, \quad \forall 1 \leq i \leq r \end{aligned} \tag{3}$$

When testing in a new scenario, different models vote to determine the best placement.

While this modification results in superior performance with new objects in new placing areas, it requires one model per object-area pair and therefore it would not scale to a large number of objects and placing areas.

We transform this optimization problem into a standard quadratic programming (QP) problem by introducing auxiliary variables to substitute for the absolute and the maximum value terms. Unlike Eq. (2) which decomposes into r sub-problems, the optimization problem in Eq. (3) becomes larger, and hence takes a lot of computational time to learn the parameters. However, inference is still fast since predicting the score is simply the dot product of the features with the learned parameters. During the test, if the task is in the training set, then its corresponding model is used to compute the scores. Otherwise we use a voting system, in which we use the average of the scores evaluated by each learned models in the training data as the predicted score (see Section V-B for different training settings).

V. Experiments

A. Robot Hardware and Setup

We use our PANDA (PersonAl Non-Deterministic Arm) robot for our experiments, which comprises a Adept Viper s850 arm with six degrees of freedom, equipped with a parallel plate gripper that can open to up to a maximum width of 5cm. The arm has a reach of 105cm, together with our gripper. The arm plus gripper has a repeatability of about 1mm in XYZ positioning, but there is no force or tactile feedback in this arm. We use a Microsoft Kinect sensor, mounted on the arm, for perceiving the point-clouds.

B. Learning Scenarios

In real-world placing, the robot may or may not have seen the placing locations (and the objects) before. Therefore, we train our algorithm for four different scenarios:

1) Same Environment Same Object (**SESO**),
2) Same Environment New Object (**SENO**),
3) New Environment Same Object (**NESO**),
4) New Environment New Object (**NENO**).

If the object (environment) is 'same', then only this object (environment) is included in the training data. If it is 'new', then all other objects (environments) are included except the one for test. Through these four scenarios, we would be able to observe the algorithm's performance thoroughly.

TABLE I: Average performance of our algorithm using different features on the SESO scenario.

	chance	contact	caging	histogram	all
R_0	29.4	1.4	1.9	1.3	**1.0**
P@5	0.10	0.87	0.77	0.86	**0.95**
AUC	0.54	0.89	0.83	0.86	**0.95**

C. Baseline Methods

We compare our algorithm with the following three heuristic methods:

Chance. The location and orientation is randomly sampled within the bounding box of the area and guaranteed to be 'collision-free.'

Flat-surface-upright rule. Several methods exist for detecting 'flat' surfaces [3], and we consider a placing method based on finding flat surfaces. In this method, objects would be placed with *pre-defined* upright orientation on the surface. When no flat surface can be found, a random placement would be picked. Note that this heuristic actually uses *more* information than our method.

Finding lowest placing point. For many placing areas, such as dish-racks or containers, a lower placing point often gives more stability. Therefore, this heuristic rule chooses the placing point with the lowest height.

D. Evaluation Metrics

We evaluate our algorithm's performance on the following metrics:

- R_0: Rank of the first valid placement. ($R_0 = 1$ ideally.)
- P@5: In top 5 candidates, the fraction of valid placements.
- AUC: Area under ROC Curve [30], a classification metric computed from all the candidates.
- P_s: Success-rate (in %) of placing the object stably with the robotic arm.
- P_p: Success-rate (in %) of placing the object stably in preferred configuration with the robotic arm.

E. Learning Experiments

For evaluating our learning algorithm, we considered 7 placing areas (3 racks, a flat surface, pen holder, stemware holder and hook) and 8 objects (mug, martini glass, plate, bowl, spoon, candy cane, disc and tuning fork), as shown in Fig. 6. We generated one training and one test dataset for each object-environment pair. Each training/test dataset contains 1800 random placements with different locations and orientations. After eliminating placements with collisions, we have 37655 placements in total. These placements were then labeled by rigid-body simulation (see Section III-B).

Table I shows the average performance when we use different types of features: supporting contacts, caging and geometric signatures. While all the three types of features outperform chance, combining them together gives the highest performance under all evaluation metrics.

Next, Table II shows the comparison of three heuristic methods and three variations of SVM learning algorithms: 1) joint SVM, where one single model is learned from all the placing tasks in the training dataset; 2) independent SVM, which treats each task as a independent learning problem and learns a separate model per task; 3) shared sparsity SVM (Section IV-B), which also learns one model per task but with parameter sharing. Both independent and shared sparsity SVM use voting to rank placements for the test case.

Table II shows that all the learning methods (last six rows) outperform heuristic rules under all evaluation metrics. Not surprisingly, the chance method performs poorly (with Prec@5=0.1 and AUC=0.49) because there are very few preferred placements in the large sampling space of possible placements. The two heuristic methods perform well in some obvious cases such as using flat-surface-upright method for table or lowest-point method for rack1. However, their performance varies significantly in non-trivial cases, including the stemware holder and the hook. This demonstrates that it is hard to script a placing rule that works universally.

We get close to perfect results for SESO case—i.e., the learning algorithm can very reliably predict object placements if a known object was being placed in a previously seen location. The performance is still very high for SENO and NESO (i.e., when the robot has not seen the object or the environment before), where the first correct prediction is ranked 2.3 and 2.5 on average. This means we only need to perform simulation for about three times. There are some challenging tasks in the NESO case, such as placing on the hook and in the pen-holder when the algorithm has never seen these placing areas before. In those cases, the algorithm is biased by the training placing area and thus tends to predict horizontal placements.

The last learning scenario NENO is extremely challenging—for each task (of an object-area pair), the algorithm is trained without either the object or the placing area in the training set. In this case, R_0 increases to 8.9 with joint SVM, and to 5.4 using independent SVM with voting. However, shared sparsity SVM (the last row in the table) helps to reduce the average R_0 down to 2.1. While shared sparsity SVM outperforms other algorithms, the result also indicates that independent SVM with voting is better than joint SVM. This could be due to the large variety in the placing situations in the training set. Thus imposing one model for all tasks decreases the performance. We also observed that in cases where the placing strategy is very different from the ones trained on, the shared sparsity SVM does not perform well. For example, R_0 is 7.0 for spoon-in-pen-holder and is 5.0 for disk-on-hook. This could potentially be addressed by expanding the training dataset.

F. Robotic Experiments

We conducted experiments on our PANDA robot, using the same dataset for training. We tested 10 different placing tasks with 10 trials for each. A placement was considered successful if it was stable (i.e., the object remained still for more than a minute) and in its preferred orientation (within $\pm 15^\circ$ of the ground-truth orientation after placing).

Table III shows the results for three objects being placed by the robotic arm in four placing scenarios (see the bottom six rows). We obtain a 94% success rate in placing the

TABLE II: **Learning experiment statistics:** The performance of different learning algorithms in different scenarios is shown. The top three rows are the results for baselines, where no training data is used. The fourth to sixth row is trained and tested for the SESO, SENO and NESO case respectively using independent SVMs. The last three rows are trained using joint, independent and shared sparsity SVMs respectively for the NENO case.

Listed object-wise, averaged over the placing areas.

		plate: flat, 3 racks			mug: flat, 3 racks			martini: flat, 3 racks, stemware holder			bowl: flat, 3 racks			candy cane: flat, hook, pen holder			disc: flat, hook, pen holder			spoon: flat, pen holder			tuning fork: flat, pen holder			**Average**		
		R_0	P@5	AUC	R_0	P@5	AUC	R_0	P@5	AUC	R_0	P@5	AUC	R_0	P@5	AUC	R_0	P@5	AUC	R_0	P@5	AUC	R_0	P@5	AUC	R_0	P@5	AUC
baseline	chance	4.0	0.20	0.49	5.3	0.10	0.49	6.8	0.12	0.49	6.5	0.15	0.50	102.7	0.00	0.46	32.7	0.00	0.46	101.0	0.20	0.52	44.0	0.00	0.53	29.4	0.10	0.49
	flat-up	4.3	0.45	0.38	11.8	0.50	0.78	16.0	0.32	0.69	6.0	0.40	0.79	44.0	0.33	0.51	20.0	0.40	0.81	35.0	0.50	0.30	35.5	0.40	0.66	18.6	0.41	0.63
	lowest	27.5	0.25	0.73	3.8	0.35	0.80	39.0	0.32	0.83	7.0	0.30	0.76	51.7	0.33	0.83	122.7	0.00	0.86	2.5	0.50	0.83	5.0	0.50	0.76	32.8	0.30	0.80
indep.	SESO	1.3	0.90	0.90	1.0	0.85	0.92	1.0	1.00	0.95	1.0	1.00	0.92	1.0	0.93	1.00	1.0	0.93	0.97	1.0	1.00	1.00	1.0	1.00	1.00	1.0	0.95	0.95
	SENO	1.8	0.65	0.75	2.0	0.50	0.76	1.0	0.80	0.83	4.3	0.45	0.80	1.7	0.87	0.97	4.3	0.53	0.78	1.0	1.00	0.98	1.5	0.60	0.96	2.3	0.65	0.83
	NESO	1.0	0.85	0.76	1.5	0.70	0.85	1.2	0.88	0.86	1.5	0.80	0.84	2.7	0.47	0.89	3.7	0.47	0.85	5.5	0.10	0.72	8.0	0.00	0.49	2.5	0.62	0.81
NENO	joint	8.3	0.50	0.78	2.5	0.65	0.88	5.2	0.48	0.81	2.8	0.55	0.87	16.7	0.33	0.76	20.0	0.33	0.81	23.0	0.20	0.66	2.0	0.50	0.85	8.9	0.47	0.81
	indep.	2.0	0.70	0.86	1.3	0.80	0.89	1.2	0.86	0.91	3.0	0.55	0.82	9.3	0.60	0.87	11.7	0.53	0.88	23.5	0.40	0.82	2.5	0.40	0.71	5.4	0.64	0.86
	shared	1.8	0.70	0.84	1.8	0.80	0.85	1.6	0.76	0.90	2.0	0.75	0.91	2.7	0.67	0.88	1.3	0.73	0.97	7.0	0.40	0.92	1.0	0.40	0.84	2.1	0.69	0.89

Listed placing area-wise, averaged over the objects.

		rack1: plate, mug, martini, bowl			rack2: plate, mug, martini, bowl			rack3: plate, mug, martini, bowl			flat: all objects			pen holder: candy cane, disc, spoon, tuningfork			hook: candy cane, disc			stemware holder: martini			**Average**		
		R_0	P@5	AUC	R_0	P@5	AUC	R_0	P@5	AUC	R_0	P@5	AUC	R_0	P@5	AUC	R_0	P@5	AUC	R_0	P@5	AUC	R_0	P@5	AUC
baseline	chance	3.8	0.15	0.53	5.0	0.25	0.49	4.8	0.15	0.48	6.6	0.08	0.48	128.0	0.00	0.50	78.0	0.00	0.46	18.0	0.00	0.42	29.4	0.10	0.49
	flat-up	2.8	0.50	0.67	18.3	0.05	0.47	4.8	0.20	0.60	1.0	0.98	0.91	61.3	0.05	0.45	42.0	0.00	0.45	65.0	0.00	0.58	18.6	0.41	0.63
	lowest	1.3	0.75	0.87	22.0	0.10	0.70	22.0	0.15	0.80	4.3	0.23	0.90	60.3	0.60	0.85	136.5	0.00	0.71	157.0	0.00	0.56	32.8	0.30	0.80
indep.	SESO	1.0	1.00	0.91	1.3	0.75	0.83	1.0	1.00	0.93	1.0	0.95	1.00	1.0	1.00	0.98	1.0	1.00	1.00	1.0	1.00	1.00	1.0	0.95	0.95
	SENO	1.25	0.70	0.83	2.5	0.40	0.66	3.5	0.45	0.72	1.4	0.93	0.96	3.5	0.55	0.89	2.5	0.60	0.81	-	-	-	2.3	0.65	0.83
	NESO	1	1.00	0.83	1.5	0.60	0.77	1.3	0.75	0.82	2.9	0.65	0.86	4.0	0.20	0.70	6.5	0.20	0.81	1.0	1.00	0.79	2.5	0.62	0.81
NENO	joint	1.8	0.60	0.92	2.5	0.70	0.84	8.8	0.35	0.70	2.4	0.63	0.86	20.5	0.25	0.76	34.5	0.00	0.69	18.0	0.00	0.75	8.9	0.47	0.81
	indep.	1.3	0.70	0.85	2.0	0.55	0.88	2.5	0.70	0.86	1.9	0.75	0.89	12.3	0.60	0.79	29.0	0.00	0.79	1.0	1.00	0.94	5.4	0.64	0.86
	shared	1.8	0.75	0.88	2.0	0.70	0.86	2.3	0.70	0.84	1.3	0.75	0.92	4.0	0.60	0.88	3.5	0.40	0.92	1.0	0.80	0.95	2.1	0.69	0.89

TABLE III: **Robotic experiments**. The algorithm is trained using shared sparsity SVM under the two learning scenarios: SESO and NENO. 10 trials each are performed for each object-placing area pair. In the experiments with object models, R_0 stands for the rank of first predicted placements passed the stability test. In the experiments without object models, we do not perform stability test and thus R_0 is not applicable. In summary, robotic experiments show a success rate of 98% when the object has been seen before and its 3D model is available, and show a success-rate of 82% (72% when also considering semantically correct orientations) when the object has not been seen by the robot before in any form.

			plate: rack1	plate: rack3	plate: flat	martini: rack1	martini: rack3	martini: flat	martini: stem.	bowl: rack1	bowl: rack3	bowl: flat	**Average**
w/ obj models	SESO	R_0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	**1.0**
		$P_s(\%)$	100	100	100	100	100	100	80	100	100	100	**98**
		$P_p(\%)$	100	100	100	100	100	100	80	100	100	100	**98**
	NENO	R_0	1.8	2.2	1.0	2.4	1.4	1.0	1.2	3.4	3.0	1.8	1.9
		$P_s(\%)$	100	100	100	80	100	80	100	100	100	100	96
		$P_p(\%)$	100	100	100	60	100	80	100	80	100	100	92
w/o obj models	SESO	R_0	-	-	-	-	-	-	-	-	-	-	-
		$P_s(\%)$	100	80	100	80	100	100	80	100	100	100	94
		$P_p(\%)$	100	80	100	80	80	100	80	100	100	100	92
	NENO	R_0	-	-	-	-	-	-	-	-	-	-	-
		$P_s(\%)$	80	60	100	80	80	100	70	70	100	80	**82**
		$P_p(\%)$	80	60	100	60	70	80	60	50	80	80	**72**

objects stably in SESO case, and 92% if we disqualify those stable placements that were not preferred ones. In the NENO case, we achieve 82% performance for stable placing, and 72% performance for preferred placing. Figure 7 shows several screenshots of our robot placing the objects. There were some cases where the martini glass and the bowl were placed horizontally in rack1. In these cases, even though the placements were stable, they were not counted as preferred. Even small displacement errors while inserting the martini glass in the narrow slot of the stemware holder often resulted in a failure. In general, several failures for the bowl and the martini glass were due to incomplete capture of the point-cloud which resulted in the object hitting the placing area (e.g., the spikes in the dish-rack).

In order to analyze the errors caused by the incomplete point-clouds, we did another experiment in which we use 3D models to reconstruct the full geometry. Furthermore, since we had access to a full solid 3D model, we verified the stability of the placement using the rigid-body simulation (Section III-B) before executing it on the robot. If it failed the stability test, we would try the placement with the next highest score until it would pass the test.

In this setting with a known library of object models, we obtain a 98% success rate in placing the objects in SESO case. The robot failed only in one experiment, when the martini glass could not fit into the narrow stemware holder due to a small displacement occurred in grasping. In the NENO case, we achieve 96% performance in stable placements and 92% performance in preferred placements. This experiment indicates that with better sensing of the point-clouds, our learning algorithm can give better performance.

Fig. 7 shows several screenshots of our robot placing

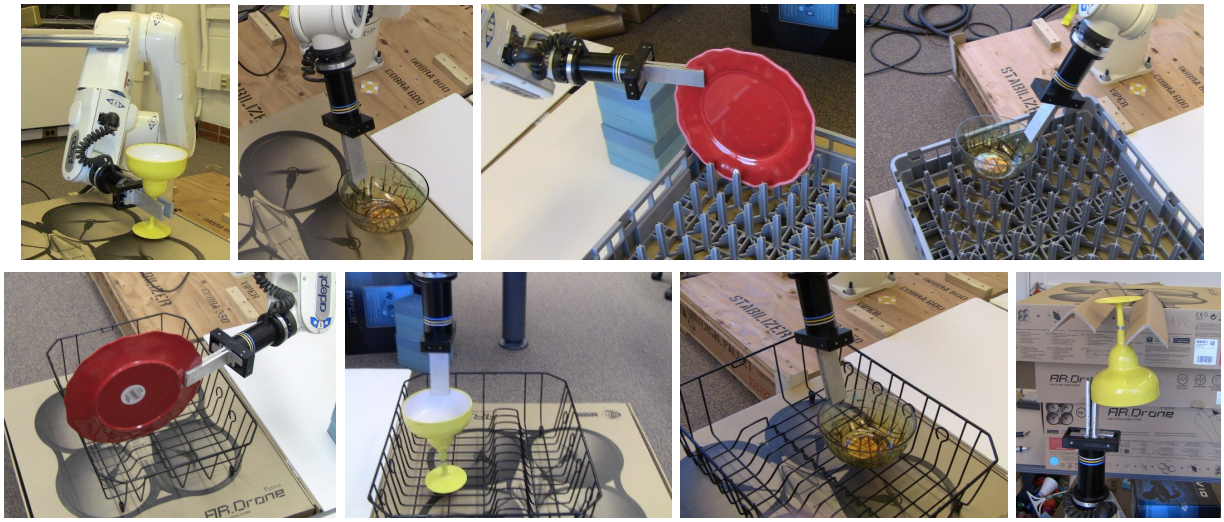


Fig. 7: Some screenshots of our robot placing different objects in several placing areas. In the examples above, the robot placed the objects in stable as well as preferred configuration, except for the top right image where a bowl is placed stably in upright orientation on rack-1. A more preferred orientation is upside down.

objects. Some of the cases are quite tricky, for example placing the martini-glass hanging from the stemware holder. Videos of our robot placing these objects is available at: `http://pr.cs.cornell.edu/placingobjects`

VI. Conclusion and Discussion

In this paper, we considered the problem of placing objects in various types of placing areas, especially when the objects and the placing areas may not have been seen by the robot before. We first presented features that contain information about stability and preferred orientations. We then used a shared-sparsity SVM learning approach for predicting the top placements. In our extensive learning and robotic experiments, we showed that different objects can be placed successfully in several environments, including flat surfaces, several dish-racks, a stemware holder and a hook.

There still remain several issues for placing objects that we have not addressed yet, such as using force or tactile feedback while placing. We also need to consider detecting the appropriate placing areas in an environment for an object (e.g., table vs ground for a placing a shoe), and how to place multiple objects in the tasks of interest such as arranging a disorganized kitchen or cleaning a cluttered room.

References

[1] Y. Jiang, C. Zheng, M. Lim, and A. Saxena, "Learning to place new objects," in *Workshop on Mobile Manipulation: Learning to Manipulate*, 2011.

[2] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, 1995.

[3] M. Schuster, J. Okerman, H. Nguyen, J. Rehg, and C. Kemp, "Perceiving clutter and surfaces for object placement in indoor environments," in *Int' Conf Humanoid Robots*, 2010.

[4] A. Edsinger and C. Kemp, "Manipulation in human environments," in *Int'l Conf Humanoid Robots*. IEEE, 2006.

[5] H. Fu, D. Cohen-Or, G. Dror, and A. Sheffer, "Upright orientation of man-made objects," *ACM Trans. Graph.*, vol. 27, pp. 42:1–42:7, 2008.

[6] A. Saxena, J. Driemeyer, and A. Y. Ng, "Learning 3-d object orientation from images," in *ICRA*, 2009.

[7] T. Lozano-Pérez, J. Jones, E. Mazer, and P. O'Donnell, "Task-level planning of pick-and-place robot motions," *Computer*, 2002.

[8] H. Sugie, Y. Inagaki, S. Ono, H. Aisu, and T. Unemi, "Placing objects with multiple mobile robots-mutual help using intention inference," in *ICRA*, 1995.

[9] A. Saxena, J. Driemeyer, and A. Ng, "Robotic grasping of novel objects using vision," *IJRR*, 2008.

[10] A. Saxena, L. Wong, and A. Y. Ng, "Learning grasp strategies with partial shape information," in *AAAI*, 2008.

[11] A. Saxena, J. Driemeyer, J. Kearns, and A. Ng, "Robotic grasping of novel objects," in *Neural Information Processing Systems*, 2006.

[12] A. Saxena, "Monocular depth perception and robotic grasping of novel objects," Ph.D. dissertation, Stanford University, 2009.

[13] Q. Le, D. Kamm, A. Kara, and A. Ng, "Learning to grasp objects with multiple contact points," in *ICRA*. IEEE, 2010, pp. 5062–5069.

[14] Y. Jiang, S. Moseson, and A. Saxena, "Efficient Grasping from RGBD Images: Learning using a new Rectangle Representation," *ICRA*, 2011.

[15] Y. Jiang, J. Amend, H. Lipson, and A. Saxena, "Learning hardware agnostic grasps for a universal jamming gripper," in *ICRA*, 2012.

[16] D. Rao, Q. Le, T. Phoka, M. Quigley, A. Sudsang, and A. Ng, "Grasping Novel Objects with Depth Segmentation," in *IROS*, 2010.

[17] C. Li, A. Kowdle, A. Saxena, and T. Chen, "Towards holistic scene understanding: Feedback enabled cascaded classification models," in *NIPS*, 2010.

[18] M. T. Ciocarlie and P. K. Allen, "On-line interactive dexterous grasping," in *Eurohaptics*, 2008.

[19] D. Berenson, R. Diankov, K. Nishiwaki, S. Kagami, and J. Kuffner, "Grasp planning in complex scenes," in *Humanoid Robots*, 2009.

[20] C. Goldfeder, M. Ciocarlie, J. Peretzman, H. Dang, and P. Allen, "Data-driven grasping with partial sensor data," in *IROS*, 2009.

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

[22] N. Gelfand, N. J. Mitra, L. J. Guibas, and H. Pottmann, "Robust global registration," in *Eurographics Symp Geometry Processing*, 2005.

[23] S. LaValle and J. Kuffner Jr, "Randomized kinodynamic planning," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 1. IEEE, 1999, pp. 473–479.

[24] V. Nguyen, "Constructing stable force-closure grasps," in *ACM Fall joint computer conf*, 1986.

[25] J. Ponce, D. Stam, and B. Faverjon, "On computing two-finger force-closure grasps of curved 2D objects," *IJRR*, 1993.

[26] R. Diankov, S. Srinivasa, D. Ferguson, and J. Kuffner, "Manipulation planning with caging grasps," in *Humanoid Robots*, 2008.

[27] T. Joachims, *Making large-Scale SVM Learning Practical*. MIT-Press, 1999.

[28] A. Jalali, P. Ravikumar, S. Sanghavi, and C. Ruan, "A Dirty Model for Multi-task Learning," *NIPS*, 2010.

[29] C. Li, A. Saxena, and T. Chen, "θ-mrf: Capturing spatial and semantic structure in the parameters for scene understanding," in *NIPS*, 2011.

[30] J. Hanley and B. McNeil, "The meaning and use of the area under a receiver operating (roc) curvel characteristic," *Radiology*, 1982.