Learning to Place New Objects in a Scene

Yun Jiang, Marcus Lim, Changxi Zheng and Ashutosh Saxena
Computer Science Department, Cornell University, USA.
yunjiang@cs.cornell.edu, mkl65@cornell.edu, {cxzheng,asaxena}@cs.cornell.edu

***Abstract*—Placing is a necessary skill for a personal robot to have in order to perform tasks such as arranging objects in a disorganized room. The object placements should not only be stable but also be in their semantically preferred placing areas and orientations. This is challenging because an environment can have a large variety of objects and placing areas that may not have been seen by the robot before.**

In this paper, we propose a learning approach for placing multiple objects in different placing areas in a scene. Given point-clouds of the objects and the scene, we design appropriate features and use a graphical model to encode various properties, such as the stacking of objects, stability, object-area relationship and common placing constraints. The inference in our model is an integer linear program, which we solve efficiently via an LP relaxation. We extensively evaluate our approach on 98 objects from 16 categories being placed into 40 areas. Our robotic experiments show a success rate of 98% in placing known objects and 82% in placing new objects stably. We use our method on our robots for performing tasks such as loading several dish-racks, a bookshelf and a fridge with multiple items.[1]

I. Introduction

In order to autonomously perform common daily tasks such as setting up a dinner table, arranging a living room or organizing a closet, a personal robot should be able to figure out where and how to place objects. However, this is particularly challenging because there can potentially be a wide range of objects and placing environments. Some of them may not have been seen by the robot before. For example, to tidy a disorganized house, a robot needs to decide *where* the best place for an object is (e.g., books should be placed on a shelf or a table and plates are better inserted in a dish-rack), and *how* to place the objects in an area (e.g. clothes can be hung in a closet and wine glasses can be held upside down on a stemware holder). In addition, limited space, such as in a cabinet, raises another problem of how to *stack* various objects together for efficient storage. Determining such a placing strategy, albeit rather natural or even trivial to (most) people, is quite a challenge for a robot.

In this paper, we consider multiple objects and placing areas represented by possibly incomplete and noisy point-clouds. Our goal is to find proper placing strategies to place the objects into the areas. A placing *strategy* of an object is described by a preferred placing area for the object and a 3D location and orientation to place it in that area. As an example, Fig. 1 shows one possible strategy to place six different types of objects onto a bookshelf. In practice, the following criteria should be considered.

[1]Parts of this work were described in Jiang et al. (2011b, 2012b).

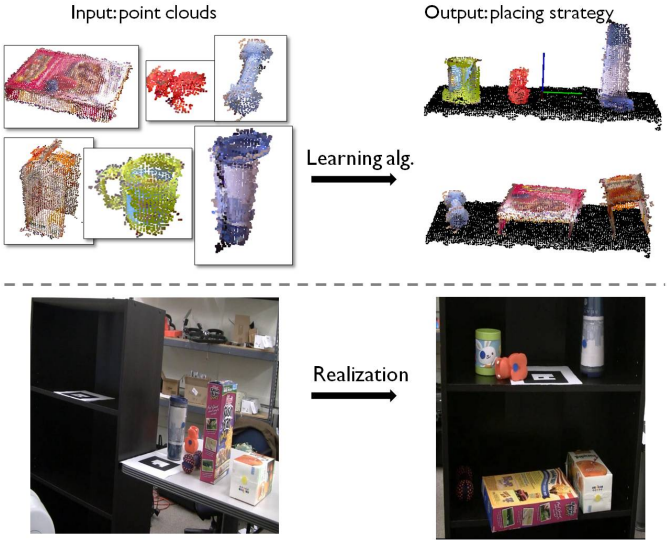


Fig. 1: An example task of placing items on a bookshelf. Given the point-clouds of the bookshelf and six objects to be placed (shown in top-left part of the figure), our learning algorithm finds out the best placing strategy, specified by the location and orientation of every object (shown in top-right). Following this inferred strategy, the robot places each object accordingly. The bottom part of the figure shows the scene before and after placing. Note that in some cases, the placing environment can be quite complex (e.g, see Fig. 10).

Stability: Objects should be placed stably in order to avoid falling. For example, some objects can be stably placed upright on flat surfaces,[2] plates can be placed stably in different orientations in a dish-rack, and a pen can be placed horizontally on a table but vertically in a pen-holder.

Semantic preference: A placement should follow common human preferences in placing. For instance, shoes should be placed on the ground but not on a dinner plate, even though both areas have geometrically similar flat surfaces. Therefore, a robot should be able to distinguish the areas semantically, and make a decision based on common human practice.

Stacking: A placement should consider possible stacking of objects such as piling up dinner plates. However, this raises more challenges for a robot because it has to decide which objects can be stacked together semantically. For example, it is a bad idea to stack a dinner plate on top of a cell phone rather than another dinner plate. In addition, the robot has to decide the order of stacking in a dynamically changing environment,

[2]Even knowing the "upright" orientation for an arbitrary object is a non-trivial task (Fu et al., 2008).

since previously placed objects can change the structure of placing areas for objects placed later.

In addition to the difficulties introduced by these criteria, perceiving the 3D geometry of objects and their placing environments is nontrivial as well. In this paper, we use a depth camera mounted on a robot to perceive the 3D geometries as point-clouds. In practice, the perceived point-clouds can be noisy and incomplete (see Fig. 1), requiring the robot to be able to infer placements with only partial and noisy geometric information.

In this paper, we address these challenges using a learning-based approach. We encode human preferences about placements as well as the geometric relationship between objects and their placing environments by designing appropriate features. We then propose a graphical model that has two substructures to capture the stability and the semantic preferences respectively. The model also incorporates stacking and constraints that keeps the placing strategy physically feasible. We use max-margin learning for estimating the parameters in our graphical model. The learned model is then used to score the potential placing strategies. Given a placing task, although inferring the best strategy (with the highest score) is provably NP-complete, we express the inference as an integer linear programming (ILP) problem which is then solved efficiently using an linear programming (LP) relaxation.

To extensively test our approach, we constructed a large placing database composed of 98 household objects from 16 different categories and 40 placing areas from various scenes. Experiments ranged from placing a single object in challenging situations to complete-scene placing where we placed up to 50 objects in real offices and apartments. In the end-to-end test of placing multiple objects in different scenes, our algorithm significantly improves the performance—on metrics of stability, semantic correctness and overall impressions on human subjects—as compared to the best baseline. Quantitatively, we achieve an average accuracy of 83% for stability and 82% for choosing a correct placing area for single-object placements. Finally, we tested our algorithm on two different robots on several placing tasks. We then applied our algorithm to several practical placing scenarios, such as loading multiple items in dish-racks, loading a fridge, placing objects on a bookshelf and cleaning a disorganized room. We have also made the code and data available online at: `http://pr.cs.cornell.edu/placingobjects`

Our contributions in this paper are as follows:

- There has been little work on robotic placing, compared to other manipulation tasks such as grasping. To the best of our knowledge, this is the first work on placing objects in a complex scene. In particular, our scenes are composed of different placing areas such as several dish-racks, a table, a hanging rod in a closet, etc.
- We propose a learning algorithm that takes object stability as well as semantic preferences into consideration.
- Our algorithm can handle objects perceived as noisy and incomplete point-clouds. The objects may not have been seen by the robot before.
- We consider stacking when placing multiple objects in a scene.
- We test our algorithm extensively on a variety of placing tasks, both on offline real data and on robots.

This paper is organized as follows. We start with a review of the related work in Section II. We describe our robot platforms in Section III. We then formulate the placing problem in a machine learning framework in Section IV. We present our learning algorithm for single-object placements in Section V and the corresponding experiments in Section VI. We give the algorithm and the experiments for multiple-object placements in Section VII and Section VIII respectively. Finally, we conclude the paper in Section IX.

II. Related Work

While there has been significant previous work on grasping objects (e.g., Bicchi and Kumar, 2000; Ciocarlie and Allen, 2008; Miller et al., 2003; Saxena et al., 2006, 2008b,c; Berenson et al., 2009; Rao et al., 2010; Hsiao et al., 2009; Brook et al., 2011; Le et al., 2010; Jiang et al., 2011a; Dogar and Srinivasa, 2011; Rosales et al., 2011; Jiang et al., 2012a), there is little work on object placement, and it is restricted to placing objects on flat horizontal surfaces. For example, Schuster et al. (2010) recently developed a learning algorithm to detect clutter-free 'flat' areas where an object can be placed. Unfortunately, there are many non-flat placing areas where this method would not apply. Even if placing only on flat surfaces, one needs to decide the upright or the current orientation of a given object, which is a challenging task. For example, Fu et al. (2008) proposed several geometric features to learn the upright orientation from an object's 3D model and Saxena et al. (2009a) predicted the orientation of an object given its 2D image. Recently, Glover et al. (2011) used a database of models to estimate the pose of objects with partial point-clouds. Our work is different and complementary to these studies: we generalize placing environment 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. Furthermore, we consider: 1) placing objects in scenes comprising a wide-variety of placing areas, such as dish-racks, stemware holders, cabinets and hanging rods in closets; 2) placing multiple objects; 3) placing objects in semantically preferred locations (i.e., how to choose a proper placing context for an object).

For robotic placing, one component is to plan and control the arm to place the objects without knocking them down. Edsinger and Kemp (2006) considered placing objects on a flat shelf, but their focus was to use passive compliance and force control to gently place the object on the shelf. Planning and rule-based approaches have been used to move objects around. For example, Lozano-Pérez et al. (2002) proposed a task-level (in contrast with motion-level) planning system and tested it on picking and placing objects on a table. Sugie et al. (1995) used rule-based planning in order to push objects on a table surface. However, most of these approaches assume known

full 3D models of the objects, consider only flat surfaces, and do not model semantic preferences in placements.

Placing multiple objects also requires planning and high-level reasoning about the order of execution. Berenson et al. (2009) coupled planning and grasping in cluttered scenes. They utilized an 'environment clearance score' in determining which object to grasp first. Toussaint et al. (2010) integrated control, planning, grasping and reasoning in the 'blocks-world' application in which table-top objects were rearranged into several stacks by a robot. How to arrange objects efficiently is also related to the classic bin packing problem (Coffman Jr et al., 1996) which can be approached as integer linear programming (Fisher, 1981), constraint satisfaction problem (Pisinger and Sigurd, 2007) or tabu search (Lodi et al., 2002). These studies focus on generic planning problems and are complementary to ours.

Contextual cues (e.g., Torralba et al., 2010) have proven helpful in many vision applications. For example, using estimated 3D geometric properties from images can be useful for object detection (Saxena et al., 2008a, 2009b; Heitz et al., 2008; Li et al., 2010; Hedau et al., 2009; Divvala et al., 2009). In Xiong and Huber (2010) and Anand et al. (2011), contextual information was employed to estimate semantic labels in 3D point-clouds of indoor environments. Deng et al. (2011) used object context for object retrieval. Fisher and Hanrahan (2010) and Fisher et al. (2011) designed a context-based search engine using geometric cues and spatial relationships to find the proper object for a given scene. Unlike our work, their goal was only to retrieve the object but not to place it afterwards. While these works address different problems, our work that captures the semantic preferences in placing is motivated by them.

Object categorization is also related to our work as objects from same category often share similar placing patterns. Categorization in 2D images is a well-studied computer vision problem. Early work (e.g., Winn et al., 2005; Leibe et al., 2004; Fergus et al., 2003; Berg et al., 2005) tried to solve shape and orientation variability, limited by a single viewpoint. Motivated by this limitation, multi-view images were considered for categorizing 3D generic object by connecting 2D features (Thomas et al., 2006; Savarese and Fei-Fei, 2007). When 3D models were available, some work categorized objects based on 3D features instead, e.g., using synthetic 3D data to extract pose and class discriminant features (Liebelt et al., 2008), and using features such as spin images (Johnson and Hebert, 1999) and point feature histograms (Rusu et al., 2008) for 3D recognition. Instead of images or 3D models, Lai et al. (2011) proposed a learning algorithm for categorization using both RGB and multi-view point-cloud. These works are complementary to ours in that we do not explicitly categorize the object before placing, but knowing the object category could potentially help in placing.

Most learning algorithms require good features as input, and often these features are hand-designed for the particular tasks (Fu et al., 2008; Saxena et al., 2008c). There have also been some previous works on high-dimensional 3D features (Ho et al., 2010; Rusu et al., 2008; Johnson and Hebert, 1999; Liebelt et al., 2008; Saxena et al., 2008c) but they do not directly apply to our problem. There is also a large body of work on automatic feature selection (see Dy and Brodley, 2004; Liu and Yu, 2005; Jetchev and Toussaint, 2011), which could potentially improve the performance of our algorithm.

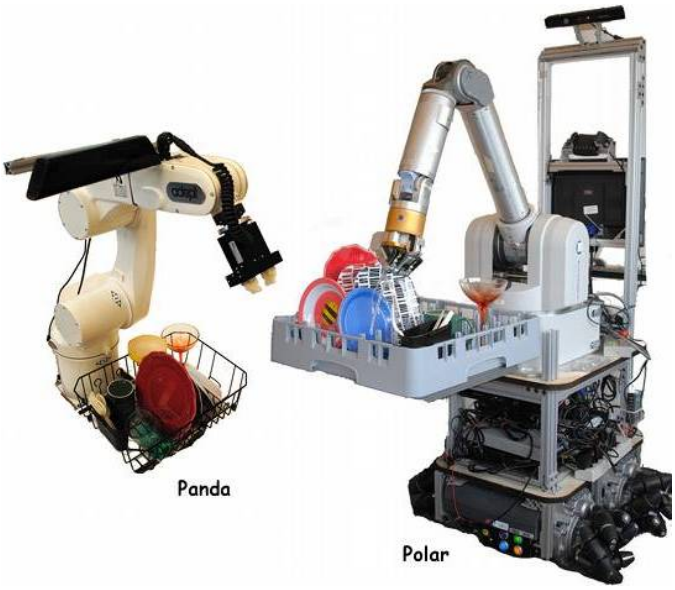


Fig. 2: Our two robotic platforms used for testing our algorithm: PANDA (PersonAl Non-Deterministic Arm, on left) is an Adept arm with a parallel-plate gripper, mounted with a Kinect sensor. POLAR (PersOnaL Assistant Robot, on right) is a 7-DOF Barrett arm mounted on a Segway Omni base, with a three-fingered hand and a Kinect sensor on top.

In the area of robotic manipulation, a wide range of problems have been studied so far, such as folding towels and clothes (Maitin-Shepard et al., 2010), opening doors (Jain and Kemp, 2010; Klingbeil et al., 2010), inferring 3D articulated objects (Sturm et al., 2010; Katz and Brock, 2008) and so on. However, they address different manipulation tasks and do not apply to the placing problem we consider in this paper. Our work is the first one to consider object placements in complex scenes.

III. Robot Platforms and System

In this section, we describe our robot platforms and details of the system.

Our primary sensor is a depth camera that gives an RGB image together with the depth value at each pixel. This could be converted into a colored point-cloud as in Fig. 1. In our experiments, we used a Microsoft Kinect sensor which has a resolution of 640x480 for the depth image and an operation range of 0.8m to 3.5m.

Our PANDA (PersonAl Non-Deterministic Arm) robot (Fig. 2 left) is a 6-DOF Adept Viper s850 arm equipped with a parallel-plate gripper and a Kinect sensor. The arm, together with the gripper, has a reach of 105cm. The arm has a repeatability of 0.03mm in XYZ positioning, but the

estimated repeatability with the gripper is 0.1mm. The Kinect sensor-arm calibration was accurate up to an average of 3mm. The arm weighs 29kg and has a rated payload of 2.5kg, but our gripper can only hold up to 0.5kg. The Adept Viper is an industrial arm and has no force or tactile feedback, so even a slight error in positioning can result in a failure to place.

Our POLAR (PersOnaL Assistant Robot) robot (Fig. 2 right) is equipped with a 7-DOF WAM arm (by Barrett Technologies) and a three-fingered hand mounted on a Segway RMP 50 Omni base. A Kinect sensor is mounted on top. The arm has a positioning accuracy of $\pm$0.6mm. It has a reach of 1m and a payload of 3kg. The hand does not have tactile feedback. The mobile base can be controlled by speed and has a positioning accuracy of only about 10cm.

Our system comprises three steps: perception, inference and realization. We first use the sensor to capture the point-cloud of the object and the scene. We then apply the learning algorithm to find good placements. In the last step, given the 3D location and orientation of the placing point, the robot uses path planning to move the object to the destination and releases it in the end.

In a more complex task involving placing multiple objects (such as organizing a room in Section VIII-F), the robot needs to move to the object, pick up the object, move to the designated area, find out the location and orientation for placing, and place it accordingly. We start with a 3D model of the room. The initial locations of the objects to be placed are given and their grasps are defined in advance. We use open-source softwares such as ROS (Quigley et al., 2009) and OpenRAVE (Diankov and Kuffner, 2008) for path planning and obstacle avoidance, and use AR-markers (Ferguson, 2011) for robot localization. Our goal in this paper is to develop an algorithm for estimating good placements given the point-clouds of the objects and the scene as input.

IV. Problem Formulation

In this section, we formulate the problem of how to predict good placements in a placing task.

Specifically, our goal is to place a set of objects in a scene that can contain several potential placing areas. Both the objects and the placing areas are perceived as point-clouds that can be noisy and incomplete. A placement of an object is specified by 1) a 3D location describing at which 3D position in the scene the object is placed, and 2) a 3D rotation describing the orientation of the object when it is placed. In the following, we first consider the problem of single-object placements. We will then extend it to multiple-object placements by adding semantic features and modifying the algorithm to handle multiple objects. The learning algorithm for single-object placements is only a special case of the algorithm for multiple-object placements.

A. Single-Object Placements

Here, our goal is to place an object stably in a designated placing area in the scene. We consider stability and preferences in orientation, but would not consider the semantic preferences about which placing area to choose. Specifically, the goal is to infer the 3D location ℓ (in a placing area E) and 3D orientation c in which to place the object O.

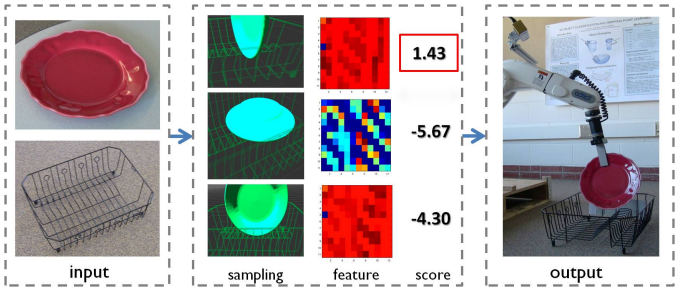


Fig. 3: Our formulation of single-object placements as a learning problem. Steps from left to right: 1) we are given an object to be placed and a placing area, in the form of point-clouds; 2) we first sample possible placements, extract features for each sample, and then use our learned model to compute a score for each sample. The higher the score is, the more likely it is to be a good placement; 3) the robot plans a path to the predicted placement and follows it to realize the placing.

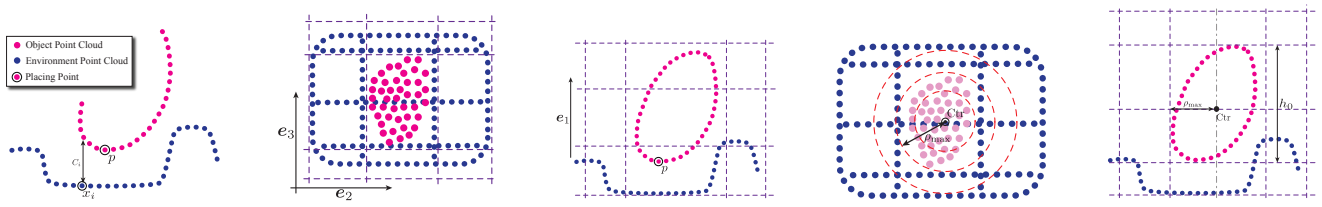
We illustrate the problem formulation in Fig. 3. We are given an object O and a placing area E in the form of point-clouds. Given the input, we first sample a set of possible placements, compute relevant features for each, and then use the learned model to compute a score for each candidate. The highest-score placement is then selected to be executed by the robot. We describe our features and the learning algorithm in Section V.

B. Multiple-Object Placements

In addition to finding a proper location and orientation to place the object, we often need to decide which placing area in the scene is semantically suitable for placing the object in. When multiple objects are involved, we also need to consider stacking while placing them.

As an example, consider organizing a kitchen (see Fig. 8). In such a case, our goal is to place a set of given objects into several placing areas. One can place objects in two ways: directly on an existing area in the environment (e.g., saucepans on the bottom drawer), or stacking one on top of another (e.g., bowls piled up in the cabinet).

Formally, the input of this problem is n objects $\mathcal{O} = \{O_1, \ldots, O_n\}$ and m placing areas (also called environments) $\mathcal{E} = \{E_1, \ldots, E_m\}$, all of which are represented by point-clouds. The output will be a placing strategy depicting the final layout of the scene after placing. It is specified by the pose of every object, which includes the configuration (i.e., 3D orientation) c_i, the placing area (or another object when stacking) selected, and the relative 3D location ℓ_i w.r.t. this area. We propose a graphical model to represent the placing strategy, where we associate every possible strategy with a potential function. Finding the best placing strategy is then equivalent to maximizing the potential function. Details are in Section VII.



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

Fig. 4: Illustration of features in our learning algorithm for single-object placements. These features are designed to capture the stability and preferred orientations in a good placement.

V. Algorithm for Single-Object Placements

In order to identify good placements, we first need to design features that indicate good placements across various objects and placing areas. We then use a max-margin learning algorithm to learn a function that maps a placement, represented by its features, to a placing score. In testing, we first randomly sample some placements, then use the function to find the highest-score candidate as our best placement.

A. Features

The features used in our learning algorithm are designed to capture the following two properties:

- **Supports and Stability.** The object should stay still after placing. Ideally, it should also be able to withstand small perturbations.
- **Preferred Orientation.** A good placement should have semantically preferred orientation as well. For example, plates should be inserted into a dish-rack vertically and glasses should be held upside down on a stemware holder.

An important property of the stability features is invariance under translation and rotation (about the gravity, i.e., Z-axis). This is because as long as the relative configuration of the object and the placing area remains same, the features should not change. Most of our features will follow this property.

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. 4a) 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 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.'[3]

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. 4b and 4c). 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. 4b), 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 $(\pm\boldsymbol{e}_2, \pm\boldsymbol{e}_3)$

[3]This idea is motivated by previous works on force closure (Nguyen, 1986; Ponce et al., 1993) and caging grasps (Diankov et al., 2008).

(Fig. 4c). 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 single-object placement 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. 4d and 4e. 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. The maximum ratio is fixed to 10 in practice. For single-object placement experiments, we set $n_z = 4$ and $n_\rho = 8$ and hence have 96 histogram features.

In total, we generate 145 features for the single-object placement experiments: 12 features for supporting contacts, 37 features for caging, and 96 for the histogram features.

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 VI-A), 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) (Cortes and Vapnik, 1995) 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 (Joachims, 1999) 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 that maximizes the margin between the positive and the negative examples. Note that our formulation here is a special case of the graphical model for the multiple-object placements described in Section VII. We also use max-margin learning to estimate the parameters in both cases.

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 (Jalali et al., 2010; Li et al., 2011) that attempt to capture the structure in the parameters of the models. Jalali et al. (2010) 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

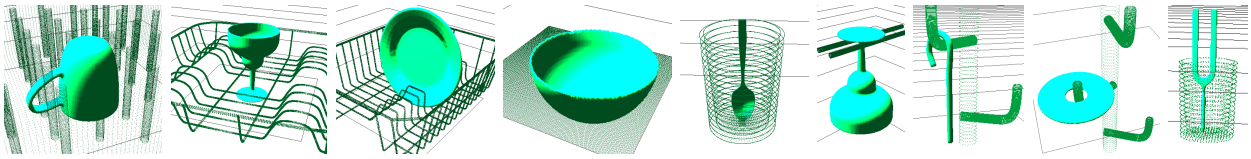


Fig. 5: Some snapshots from our rigid-body simulator showing different objects placed in different placing areas. Placing areas from left: rack1, rack2, rack3, flat surface, pen holder, stemware holder, hook, hook and pen holder. Objects from left: mug, martini glass, plate, bowl, spoon, martini glass, candy cane, disc and tuning fork. Rigid-body simulation is only used in labeling the training data (Section VI-A) and in first half of the robotic experiments when 3D object models are used (Section VI-F).

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**

$\|B\|_{1,\infty} = \sum_{j=1}^{p} \max_i |B_i^j|$. 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 does not scale to a large number of objects and placing areas.

VI. Experiments on Placing Single Objects

We perform experiments on placing a single object in a designated placing area. The dataset includes 8 different objects and 7 placing areas. In these experiments, our main purpose is to analyze the performance of our learning approach in finding stable placements with preferred orientations. Section VIII describes our full experiments with placing multiple objects in complete 3D scenes.

A. Data

Our dataset contains 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). 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 that have collisions, we have 37655 placements in total.

These placements were labeled by rigid-body simulation (Fig. 5) and then used for our supervised learning algorithm. Simulation enabled us to generate massive amounts of labeled data. However, the simulation itself had no knowledge of placing preferences. When creating the ground-truth training data, we manually labeled all the stable (as verified by the simulation) but non-preferred placements as negative examples.

B. Learning Scenarios

In real-world placing, the robot may or may not encounter new placing areas and new objects. Therefore, we trained our algorithm for two different scenarios: 1) Same Environment Same Object (**SESO**), where training data only contains the object and the placing environment to be tested. 2) New Environment New Object (**NENO**). In this case, the training data includes all other objects and environments except the one for test. We also considered two additional learning scenarios, SENO and NESO, in Jiang et al. (2012b). More results can be found there.

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 (Schuster et al., 2010), 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 (Hanley and McNeil, 1982), a classification metric computed from all the candidates.

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 row is trained and tested for the SESO case. 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			mug			martini			bowl			candy cane			disc			spoon			tuning fork			**Average**		
		flat, 3 racks			flat, 3 racks			flat, 3 racks, stemware holder			flat, 3 racks			flat, hook, pen holder			flat, hook, pen holder			flat, pen holder			flat, pen holder					
		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
	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
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			rack2			rack3			flat			pen holder			hook			stemware holder			**Average**		
		plate, mug, martini, bowl			plate, mug, martini, bowl			plate, mug, martini, bowl			all objects			candy cane, disc, spoon, tuningfork			candy cane, disc			martini					
		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
	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
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

- $P_{\text{stability}}$: Success-rate (in %) of placing the object stably with the robotic arm.
- $P_{\text{preference}}$: Success-rate (in %) of placing the object stably in preferred configuration with the robotic arm.

E. Learning Experiments

We first verified that having different types of features is helpful in performance, as shown in Table I. While all three types of features outperform chance, combining them together gives the best results 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 V-C), 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 four 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 the 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 learning scenario NENO is extremely challenging—here, 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 from 1.0 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 issue could potentially be addressed by expanding the training dataset. For comparison, the average AUC (area under ROC curve) of shared sparsity SVM is 0.89, which compares to 0.94 in Table V for corresponding classes (dish-racks, stemware holder and pen holder).

F. Robotic Experiments

We conducted single-object placing experiments on our PANDA robot with the Kinect sensor, using the same dataset (Section VI-A) for training. We tested 10 different placing tasks with 10 trials for each.

In each trial, the robot had to pick up the object and place it in the designated area. The input to our algorithm in these experiments was raw point-clouds of the object and the placing area (see Fig. 6). Given a placement predicted by our learning algorithm and a feasible grasp, the robot used path planning to move the object to the destination and released it. A placement

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. P_s stands for $P_{\text{stability}}$ and P_p stands for $P_{\text{preference}}$. 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			martini				bowl			**Average**
			rack1	rack3	flat	rack1	rack3	flat	stem.	rack1	rack3	flat	
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**

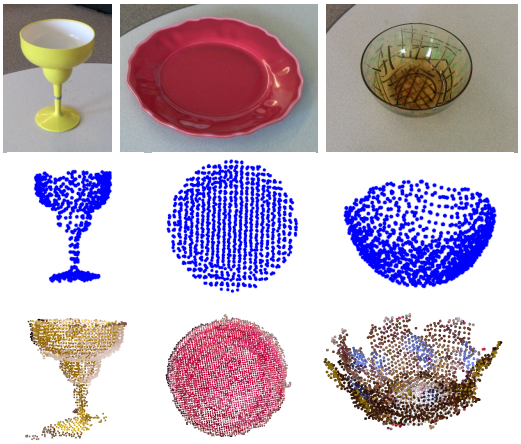


Fig. 6: Three objects used in our robotic experiments. The top row shows the real objects. Center row shows the perfect point-clouds extracted from object models. Bottom row shows the raw point-clouds perceived from the Kinect sensor, used in the robotic experiments.

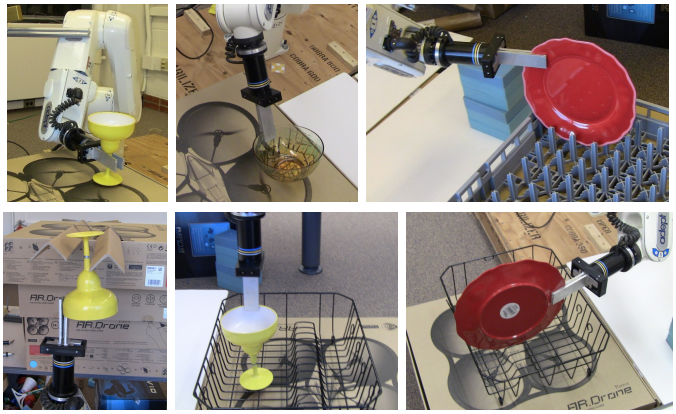


Fig. 7: Robotic arm placing different objects in several placing areas: a martini glass on a flat surface, a bowl on a flat surface, a plate in rack1, a martini glass on a stemware holder, a martini glass in rack3 and a plate in rack3.

was considered successful if it was stable (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 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 source of the errors in robotic placing, we did another experiment in which we factored away the errors caused by the incomplete point-clouds. In detail, we recovered the full 3D geometry by registering the raw point-cloud against a few parameterized objects in a database using the Iterative Closest Point (ICP) algorithm (Rusinkiewicz and Levoy, 2001; Gelfand et al., 2005). Furthermore, since we had access to a full solid 3D model, we verified the stability of the placement using the rigid-body simulation 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. Even though this simulation was computationally

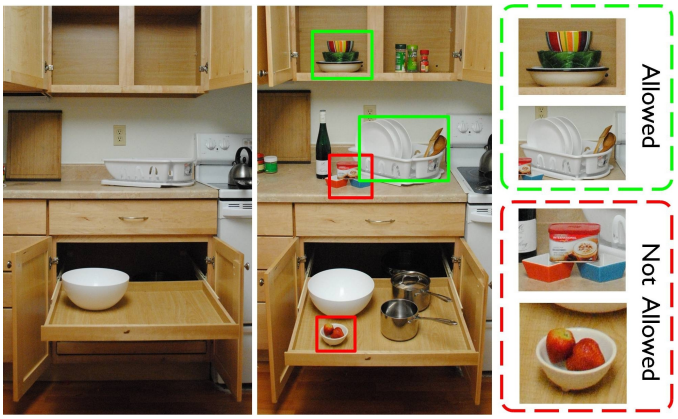


Fig. 8: Given an initial kitchen scene (left), a possible placing strategy for multiple objects could be as shown in the middle image: loading the dish-rack with spatulas and plates, or stacking them up in the cabinet, storing saucepans on the bottom drawer, etc. In this paper, we only allow chain stacking (see text in Section VII), which allows most but not all the possible placing situations (right column).

expensive,[4] we only needed to compute this for a few top scored placements.

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.

VII. Algorithm for Placing Multiple Objects

We will now describe our approach for placing multiple objects in a scene, where we also need to decide which placing area in the scene is semantically suitable for placing every object in.

We first state our assumptions in this setting. We consider two scenarios: 1) the object is placed directly on the placing area; 2) the object is stacked on another object. While an unlimited number of objects can stack into one pile in series (e.g., the plates and bowls in the cabinet in Fig. 8), we do not allow more than one object to be placed on one single object and we do not allow one object to be placed on more than one object. We refer this assumption as **'chain stacking'**. For instance, in Fig. 8, it is not allowed to place two strawberries in the small bowl, nor is it allowed to place a box on top of the blue and the orange sauce bowls at the same time. Note that this constraint does not limit placing multiple objects on a placing area directly, e.g., the dish-rack is given as a placing area and therefore we can place multiple plates in it.

A physically feasible placing strategy needs to satisfy two constraints: 1) **Full-coverage**: every given object must be placed somewhere, either directly on a placing area or on top of another object. 2) **Non-overlap**: two objects cannot be placed at same location, although being in the same placing area is allowed. For example, multiple plates can be loaded into one dish-rack, however, they cannot occupy the same slot at the same location. 3) **Acyclic**: placing strategy should be acyclic, i.e., if object A is on top of B (either directly or indirectly), then B cannot be above A.

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

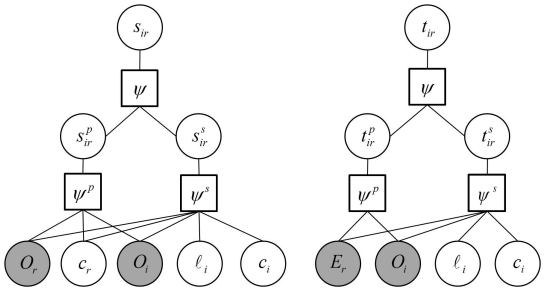


Fig. 9: Graphical models for two types of single placements: stacking on another object (left) and directly placing on an environment (right). The shaded nodes are observed point-clouds.

A. Graphical Model

We now introduce our representation of a placing strategy as a graphical model, followed by our design of features, max-margin learning algorithm, and inference as a linear programming problem.

As we mentioned in Section IV-B, given n objects $\mathcal{O} = \{O_1, \ldots, O_n\}$ and m placing areas (or environments) $\mathcal{E} = \{E_1, \ldots, E_m\}$ our goal is to find a placing strategy, that can be specified by a tuple (S, T, C, L):

- $S = \{s_{ir} \in \{0,1\} | 1 \le i \le n, 1 \le r \le n\}$: whether O_i is stacking on top of O_r.
- $T = \{t_{ir} \in \{0,1\} | 1 \le i \le n, 1 \le r \le m\}$: whether O_i is put directly on E_r.
- $C = \{c_i \in \text{SO(3)} | 1 \le i \le n\}$: the configuration (i.e., 3D orientation) of O_i.
- $L = \{\ell_i \in \mathbb{R}^3 | 1 \le i \le n\}$: the 3D location of O_i w.r.t. its base.

We now design a potential function over all placing strategies given a scene, i.e., $\Psi(S, T, L, C, \mathcal{O}, \mathcal{E})$. This function reflects the placing quality and we associate higher potential to better placing strategies. The best strategy is the one with the maximum potential. Therefore, our goal is,

$$(S^*, T^*, L^*, C^*) = \arg \max_{S,T,L,C} \Psi(S, T, L, C, \mathcal{O}, \mathcal{E}) \qquad (4)$$

where the solution (S, T, L, C) should follow certain constraints that we will discuss in Section VII-D.

We use an undirected graphical model (Markov networks (Koller and Friedman, 2009)) for defining the potential. Fig. 9 shows two simplified graphical models for a single placement. The entire graphical model for multiple objects and placing areas is an assembly of these basic structures (one for each s_{ir} and t_{ir}) with shared O, E, ℓ and c nodes. We now explain this graphical model and show how to factorize

$\Psi(S,T,L,C,\mathcal{O},\mathcal{E})$ into small pieces so that learning and inference is tractable.

Our graphical model indicates the following independence:

- s_{ir} and t_{ir} are conditionally independent given $L,C,\mathcal{O},\mathcal{E}$.
- $s_{ir}(t_{ir})$ only depends on objects (environments) involved in this single placing task of placing O_i on O_r (E_r). This implies that, when placing an object i on object j, it does not matter where object j is placed.

Consequently, we can factorize the overall potential as,

$$\begin{aligned}\Psi(S,T,L,C,\mathcal{O},\mathcal{E}) &= \prod_{i,r}\Psi(s_{ir},O_i,\ell_i,c_i,O_r,c_r) \\ &\quad\times\prod_{i,r}\Psi(t_{ir},O_i,\ell_i,c_i,E_r) \quad (5)\end{aligned}$$

We further factorize the potential of each placement into three terms to encode the stability and semantic preference in placing, as shown in Fig. 9. For each placement, we introduce two binary variables indicating its stability (with superscript s) and semantic preference (with superscript p). Considering they are latent, we have,

$$\begin{aligned}&\Psi(s_{ir},O_i,\ell_i,c_i,O_r,c_r)= \\ &\textstyle\sum_{s^s_{ir},s^p_{ir}\in\{0,1\}}\psi(s_{ir},s^s_{ir},s^p_{ir})\psi^s(s^s_{ir},O_i,\ell_i,c_i,O_r,c_r)\psi^p(s^p_{ir},O_i,O_r,c_r) \\ &\Psi(t_{ir},O_i,\ell_i,c_i,E_r)= \\ &\textstyle\sum_{t^s_{ir},t^p_{ir}\in\{0,1\}}\psi(t_{ir},t^s_{ir},t^p_{ir})\psi^s(t^s_{ir},O_i,\ell_i,c_i,E_r)\psi^p(s^p_{ir},O_i,E_r) \quad (6)\end{aligned}$$

The intuition is that a placement is determined by two factors: stability and semantic preference. This is quantified by the potential function ψ. ψ^s captures stability which is only dependent on local geometric information, i.e., exact poses and locations. On the other hand, semantic preference concerns how well this object fits the environment. So the function ψ^p is determined by the object and the base regardless of the details of the placement. For example, placing a plate in a dish-rack has a high semantic preference over the ground, but different placements (vertical vs. horizontal) would only change its stability. Note that in stacking, semantic preference (s^p_{ir}) is also dependent on the configuration of the base (c_r), since the base's configuration would affect the context. For instance, we can place different objects on a book lying horizontally, but it is hard to place any object on a book standing vertically.

Based on the fact that a good placement should be stable as well as semantically correct, we set $\psi(s_{ir},s^s_{ir},s^p_{ir})=1$ if $s_{ir}=(s^s_{ir}\wedge s^p_{ir})$ otherwise 0. We do the same for $\psi(t_{ir},t^s_{ir},t^p_{ir})$. As for the potential functions ψ^s and ψ^p, they are based on a collection of features that indicate good stability and semantic preference. Their parameters are learned using the max-margin algorithm described in Section VII-C.

B. Features

In the multiple-object placements, we introduce additional semantic features for choosing a good placing area for an object. Our stability features ϕ_s are similar to those described in Section V-A.[5] Semantic features ϕ_p depend only on O and B, where O denotes the point-cloud of object O to be placed and B denote the point-cloud of the base (either an environment or another object). This is because the semantic features should be invariant to different placements of the object within the same placing area. We describe them in the following:

TABLE IV: Features for multiple objects placements and their dimensions ('Dim').

Feature descriptions	Dim	Feature descriptions	Dim
Stability	**178**	**Semantic preference**	**801**
Supporting contacts	12	Zernike	37×4
Caging features	4	BOW	100×4
Histograms	162	Color histogram	46×4
		Curvature histogram	12×4
		Overall shape	5×4
		Relative height	1

- **3D Zernike Descriptors:** Often the semantic information of an object is encoded in its shape. Because of their rotation- and translation-invariant property, we apply 3D Zernike descriptors (Novotni and Klein, 2003) to O and B for capturing their geometric information. This gives us 37 values for a point-cloud.
- **Bag-of-words (BOW) Features:** Fast Point Feature Histograms (FPFH) (Rusu et al., 2009) are persistent under different views and point densities and produce different signatures for points on different geometric surfaces such as planes, cylinders, spheres, etc. We compute a vocabulary of FPFH signatures by extracting 100 cluster centers from FPFH of every point in our training set. Given a test point-cloud, we compute the FPFH signature for each point and associate it with its nearest cluster center. The histogram over these cluster centers makes our BOW features.
- **Color Histograms:** The features described above capture only the geometric cues, but not the visual information such as color. Color information can give clues to the texture of an object and can help identify some semantic information. We compute a 2D histogram of hue and saturation (6×6) and a 1D 10-bin histogram of intensity, thus giving a total of 46 features.
- **Curvature Histograms:** We estimated the curvature of every point using Point Cloud Library (Rusu and Cousins, 2011), and then compute its 12-bin histogram.
- **Overall Shape:** Given a point-cloud, we compute three Eigenvalues of its covariance matrix ($\lambda_1\geq\lambda_2\geq\lambda_3$). We then compute their ratios (λ_2/λ_1, λ_3/λ_2), thus giving us a total of 5 features.

We extract the aforementioned semantic features for both O and B separately. For each, we also add their pairwise product and pairwise minimum, thus giving us 4 values for

[5] In multiple-object placement experiments, we only use the relative height caging features with $n_r=1$ and $n_\theta=4$, and use 81-bin (9×9) grid without the ratios for the histogram features.

each feature. The last feature is the height to the ground (only for placing areas), thus giving a total of 801 features. We summarize the semantic features in Table IV.

C. Max-margin Learning

Following Taskar et al. (2003), we choose the log-linear model for potential functions ψ^s and ψ^p,[6]

$$\log \psi^s(s^s_{ir}) \propto \theta_s^\top \phi_s(O_i, \ell_i, c_i, O_r, c_r) \tag{7}$$

where θ_s is the weight vector of the features and $\phi_s(\cdot)$ is the feature vector for stability. Let θ_p and $\phi_p(\cdot)$ denote the weight and features for semantic preference respectively. The discriminant function for a placing strategy is given by (from Eq. (5))

$$f(S,T) = \sum_{ir} \log \Psi(s_{ir}) + \sum_{ir} \log \Psi(t_{ir}) \tag{8}$$

In the learning phase, we use supervised learning to estimate θ_s and θ_p. The training data contains placements with ground-truth stability and semantic preference labels (we describe it later in Section VIII-A). By Eq. (6),

$$f(S,T) = \sum_{ir} \theta_s^T \phi_s(s^s_{ir}) + \theta_p^T \phi_p(s^p_{ir}) + \sum_{ir} \theta_s^T \phi_s(t^s_{ir}) + \theta_p^T \phi_p(t^p_{ir}) \tag{9}$$

A desirable θ_s and θ_p should be able to maximize the potential of a good placing strategy (S,T), i.e., for any different placing strategy (S',T'), $f(S,T) > f(S',T')$. Furthermore, we want to maximize this difference to increase the confidence in the learned model. Therefore, our objective is,

$$\arg\max_{\theta_s,\theta_p} \gamma \quad \text{s.t.} \quad f(S,T) - f(S',T') \geq \gamma, \quad \forall (S',T') \neq (S,T) \tag{10}$$

which, after introducing slack variables to allow errors in the training data, is equivalent to[7]

$$\arg\min_{\theta_s,\theta_p} \tfrac{1}{2}(||\theta_s||^2 + ||\theta_p||^2) + C \textstyle\sum (\xi^s_{s,ir} + \xi^p_{s,ir} + \xi^s_{t,ir} + \xi^p_{t,ir}) \tag{11}$$

$$\text{s.t. } s^s_{ir}(\theta_s^\top \phi_s(\cdot)) \geq 1 - \xi^s_{s,ir}, \quad s^p_{ir}(\theta_p^\top \phi_p(\cdot)) \geq 1 - \xi^p_{s,ir}, \tag{12}$$

$$t^s_{ir}(\theta_s^\top \phi_s(\cdot)) \geq 1 - \xi^s_{t,ir}, \quad t^p_{ir}(\theta_p^\top \phi_p(\cdot)) \geq 1 - \xi^p_{t,ir}, \forall i,r \tag{13}$$

We use max-margin learning (Joachims, 1999) to learn θ_s and θ_p respectively.

Note that the learning method in Section V-B for single-object placements is actually a special case of the above. Specifically, for one object and one placing area, the stacking and the semantic preference problems are trivially solved, and the subscripts i and r are not needed. Therefore, this equation reduces to Eq. (2) with θ_s, $\phi_s(\cdot)$ and t^s corresponding to θ, ϕ_i and y_i in Eq. (2).

[6]For conciseness, we use $\psi(s^s_{ir})$ to denote the full term $\psi^s(s^s_{ir}, O_i, \ell_i, c_i, O_r, c_r)$ explicitly in the rest of the paper unless otherwise clarified. We do the same for ψ^p.

[7]Here, without loss of the generality, we map the domain of S and T from {0,1} to {-1,1}.

D. Inference

Once we have learned the parameters in the graphical model, given the objects and placing environment, we need to find the placing strategy that maximizes the likelihood in Eq. (4) while satisfying the aforementioned constraints as well:

$$\sum_{i=1}^{n} s_{ir} \leq 1, \quad \forall r; \qquad \text{(chain stacking)} \tag{14}$$

$$\sum_{r=1}^{n} s_{ir} + \sum_{r=1}^{m} t_{ir} = 1, \quad \forall i; \qquad \text{(full-coverage)} \tag{15}$$

$$\ell_i \neq \ell_j, \quad \forall t_{ir} = t_{jr} = 1, \tag{16}$$

$$t_{ir} + t_{jr} \leq 1, \quad \forall O_i \text{ overlaps } O_j. \qquad \text{(non-overlap)} \tag{17}$$

Eq. (14) states that for every object r, there can be at most one object on its top. Eq. (15) states that every object i should be placed exactly at one place. Eq. (16) states that two objects cannot occupy the same location. Eq. (17) states that even if the objects are at different locations, they still cannot have any overlap with each other. In another word, if O_i placed at ℓ_i conflicts with O_j placed at ℓ_j, then only one of them can be placed. We do not need constrain s_{ir} since 'chain stacking' already eliminates this overlap issue.

Enforcing the acyclic property could be hard due to the exponential number of possible cycles in placing strategies. Expressing all the cycles as constraints is infeasible. Therefore, we assume a topological order on stacking: s_{ir} can be 1 only if O_r with configuration c_r does not have smaller projected 2D area on XY-plane than O_i with configuration c_i. This assumption is reasonable as people usually stack small objects on big ones in practice. This ensures that the optimal placing strategy is acyclic.

As mentioned before, the search space of placing strategies is too large for applying any exhaustive search to. Several prior works have successfully applied ILP to solve inference in Conditional Random Fields or Markov networks, e.g., Roth and Yih, 2005; Taskar et al., 2004; Yanover et al., 2006; Globerson and Jaakkola, 2007. Motivated by their approach, we formulate the inference (along with all constraints) as an ILP problem and then solve it by LP relaxation, which works well in practice.

We use random samples to discretize the continuous space of the location ℓ_i and configuration c_i.[8] We abuse the notation for S, T and use the same symbols for representing sampled variables. We use $s_{ijrtk} \in \{0,1\}$ to represent object O_i with the jth sampled configuration is placed on object O_r with the tth configuration at the kth location. Similarly, we use t_{ijrk} to represent placing object O_i in configuration j on top of E_r

[8]In our experiments, we randomly sample ℓ_i in about every 10cm $\times$ 10cm area. An object is rotated every 45 degree along every dimension, thus generating 24 different configurations.

TABLE V: Stability results for three baselines and our algorithm using different features (contact, caging, histograms and all combined) with two different kernels (linear and quadratic polynomial).

	dish-racks			stemware-holder			closet			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
chance	1.5	0.70	0.50	2.0	0.48	0.50	2.0	0.47	0.50	2.1	0.44	0.50	1.9	0.52	0.50
vert	1.3	0.79	0.74	2.0	0.70	0.77	2.5	0.63	0.71	1.0	1.00	1.00	1.7	0.78	0.80
hori	2.1	0.22	0.48	4.3	0.35	0.54	6.0	0.03	0.36	7.2	0.00	0.38	4.9	0.15	0.44
lin-contact	1.9	0.81	0.80	2.0	0.60	0.71	1.8	0.70	0.79	1.0	0.81	0.91	1.7	0.73	0.80
lin-caging	3.5	0.60	0.74	1.3	0.94	0.95	1.2	0.93	0.77	2.5	0.70	0.85	2.1	0.79	0.83
lin-hist	1.4	0.93	0.93	2.3	0.70	0.85	1.0	1.00	0.99	1.0	1.00	1.00	1.4	0.91	0.94
lin-all	1.2	0.91	0.91	2.0	0.80	0.86	1.0	1.00	1.00	1.0	1.00	1.00	1.3	0.93	0.94
poly-contact	1.2	0.95	0.88	1.0	0.80	0.85	1.2	0.93	0.93	1.2	0.91	0.88	1.1	0.90	0.89
poly-caging	2.1	0.81	0.87	2.0	0.60	0.79	1.0	0.97	0.94	2.2	0.70	0.88	1.8	0.77	0.87
poly-hist	1.2	0.94	0.91	3.0	0.30	0.67	1.0	1.00	1.00	1.0	1.00	1.00	1.6	0.81	0.90
poly-all	1.1	0.95	0.94	1.8	0.60	0.92	1.0	1.00	1.00	1.0	1.00	1.00	1.2	0.89	0.96

at location k. Now our problem becomes:

$$\begin{aligned}
&\arg\max_{S,T} \textstyle\sum_{ijrk}(t_{ijrk}\log\Psi(t_{ijrk}{=}1){+}(1{-}t_{ijrk})\log\Psi(t_{ijrk}{=}0))\\
&+\textstyle\sum_{ijrtk}(s_{ijrtk}\log\Psi(s_{ijrtk}{=}1){+}(1{-}s_{ijrtk})\log\Psi(s_{ijrtk}{=}0))\\
&\quad\text{s.t. } \textstyle\sum_{ijk} s_{ijrtk}{\le}\sum_{pqk} s_{rtpqk}{+}\sum_{pk} t_{rtpk},\ \forall r,t;\\
&\quad\textstyle\sum_{jrtk} s_{ijrtk}{+}\sum_{jrk} t_{ijrk}{=}1,\ \forall i;\\
&\quad\textstyle\sum_{ij} t_{ijrk}{\le}1,\ \forall r,k;\\
&\quad t_{ijrk}{+}t_{i'j'rk'}{\le}1,\ \forall t_{ijrk}\text{ overlaps }t_{i'j'rk'}.
\end{aligned}\tag{18}$$

While this ILP is provably NP-complete in the general case, for some specific cases it can be solved in polynomial time. For example, if all objects have to stack in one pile, then it reduces to a dynamic programming problem. Or if no stacking is allowed, then it becomes a maximum matching problem. For other general cases, an LP relaxation usually works well in practice. We use an open-sourced Mixed Integer Linear Programming solver (Berkelaar et al., 2004) in our implementation.

VIII. Experiments on Placing Multiple Objects

In these experiments, the input is the raw point-clouds of the object(s) and the scene, and our output is a placing strategy composed of the placing location and orientation for each object. Following this strategy, we can construct the point-cloud of the scene after placing (e.g., Fig. 1 top-right) and then use path planning to guide the robot to realize it.

We extensively tested our approach in different settings to analyze different components of our learning approach. In particular, we considered single-object placing, placing in a semantically appropriate area, multiple-object single-environment placing, and the end-to-end test of placing in offices and houses. We also tested our approach in robotic experiments, where our robots accomplished several placing tasks, such as loading a bookshelf and a fridge.

A. Data

We consider 98 objects from 16 categories (such as books, bottles, clothes and toys shown Table VI) and 40 different placing areas in total (e.g., dish-racks, hanging rod, stemware holder, shelves, etc). The robot observes objects using its Kinect sensor, and combines point-clouds from 5 views. However, the combined point-cloud is still not complete due to reflections and self-occlusions, and also because the object can be observed only from the top (e.g., see Fig. 1). For the environment, only a single-view point-cloud is used. We pre-processed the data and segmented the object from its background to reduce the noise in the point-clouds. For most of the placing areas,[9] our algorithm took real point-clouds. For *all the objects*, our algorithm took only real point-clouds as input and we did not use any assumed 3D model.

B. Placing Single New Objects with Raw Point-Cloud as Input

The purpose of this experiment is, similar to Section VI, to test our algorithm in placing a *new* object in a scene, but instead with raw point-clouds composing the training data. We test our algorithm on the following four challenging scenes:

1) dish-racks, tested with three dish-racks and six plates;
2) stemware holder, tested with one holder and four martini glass/handled cups;
3) hanging clothes, tested with one wooden rod and six articles of clothing on hangers;
4) cylinder holders, tested with two pen-holder/cup and three stick-shaped objects (pens, spoons, etc.).

Since there is only a single placing environment in every scene, only stability features play a role in this test. We manually labeled 620 placements in total, where the negative examples were chosen randomly. Then we used leave-one-out training/testing so that the testing object is always *new* to the algorithm.

We compare our algorithm with three heuristic methods:

- *Chance*. Valid placements are randomly chosen from the samples.
- *Vertical placing*. Placements are chosen randomly from samples where the object is vertical (the height is greater than the width). This is relevant for cases such as placing plates in dish-racks.
- *Horizontal placing*. Placements are chosen where the object is horizontal (opposed to 'vertical'). This applies to cases when the placing area is somewhat flat, e.g., placing a book on a table.

For the placing scenarios considered here, the heuristic based on finding flat surface does not apply at all. Since we do not know the upright orientation for all the objects and it is not

[9]For some racks and holders that were too thin to be seen by our sensor, we generated synthetic point-clouds from tri-meshes.

TABLE VI: Semantic results for three baselines and our algorithm using different features (BOW, color, curvature, Eigenvalues, Zernike and all combined) with two different kernels (linear and quadratic polynomial) on AUC metric.

	big bottles	books	boxes	clothes	comp.+ accssry	hats & gloves	dishware	food	misc. house.	martini glasses	mugs	shoes	small bottles	stationery	toys	utensils	AVG
chance	.50	.50	.50	.50	.50	.50	.50	.50	.50	.50	.50	.50	.50	.50	.50	.50	.50
table	.70	.61	.59	.90	.69	.78	.61	1.0	.65	.32	.45	.60	.64	.57	.66	.50	.66
size	.68	.67	.67	.80	.81	.80	.66	1.0	.83	.46	.52	.90	.70	.60	.82	.51	.72
lin-BOW	.90	1.0	.98	.08	.93	.96	.86	1.0	.91	.68	.89	.70	.96	.86	.94	.79	.85
lin-color	.75	1.0	1.0	.15	.96	.93	.59	1.0	.80	.64	.72	.45	.98	.95	.86	.61	.79
lin-curv.	.85	.90	.89	.78	.92	.85	.62	1.0	.74	.61	.67	.60	.86	.82	.83	.48	.79
lin-Eigen	.67	.93	.93	.40	.93	.95	.53	1.0	.87	.43	.47	.90	.89	.89	.96	.53	.79
lin-Zern.	.86	1.0	.91	.20	.86	.89	.73	.80	.83	.79	.74	.65	.80	.82	.84	.59	.77
lin-all	.93	1.0	1.0	.60	.93	.96	.84	1.0	.88	.71	.87	.50	.99	.89	.89	.82	.85
poly-BOW	.86	1.0	.89	.47	.90	.97	.90	1.0	.89	.82	.87	.75	.96	.85	.87	.82	.87
poly-all	.91	1.0	1.0	.62	.90	.99	.93	1.0	.89	.86	.89	.75	1.0	.89	.88	.84	.90

well-defined in some cases (e.g. an article of clothing on a hanger), we do not use the flat-surface-upright rule in these experiments.

Results are shown in Table V. We use the same three evaluation metrics as in Section VI-D: R_0, P@5 and AUC. The top three rows of the table shows our three baselines. Very few of these placing areas are 'flat', therefore the horizontal heuristic fares poorly. The vertical heuristic performs perfectly in placing in pen-holder (all vertical orientations would succeed here), but its performance in other cases is close to chance. All the learning algorithms perform better than all the baseline heuristics in the AUC metric.

For our learning algorithm, we compared the effect of different stability features as well as using linear and polynomial kernels. The results show that combining all the features together give the best performance in all metrics. The best result is achieved by polynomial kernel with all the features, giving an average AUC of 0.96.

C. Selecting Semantically Preferred Placing Areas

In this experiment, we test if our algorithm successfully learns the preference relationship between objects and their placing areas. Given a single object and multiple placing areas, we test whether our algorithm correctly picks out the most suitable placing area for placing that object. As shown in Table VI, we tested 98 objects from 16 categories on 11 different placing areas: the ground, 3 dish-racks, a utensil organizer, stemware-holder, table, hanging rod, pen-holder and sink. We exhaustively labeled every pair of object and area, and used leave-one-out method for training and test. Again, we build three baselines where the best area is chosen 1) by chance; 2) if it is a table (many of objects in our dataset can be placed on table); 3) by its size (whether the area can cover the object or not).

Our algorithm gives good performance in most object categories except clothes and shoes. We believe this is because clothes and shoes have a lot of variation making it harder for our algorithm to learn. Another observation is that the heuristic 'table' performs well on clothes. We found that this is because, in our labeled data set, the table was often labeled as the second best placing area for clothes after the hanging rod in the closet. We tested different semantic features with linear SVM and also compared linear and polynomial SVM on all features combined. In comparison to the best baseline of 0.72, we get an average AUC of 0.90 using polynomial SVM with all the features.

D. Multiple Objects on Single Area

In this section, we consider placing multiple objects in a very limited space: 14 plates in one dish-rack without any stacking; 17 objects including books, dishware and boxes on a small table so that stacking is needed; and five articles of clothing with hangers on a wooden rod. This experiment evaluates stacking and our LP relaxation.

The results are shown in Fig. 10. In the left image, plates are vertically placed without overlap with spikes and other plates. Most are aligned in one direction to achieve maximum loading capacity. For the second task in the middle image, four piles are formed where plates and books are stacked separately, mostly because of semantic features. Notice that all the dishware, except the top-most two bowls, is placed horizontally. The right image shows all clothes are placed perpendicular upon the rod. However the first and last hangers are little off the rod due to the outliers in the point-cloud which are mistakenly treated as part of the object . This also indicates that in an actual robotic experiment, the placement could fail if there is no haptic feedback.

E. Placing in Various Scenes

In this experiment, we test the overall algorithm with multiple objects being placed in a scene with multiple placing areas. We took point-clouds from 3 different offices and 2 different apartments. Placing areas such as tables, cabinets, floor, and drawers were segmented out. We evaluated the quality of the final placing layout by asking two human subjects (one male and one female, not associated with the project) to label placement for each object as stable or not, semantically correct or not, and also report a qualitative metric score on how good the overall placing was (0 to 5 scale).

Table VII shows the results for different scenes (averaged over objects and placing areas) and for different algorithms and baselines. We considered baselines where objects were

Fig. 10: Placing multiple objects on a dish-rack (left), a table (middle) and a hanging rod (right). Most objects are placed correctly, such as the plates vertically in the dish-rack, books and plates stacked nicely on the table and the hangers with the clothes aligned on the rod. However, two top-most plates on the table are in wrong configuration and the right-most hanger in the right figure is off the rod.

placed *vertically*, *horizontally* or according to configuration *priors* (e.g., flat surface prefers horizontal placing while dish-racks and holders prefer vertical placing). As no semantic cues were used in the baselines, placing areas were chosen randomly. The results show that with our learning algorithm, the end-to-end performance is substantially improved under all metrics, compared to the heuristics. Fig. 11 shows the point-cloud of two offices after placing the objects according to the strategy. We have marked some object types in the figure for better visualization. We can see that books are neatly stacked on the left table, and horizontally on the couch in the right image. While most objects are placed on the table in the left scene, some are moved to the ground in the right scene, as the table there is small.

We do not capture certain properties in our algorithm, such as object-object co-occurrence preferences. Therefore, some placements in our results could potentially be improved in the future by learning such contextual relations (e.g., Anand et al., 2011). For example, a mouse is typically placed on the side of a keyboard when placed on a table and objects from the same category are often grouped together.

F. Robotic Experiment

We tested our approach on both our robots, the PANDA and POLAR, with the Kinect sensor. We performed the end-to-end experiments as follows. (For quantitative results on placing single objects, see Section VI-F.)

In the first experiment, our goal was to place 16 plates in a dish-rack. Once the robot inferred the placing strategy (an example from the corresponding offline experiment is shown in Fig. 10-left), it placed the plates one-by-one in the dish-rack (Fig. 13a). Out of 5 attempts (i.e., a total of 5x16=80 placements), less than 5% of placements failed. This is because the plate moved within the gripper after grasping.

In the second experiment, our aim was to place a cup, a plate and a martini glass together on a table and then into a dish-rack. The objects were originally placed at pre-defined locations and were picked up using the given grasps. The result is shown in Fig. 13b and 13c.

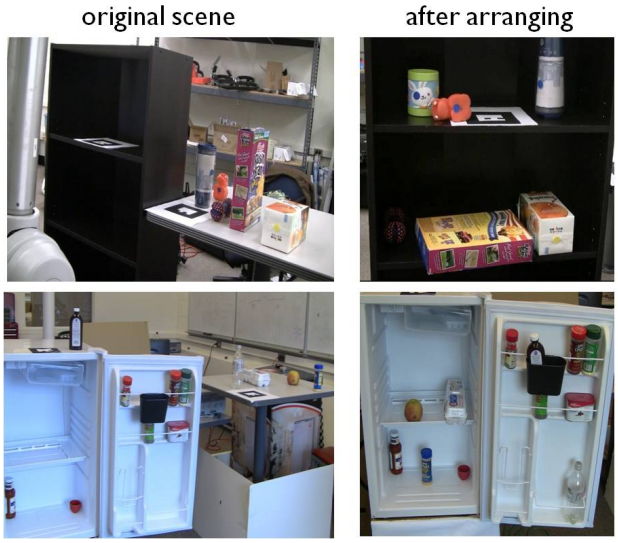


Fig. 12: Loading a bookshelf (top) and a fridge (bottom). Snapshots of the scenes before and after the arrangement by our robot POLAR using our learning algorithm.

In the third experiment, the robot was asked to arrange a room with a bookshelf, a fridge, a recycle-bin and a whiteboard as potential placing areas. The room contained 14 objects in total, such as bottles, rubber toys, a cereal box, coffee mug, apple, egg carton, eraser, cup, juice box, etc. Our robot first inferred the placing strategy and then planned a path and placed each one. In some cases when the object was far from its placing location, the robot moved to the object, picked it up, moved to the placing area and then placed it. For more details on the system, see Section III. Fig. 13 (last row) shows some placing snapshots and the scene before and after placing is shown in Fig. 12.

Videos of our robot placing these objects are available at: `http://pr.cs.cornell.edu/placingobjects`

TABLE VII: End-to-end test results. In each scene, the number of placing areas and objects is shown as $(\cdot, \cdot)$. **St**: % of stable placements, **Co**: % of semantically correct placements, **Sc**: average score (0 to 5) over areas.

	office-1 (7,29)			office-2 (4,29)			office-3 (5,29)			apt-1 (13,51)			apt-2 (8,50)			Average		
	St	Co	Sc	St	Co	Sc	St	Co	Sc	St	Co	Sc	St	Co	Sc	St	Co	Sc
Vert.	36	60	2.4	33	52	2.4	52	59	2.8	38	41	1.9	46	59	3	39	59	2.4
Hori.	50	62	2.9	57	52	2.7	69	60	3.4	54	50	2.5	60	53	2.7	57	58	2.7
Prior	45	52	2.4	64	59	2.8	69	60	3.5	44	43	2.3	61	55	2.7	55	53	2.7
Our approach	78	79	4.4	83	88	4.9	90	81	4.5	81	80	3.8	80	71	4.2	83	82	4.4

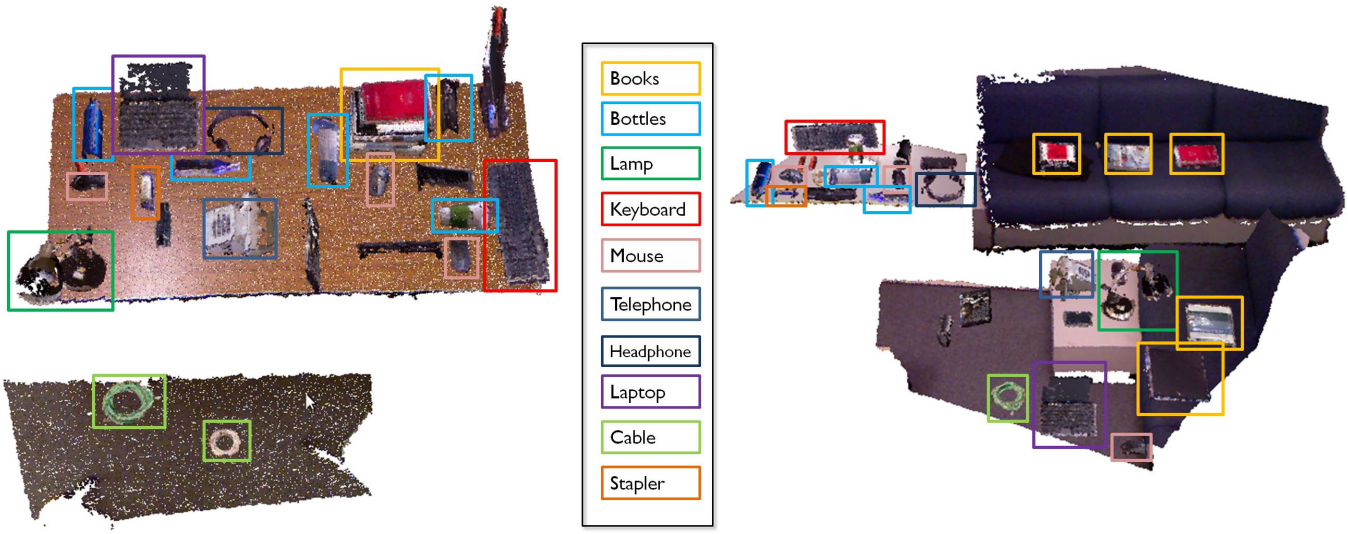


Fig. 11: Two office scenes after placing, generated by our algorithm. Left scene comprises a table and ground with several objects in their final predicted placing locations. Right scene comprises two tables, two couches and ground with several objects placed.

(a) (b) (c)

(d) (e) (f)

Fig. 13: Robots placing multiple objects in different scenes. Top row shows PANDA: (a) loading a dish-rack with plates, (b) placing different objects on a table, and (c) placing different objects in a dish-rack. Bottom row shows POLAR: (d) placing six items on a bookshelf, (e) loading five items in a fridge, and (d) placing an empty juice box in a recycle bin.

IX. Conclusion and Future Work

We considered the problem of placing objects in a scene with multiple placing areas, in particular, **where** and **how** to place each object when **stacking** one on another is allowed. We designed and used appropriate features that captured both stability and semantic preferences in placing. We proposed a graphical model that encoded these three properties and certain constraints to force the placing strategy to be feasible. Inference was expressed as an ILP problem, which was solved using an efficient LP relaxation. Our extensive experiments demonstrated that our algorithm could place objects stably and reasonably most of the time. In the end-to-end test of placing multiple objects in different scenes, our algorithm improved the performance significantly compared to the best baseline. Quantitatively, we achieved an average accuracy of 83% for stable placements and 82% for semantically meaningful placements. In our robotic experiments, the robots successfully placed new objects in new placing areas stably with a 82% success rate and 72% when considering semantically preferred orientations as well. However, if the robot had seen the object before (and its 3D model was available), these numbers were 98% for both stable as well as preferred placements. We also used out algorithm on our robots for accomplishing the tasks of loading items into dish-racks, a bookshelf, a fridge, etc.

In this work, we have captured only limited contextual information. Incorporating such information in future work would allow an algorithm to place the objects more meaningfully. Currently, we have only focussed on using visual and shape features for placing new objects. However, there are several other attributes based on material parameters such as surface friction and fragility that may be relevant for determining good placements. We believe that considering these attributes is an interesting direction for future work. Finally, incorporating haptic and force sensing while placing would also improve performance, especially when placing objects in confined spaces such as refrigerators.

References

Anand, A., Koppula, H. S., Joachims, T., and Saxena, A. (2011). Semantic labeling of 3d point clouds for indoor scenes. In *NIPS*.

Berenson, D., Diankov, R., Nishiwaki, K., Kagami, S., and Kuffner, J. (2009). Grasp planning in complex scenes. In *Int'l conf Humanoid Robots*.

Berg, A., Berg, T., and Malik, J. (2005). Shape matching and object recognition using low distortion correspondences. In *CVPR*.

Berkelaar, M., Eikland, K., and Notebaert, P. (2004). lp_solve 5.5, open source (mixed-integer) linear programming system. Software.

Bicchi, A. and Kumar, V. (2000). Robotic grasping and contact: a review. In *International Conference on Robotics and Automation (ICRA)*.

Brook, P., Ciocarlie, M., and Hsiao, K. (2011). Collaborative grasp planning with multiple object representations. In *ICRA*.

Ciocarlie, M. T. and Allen, P. K. (2008). On-line interactive dexterous grasping. In *Eurohaptics*.

Coffman Jr, E., Garey, M., and Johnson, D. (1996). Approximation algorithms for bin packing: A survey. In *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Co.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Deng, J., Berg, A., and Fei-Fei, L. (2011). Hierarchical semantic indexing for large scale image retrieval. In *CVPR*.

Diankov, R. and Kuffner, J. (2008). Openrave: A planning architecture for autonomous robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*.

Diankov, R., Srinivasa, S., Ferguson, D., and Kuffner, J. (2008). Manipulation planning with caging grasps. In *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, pages 285–292. IEEE.

Divvala, S., Hoiem, D., Hays, J., Efros, A., and Hebert, M. (2009). An empirical study of context in object detection. *CVPR*.

Dogar, M. and Srinivasa, S. (2011). A framework for push-grasping in clutter. In *RSS*.

Dy, J. and Brodley, C. (2004). Feature selection for unsupervised learning. *The Journal of Machine Learning Research*, 5:845–889.

Edsinger, A. and Kemp, C. (2006). Manipulation in human environments. In *Int'l Conf Humanoid Robots*.

Fergus, R., Perona, P., and Zisserman, A. (2003). Object class recognition by unsupervised scale-invariant learning. In *CVPR*.

Ferguson, M. (2011). Improved ar markers for topological navigation. *ROS 3D Contest*.

Fisher, M. (1981). The lagrangian relaxation method for solving integer programming problems. *Management science*, pages 1–18.

Fisher, M. and Hanrahan, P. (2010). Context-based search for 3d models. *ACM TOG*, 29(6).

Fisher, M., Savva, M., and Hanrahan, P. (2011). Characterizing structural relationships in scenes using graph kernels. *SIGGRAPH*.

Fu, H., Cohen-Or, D., Dror, G., and Sheffer, A. (2008). Upright orientation of man-made objects. *ACM Trans. Graph.*, 27:42:1–42:7.

Gelfand, N., Mitra, N. J., Guibas, L. J., and Pottmann, H. (2005). Robust global registration. In *Eurographics Symposium on Geometry Processing*.

Globerson, A. and Jaakkola, T. (2007). Fixing max-product: Convergent message passing algorithms for map lp-relaxations. In *NIPS*.

Glover, J., Rusu, R., and Bradski, G. (2011). Monte carlo pose estimation with quaternion kernels and the bingham distribution. In *Proceedings of Robotics: Science and Systems*.

Hanley, J. and McNeil, B. (1982). The meaning and use of the area under a receiver operating (roc) curvel characteristic. *Radiology*, 143(1):29–36.

Hedau, V., Hoiem, D., and Forsyth, D. (2009). Recovering the spatial layout of cluttered rooms. In *ICCV*.

Heitz, G., Gould, S., Saxena, A., and Koller, D. (2008). Cascaded classification models: Combining models for holistic scene understanding. In *Neural Information Processing Systems (NIPS)*.

Ho, E. S. L., Komura, T., and Tai, C.-L. (2010). Spatial relationship preserving character motion adaptation. *ACM Trans. Graph.*, 29(4).

Hsiao, K., Nangeroni, P., Huber, M., Saxena, A., and Ng, A. Y. (2009). Reactive grasping using optical proximity sensors. In *ICRA*.

Jain, A. and Kemp, C. (2010). Pulling open doors and drawers: Coordinating an omni-directional base and a compliant arm with equilibrium point control. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1807–1814. IEEE.

Jalali, A., Ravikumar, P., Sanghavi, S., and Ruan, C. (2010). A Dirty Model for Multi-task Learning. *NIPS*.

Jetchev, N. and Toussaint, M. (2011). Task space retrieval using inverse feedback control. In *ICML*, pages 449–456.

Jiang, Y., Amend, J., Lipson, H., and Saxena, A. (2012a). Learning hardware agnostic grasps for a universal jamming gripper. In *ICRA*.

Jiang, Y., Moseson, S., and Saxena, A. (2011a). Efficient Grasping from RGBD Images: Learning using a new Rectangle Representation. In *ICRA*.

Jiang, Y., Zheng, C., Lim, M., and Saxena, A. (2011b). Learning to

place new objects. In *Workshop on Mobile Manipulation: Learning to Manipulate*.

Jiang, Y., Zheng, C., Lim, M., and Saxena, A. (2012b). Learning to place new objects. In *ICRA*.

Joachims, T. (1999). *Making large-Scale SVM Learning Practical*. MIT-Press.

Johnson, A. and Hebert, M. (1999). Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE PAMI*, 21(5):433–449.

Katz, D. and Brock, O. (2008). Manipulating articulated objects with interactive perception. In *ICRA*.

Klingbeil, E., Saxena, A., and Ng, A. Y. (2010). Learning to open new doors. In *IROS*.

Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. The MIT Press.

Lai, K., Bo, L., Ren, X., and Fox, D. (2011). Sparse distance learning for object recognition combining rgb and depth information. *ICRA*.

Le, Q. V., Kamm, D., Kara, A., and Ng, A. Y. (2010). Learning to grasp objects with multiple contact points. In *ICRA*.

Leibe, B., Leonardis, A., and Schiele, B. (2004). Combined object categorization and segmentation with an implicit shape model. In *Workshop on Statistical Learning in Computer Vision, ECCV*, pages 17–32.

Li, C., Kowdle, A., Saxena, A., and Chen, T. (2010). Towards holistic scene understanding: Feedback enabled cascaded classification models. In *Neural Information Processing Systems (NIPS)*.

Li, C., Saxena, A., and Chen, T. (2011). θ-mrf: Capturing spatial and semantic structure in the parameters for scene understanding. In *NIPS*.

Liebelt, J., Schmid, C., and Schertler, K. (2008). Viewpoint-independent object class detection using 3d feature maps. In *CVPR*.

Liu, H. and Yu, L. (2005). Toward integrating feature selection algorithms for classification and clustering. *Knowledge and Data Engineering, IEEE Transactions on*, 17(4):491–502.

Lodi, A., Martello, S., and Vigo, D. (2002). Heuristic algorithms for the three-dimensional bin packing problem. *European Journal of Operational Research*, 141(2):410–420.

Lozano-Pérez, T., Jones, J., Mazer, E., and O'Donnell, P. (2002). Task-level planning of pick-and-place robot motions. *Computer*, 22(3):21–29.

Maitin-Shepard, J., Cusumano-Towner, M., Lei, J., and Abbeel, P. (2010). Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *ICRA*, pages 2308–2315. IEEE.

Miller, A. T., Knoop, S., Christensen, H. I., and Allen, P. K. (2003). Automatic grasp planning using shape primitives. In *ICRA*, pages 1824–1829.

Nguyen, V. (1986). Constructing stable force-closure grasps. In *ACM Fall joint computer conf.*

Novotni, M. and Klein, R. (2003). 3d zernike descriptors for content based shape retrieval. In *ACM symp. Solid modeling and applications*.

Pisinger, D. and Sigurd, M. (2007). Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. *INFORMS Journal on Computing*, 19(1):36–51.

Ponce, J., Stam, D., and Faverjon, B. (1993). On computing two-finger force-closure grasps of curved 2D objects. *IJRR*, 12(3):263.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. (2009). Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*.

Rao, D., Le, Q., Phoka, T., Quigley, M., Sudsang, A., and Ng, A. (2010). Grasping Novel Objects with Depth Segmentation. In *IROS*.

Rosales, C., Porta, J., and Ros, L. (2011). Global optimization of robotic grasps. *RSS*.

Roth, D. and Yih, W. (2005). Integer linear programming inference for conditional random fields. In *ICML*.

Rusinkiewicz, S. and Levoy, M. (2001). Efficient variants of the icp algorithm. In *3DIM*.

Rusu, R., Marton, Z., Blodow, N., and Beetz, M. (2008). Learning informative point classes for the acquisition of object model maps. In *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*, pages 643–650. IEEE.

Rusu, R. B., Blodow, N., and Beetz, M. (2009). Fast point feature histograms (fpfh) for 3d registration. In *ICRA*.

Rusu, R. B. and Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China.

Savarese, S. and Fei-Fei, L. (2007). 3d generic object categorization, localization and pose estimation. *IJCV*.

Saxena, A., Chung, S. H., and Ng, A. Y. (2008a). 3-d depth reconstruction from a single still image. *International Journal of Computer Vision (IJCV)*, 76(1):53–69.

Saxena, A., Driemeyer, J., Kearns, J., and Ng, A. Y. (2006). Robotic grasping of novel objects. In *NIPS*.

Saxena, A., Driemeyer, J., and Ng, A. (2008b). Robotic grasping of novel objects using vision. *IJRR*, 27(2):157.

Saxena, A., Driemeyer, J., and Ng, A. Y. (2009a). Learning 3-d object orientation from images. In *ICRA*.

Saxena, A., Sun, M., and Ng, A. (2009b). Make3d: Learning 3d scene structure from a single still image. *IEEE transactions on pattern analysis and machine intelligence (PAMI)*.

Saxena, A., Wong, L., and Ng, A. Y. (2008c). Learning grasp strategies with partial shape information. In *AAAI*.

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

Sturm, J., Konolige, K., Stachniss, C., and Burgard, W. (2010). 3d pose estimation, tracking and model learning of articulated objects from dense depth video using projected texture stereo. In *RSS*.

Sugie, H., Inagaki, Y., Ono, S., Aisu, H., and Unemi, T. (1995). Placing objects with multiple mobile robots-mutual help using intention inference. In *ICRA*.

Taskar, B., Chatalbashev, V., and Koller, D. (2004). Learning associative markov networks. In *ICML*.

Taskar, B., Guestrin, C., and Koller, D. (2003). Max-margin markov networks. In *NIPS*.

Thomas, A., Ferrar, V., Leibe, B., Tuytelaars, T., Schiel, B., and Van Gool, L. (2006). Towards multi-view object class detection. In *CVPR*.

Torralba, A., Murphy, K., and Freeman, W. T. (2010). Using the forest to see the trees: object recognition in context. *Communications of the ACM, Research Highlights*, 53(3):107–114.

Toussaint, M., Plath, N., Lang, T., and Jetchev, N. (2010). Integrated motor control, planning, grasping and high-level reasoning in a blocks world using probabilistic inference. *ICRA*.

Winn, J., Criminisi, A., and Minka, T. (2005). Object categorization by learned universal visual dictionary. *ICCV*.

Xiong, X. and Huber, D. (2010). Using context to create semantic 3d models of indoor environments. In *BMVC*.

Yanover, C., Meltzer, T., and Weiss, Y. (2006). Linear programming relaxations and belief propagation–an empirical study. *The Journal of Machine Learning Research*, 7:1887–1907.