Flexible User Specification of Perceptual Landmarks for Robot Manipulation

Justin Huang and Maya Cakmak

Abstract— **Programming robots to do manipulation tasks requires users to specify relevant perceptual landmarks, which include objects, parts of objects, or parts of the workspace. While many techniques have been developed for object detection, few are designed to detect arbitrary parts of objects or of the workspace. This paper presents *CustomLandmarks*, a flexible tool that lets non-roboticists build their own perceptual detectors for many kinds of landmarks. The system components include a simple 3D interface for specifying landmarks, a novel representation for landmarks, and an algorithm for locating landmarks in new scenes. We evaluate the system's detection performance through systematic experiments and by using the system to aid a PR2 robot with several manipulation tasks. Finally, we present a user study showing that novices to the system are able to understand and use CustomLandmarks quickly, creatively, and effectively.**

I. Introduction

A robot's ability to perform useful manipulation tasks strongly depends on its ability to robustly perceive task-relevant "landmarks," *i.e.,* parts of objects or the scene relevant for the task. Programming robots to do manipulation tasks typically entails the development of custom, task-specific perception software. Adjusting such software to minor task modifications often require additional labor from experts in robotics or computer vision. While researchers have tried to lift that burden by developing general-purpose object detectors, these advancements are not designed to detect *parts* of objects or of the scene. Another approach is to attach visual markers that the robot can easily locate ("fiducials") to parts of the scene. However, attaching markers to all landmarks is not always possible or desirable, such as when the landmark's position can frequently change.

In this paper, we address the challenge of enabling non-expert, novice users to specify arbitrary perceptual landmarks. Having solutions to this problem could make manipulation task programming faster and cheaper. To that end, we introduce *CustomLandmarks*, a system that lets users define partial 3D models of landmarks from sensor data and locate them in new scenes. A key contribution of our system is its representation of a landmark, which captures not only its geometry, but also nearby space that is expected to be empty. This representation is flexible and can be leveraged in creative ways. For example, to visually check whether the robot has successfully grasped a thin, lightweight object, a user can create a landmark of the robot's gripper with a region of empty space beneath it and program the robot to search for this landmark. We also present an algorithm for locating these landmarks that behaves predictably and is simple for users to understand.

The landmark representation requires users to adopt a different way of thinking about object and workspace detection, and a natural question to ask is whether novice users are capable of using landmarks in this way. A second contribution of our work is to show, through a user evaluation of *CustomLandmarks*, that users are capable of using the system in non-obvious ways after minimal training.

In the rest of this paper, we first discuss related work. In Section III, we describe the representation of a landmark and give an algorithm for locating them in new scenes. Section IV characterizes the performance of the system on a variety of case studies. We also demonstrate how we used the system to perform a variety of manipulation tasks on a PR2 robot in Section V. Finally, Section VI presents the design and results of our user study.

II. Related work

Our work presents an interactive system for defining visual 3D landmarks for manipulation. In this section, we relate our work to past research on automatic vision systems for manipulation, user-guided perception, and landmarks for programming by demonstration.

A. Object detection and pose estimation

Although the landmarks represented by our system are not limited to objects, object detection is an important and closely related area of research, with a vast body of literature behind it. Most techniques compute alignments between two point clouds by repeatedly matching 3D features sampled from the point clouds using different features or speed optimizations [1], [2], [3]. Our approach differs in that our landmark representation also specifies areas of empty space around the point cloud. This allows users to make distinctive landmarks out of otherwise generic point cloud segments. Many systems require full 3D object models as input, either from CAD models or from a scan of the object [4], [5]. This can improve detection performance but make the system harder to use and deploy. Because usability is one of the primary goals of our research, we only record object models from a single viewpoint.

Localizing landmarks can also be viewed as an object recognition or representation learning problem, which has seen major progress in recent 2D computer vision research [6]. However, these systems recognize only a predefined set of objects and require large amounts of training data to work well, which non-experts might not have the time or

The authors are with Computer Science and Engineering Department, University of Washington, Seattle, WA 98195, USA.

ability to assemble. Additional work has gone into learning representations of 3D shapes using deep neural networks [7], [8]. These representations could be used to compare sampled volumes from the scene to a landmark. However, the current resolution of these voxel grid representations (up to $32 \times 32 \times 32$ voxels) may be too limited to represent most landmarks. Another approach is to use these representations to find correspondences between the landmark and the scene as part of a larger detection framework [9].

B. *Vision for manipulation*

One of the most common tasks for robot manipulation is grasping. To this end, researchers have developed different techniques for inferring grasp points using vision. Many approaches use probabilistic models [10], [11], including neural networks [12], [13], [14]. All of these approaches are specifically designed for the purpose of grasping objects and cannot be applied to other perception tasks such as locating parts of the workspace.

C. *User-guided vision*

Our approach uses human input to create landmarks. Other researchers have looked at how robots can learn about object parts from human demonstrations. For example, Hsaio et al. explored how robots can learn how to grasp objects by comparing them to objects grasped by a teleoperator in simulation [15]. Herzog et al. extended this approach by inferring grasp templates from demonstrations [16].

Outside of robotic manipulation, researchers have investigated using human input to aid with vision tasks. In particular, a common vision task that makes use of user input is interactive image segmentation, in which the foreground of a 2D image is separated from the background [17], [18]. This approach could be used to define landmarks from a 2D image of the scene. However, landmarks representing parts of the workspace may frequently appear to be the in background of an image, so most techniques for background subtraction are not likely to work.

D. *Landmarks for programming by demonstration*

In Section V, we show how our system can be used in conjunction with programming by demonstration (PbD). A widely used technique, PbD ships with robots like Kuka's LWR, ABB's YuMi, or Rethink's Baxter. Previous research on PbD has not focused on scene understanding, and has instead used fiducials [19], [20], marker-based motion capture [21], or simulated environments [22]. Others have used limited or special-purpose perceptual systems such as object detectors on flat, uncluttered tabletop surfaces [23], [24]. Our work improves on the perception capabilities of such PbD systems by detecting landmarks in arbitrary scenes.

III. CustomLandmarks

CustomLandmarks is a system for specifying and locating 3D landmarks from RGBD point cloud data. The system consists of three key components: the landmark representation, a user interface for creating landmarks, and an algorithm for locating landmarks in new scenes.

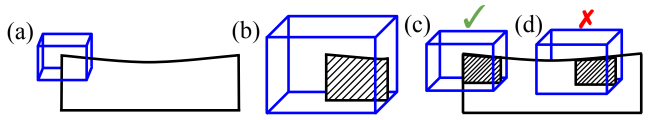


Fig. 1: An example that illustrates custom landmark specification and search. (a) A landmark representing the top left corner of a flat vertical area of the robot's workspace (e.g., a plastic divider) is selected by drawing a box around the corner. The top and left halves of the box are empty space. (b) A close-up view of the landmark. The shaded region represents the captured point cloud. (c) The corner landmark matches well with the corner of another divider. (d) However, it will not match well when aligned with the top middle of the divider. Although all the points of the landmark (shaded region) match well with the divider, part of the scene intrudes upon the empty space of the landmark box.

A. *Landmark representation*

Our system represents landmarks with a point cloud, which captures the shape of the landmark, as well as a box that surrounds the point cloud. The boundaries of the box specify the region of empty space around the landmark that is expected be unoccupied. Fig. 1 illustrates the landmark representation and use of empty space, and Section IV explores the consequences of this representation in detail.

B. *User interface*

The user interface for creating landmarks shows a point cloud view from the robot's depth sensor, as well as a 3D box-shaped selector, with controls to set its position and dimensions. A screenshot of the box selector is shown in Section VI, Fig. 7. With the landmark placed in the robot's view, the user moves and resizes the box to surround the landmark of interest, potentially including margins of empty space on the side. When the landmark is saved, the system records to a database the subset of the scene within the box, as well as the pose and dimensions of the box. For convenience, the system also saves the entirety of the scene, so that the landmark's box can be edited offline using the same scene. We chose to make the selector shaped like a box and aligned with the base of the robot for simplicity to the user and to the implementation. However, a more advanced interface could be used to define an arbitrary shape and orientation for the landmark. The landmark is captured from a single view from one or more point clouds, and does not include color information. In our implementation, we fused five separate point cloud readings from the same static scene to reduce noise in the data.

C. *Landmark search algorithm*

To make use of custom landmarks, we need a way of localizing them in a new scene. We first formally describe the inputs and outputs. A point p is a location vector $p = (p_x, p_y, p_z)$ and a scene S is simply a set of points. A landmark ℓ is a tuple (P, B), where P is a set of points

Algorithm 1: FindLandmark

```
Input  : scene, landmark, miscellaneous parameters
Output: a list of aligned landmarks in the scene, or
        empty list if not found
crop scene;
downsample scene using a voxel grid;
samples = sample points from scene using a voxel grid,
  leaf size of 1/3 of the landmark's dimensions ;
candidates = [];
for (sample in samples) {
    c = copy of landmark;
    move c such that c.cloud is centered on sample;
    c = run ICP to align c.cloud with scene;
    c.error = CandidateError (c, scene);
    add c to candidates;
}
remove all candidates that do not have the lowest error
  within a certain radius (non-max suppression);
output = $\{c \in candidates \mid c.error < threshold\}$;
return output;
```

Algorithm 2: CandidateError

```
   Input  : scene, candidate landmark
   Output: error score
1  error = 0;
2  denominator = 0;
3  croppedScene = scene cropped to candidate.box;
4  visited = [];
5  for (scenePt in croppedScene) {
6      candidatePt = nearest point in candidate to scenePt;
7      error += distance between scenePt and candidatePt;
8      denominator += 1;
9      add candidatePt to visited;
10 }
11 for (candidatePt in candidate.cloud) {
12     if (candidatePt not in visited) {
13         scenePt = nearest point in scene to candidatePt;
14         error += distance between scenePt and
             candidatePt;
15         denominator += 1;
16     }
17 }
18 return error / denominator;
```

(*i.e.*, points in the point cloud selected by the user) and B represents a box, specified by a 6-dimensional pose and a 3-dimensional size vector.

Our search algorithm takes as input a scene S, represented by the complete point cloud captured before an execution, and a landmark ℓ to be localized in that scene. It outputs a set of landmarks O, where each landmark $\ell_o \in O$ is potentially an instance of the input landmark ℓ in the scene S.

Pseudocode for the algorithm is given in Algorithm 1. First, we crop and downsample the scene according to application parameters.[1] Then, we sample scene points and initialize an instance of ℓ at each sampled point. The sampling is done by downsampling the scene again using a voxel grid, which ensures that the entire scene is systematically searched. The leaf size of this voxel grid is set to be a fraction of the dimensions of the landmark (we found 1/3 to be a good value). Next, we run the iterative closest point (ICP) algorithm [25] to align the landmark's point cloud P with S, which produces a modified landmark ℓ'. For each landmark ℓ', we compute an error metric (Algorithm 2) using ℓ' and S. We then perform non-max suppression so that we do not produce duplicates of the same result. Finally, we filter the results by thresholding on the error metric.

The error metric (Algorithm 2) can be thought of as summing two measurements of error. The first measurement is of how well the landmark's shape matches with the scene, while the second is of how much of the scene intrudes on the empty space in the landmark's box. Formally, the first measurement is the sum of the distances between points on the landmark and their nearest points in the scene (lines 11-17). The second measurement is the sum of the distances between points of the scene (within the landmark box) and their nearest points on the landmark (lines 5-10). Adding a margin of empty space around the landmark helps eliminate false positive matches. If scene points are found where there is expected to be empty space, then they will increase the mean error, since the nearest points on the landmark are far away. This process is illustrated in Fig. 1(c) and 1(d).

IV. Case studies

In this section, we show how we used *CustomLandmarks* for 15 separate tasks and quantify the performance of our landmark search algorithm.

A. Non-object landmarks

CustomLandmarks can be used to model a variety of non-object landmarks. Below, we describe five categories of such landmarks and provide examples of how they can be used. Fig. 2 illustrates these examples.

1) Parts of an object: Sometimes it is helpful to model just part of an object as opposed to the entire object. For example, in a stack of bowls, only the rim of the top bowl may be visible, especially if the bowls are located above eye level on a kitchen shelf. For this case, we created a landmark of the bowl rim, which allows us to locate the top bowl (Fig. 2a).

Multiple objects may share an identical part. For example, researchers have developed 3D-printed adaptors that robots can securely grasp [26]. These adaptors can be attached to many kinds of tools, including non-rigid tools like feather dusters that would be hard to model in 3D. We created

[1] For this paper, the scene was cropped to a volume in front of the robot roughly equal to the reach of its arms. The scene was downsampled to a leaf size of 1 centimeter. The non-max suppression radius was computed to be half of the longest dimension of the landmark. We varied the threshold for our error metric for experimental purposes but generally use a value of 0.75 centimeters.

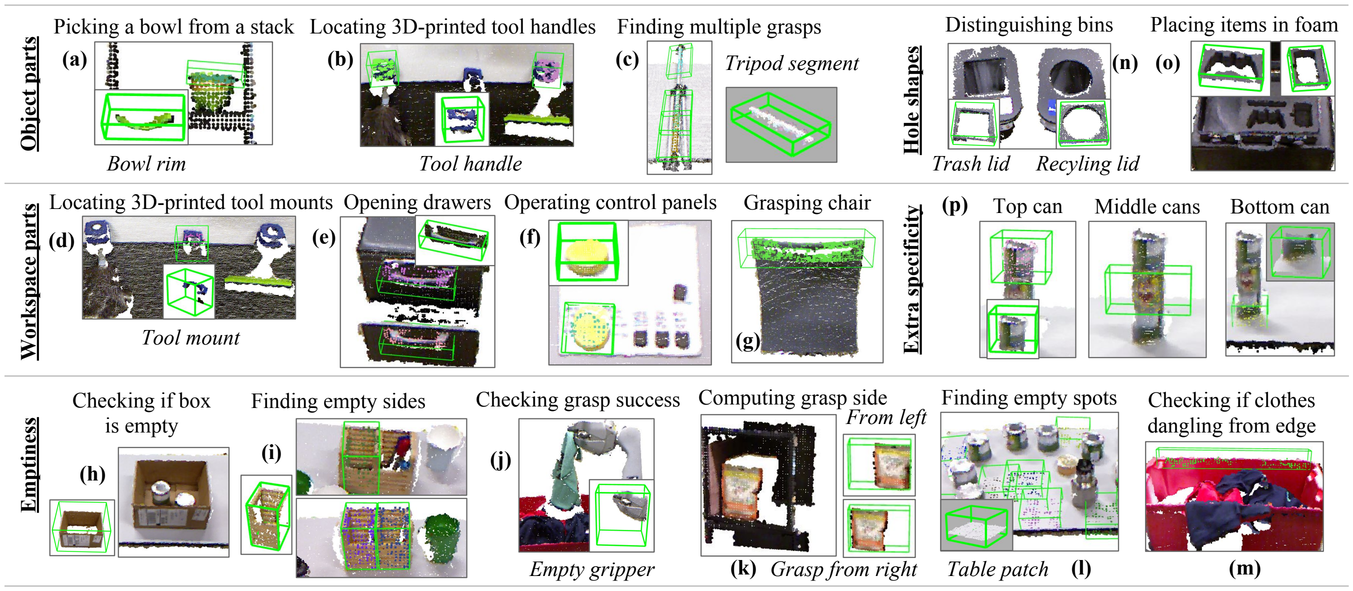


Fig. 2: Scenes and landmarks described in the case studies of Section IV-A. Five categories of landmarks are illustrated above: object parts, workspace parts, emptiness, hole shapes, and extra specificity. For each case study, we show one of the scenes we evaluated and the landmark or landmarks involved. The scenes also show examples of detections our system found, shown with green boxes.

a landmark of one of those adaptors to locate the tools (Fig. 2b).

Finally, users can specify parts of objects that serve as grasp points. Finding multiple grasp points is helpful because some parts of the object may be out of the robot's reach. Fig. 2(c) shows how we used a small segment of a tripod to find multiple grasp points on the tripod.

2) Parts of the workspace: Landmarks can also represent parts of the scene or of large objects. Examples include the handles of a drawer or the top of a chair (Fig. 2e, g). We also made a landmark of a wall mount for a tool to help a robot hang a tool in an available space (Fig. 2d).

Other workspace landmarks can serve as anchor points for the rest of the workspace. For example, on a mockup of a control panel, we localized the buttons, which are too flat to be landmarks themselves, by creating a landmark of a nearby prominent feature—in this case, a large dial (Fig. 2f).

As with objects, it can be useful to locate just part of the workspace. For example, landmarks of table corners can be used to locate the corners of many different-sized tables.

3) Emptiness: Many useful checks can be programmed by creating and searching for landmarks that model emptiness in some way. A simple use of emptiness in landmarks is to check whether a container or part of a container is empty. In one example, we created a landmark of an empty cardboard box so the robot could check if it was empty (Fig. 2h). In another example (Fig. 2i), we made a landmark of an empty side of a two-sided pencil holder, which allowed us to determine which side or sides were empty.

There are other creative uses of emptiness in landmarks. For example, to know whether the robot successfully grasped a thin, lightweight object, the robot may need to visually inspect its gripper. We were able to program this by creating a landmark of the gripper with empty space underneath (Fig. 2j).

We programmed the robot to determine whether it should use its right gripper or its left gripper to grasp an object from a shelf. In the confined shelf space, there was only room to grasp the object from one side. To determine which side to use, we created an object landmark with space to its right and another version with space to the left (Fig. 2k).

Landmarks can also be used to find empty regions of the workspace. For example, we were able to sample an empty spot on the cluttered tabletop by creating a landmark of a tabletop patch with empty space above it (Fig. 2l). In our final example of using empty space, we envisioned the robot loading clothes into a basket. To check if clothes were hanging from the edge of the basket, we created a landmark of the basket edge with some empty space around it. Fig. 2(m) shows that this landmark can distinguish between the back edge of the basket, which has nothing on it, and the front edge, which has a shirt hanging over it.

4) Hole shapes: The use of empty space in our landmark representation means that users can create landmarks of workspace areas that are predominantly hole-shaped. For example, some trash bins and recyling bins have lids with differently-shaped holes. We were able to differentiate them using landmarks of the lids (Fig. 2n). Die-cut foam packaging, which has shapes cut out for holding items during shipping (Fig. 2o), is another example. Using hole-shaped landmarks, we were able to locate the spaces for two different objects in the foam packaging.

5) Extra specificity: Fig. 2(p) shows how landmarks can be designed slightly differently to represent different task

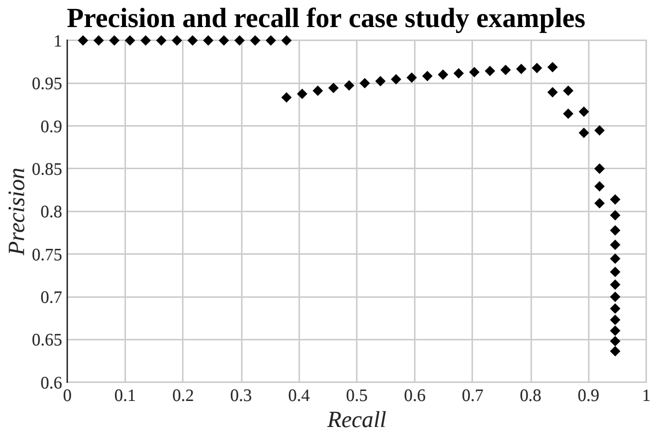


Fig. 3: The precision and recall of the search algorithm for the case study examples described in Section IV-A. The vertical axis is shown with a minimum value of 0.6. The large drop in the top left is due to a false positive match that occurs at a relatively low error threshold.

TABLE I: Numeric values of precision and recall as shown in Fig. 3 for several values of the error threshold.

Error threshold	Precision	Recall
0.0049	1.00	0.38
0.0062	0.97	0.84
0.0064	0.94	0.87
0.0068	0.92	0.90
0.0070	0.90	0.92
0.0075	0.81	0.95

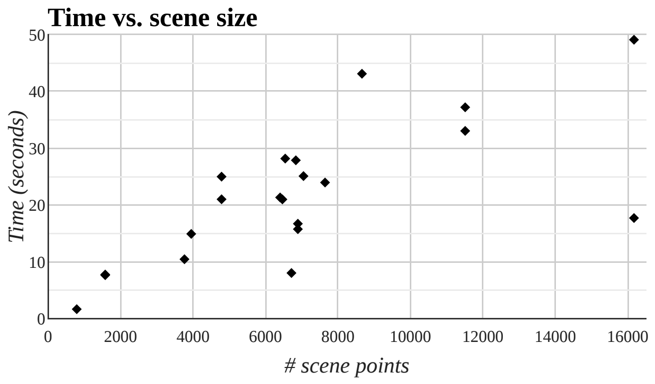


Fig. 4: Time taken to search for landmarks over different scene sizes.

requirements in the same scene. By adding empty space to the top of a landmark of a can, the system can locate the can on the top of a stack. In contrast, a landmark of a can that includes part of the tabletop represents the can on the bottom. Cans in the middle of a stack are harder to detect because a generic can landmark, without the extra features of the top or bottom cans, will have many matches along the stack as the boundaries between cans is invisible. When working with stacks of objects, we expect that most users will be interested in locating only the object on the top.

B. Case study performance

In this section, we quantify the performance of the search algorithm on the case studies described in Section IV-A.

1) Precision and recall: To measure the accuracy of the system, we recorded the precision and recall of the system using different error thresholds (Algorithm 1, line 13). For each case study, we created a landmark in one scene and searched for the landmark in a different scene, in which objects and/or the workspace changed positions. We labeled each scene with the correct locations of landmarks. For some scenes, we also searched for landmarks that were absent from the scene. For examples like finding an empty patch of tabletop, which had many possible correct answers, we manually graded the output of the algorithm instead of labeling the scene. Fig. 3 shows the precision and recall curve for this experiment. In this experiment, we were not able to obtain 100% recall with any error threshold, because the non-max suppression step of Algorithm 1 prevents some results from being output. Table I shows the precision and recall numbers for several values of the error threshold.

2) Speed: Most of the time spent in *CustomLandmarks* is in the loop between lines 5 and 11 of Algorithm 1, which evaluates a sample position in the scene. The number of samples, in turn, is determined by the size of the scene. Fig. 4 shows a plot of the speed of the algorithm against the number of points in the scene. This experiment was conducted on a computer with an Intel® Core™ i7-4770 CPU and 16 gigabytes of memory.

These results show that the system is mainly useful for functional tasks in which speed is not critical. Implementing improvements, such as using GPU computing or algorithmically eliminating excess work, could make the system faster in the future. However, experienced robot programmers can mitigate this issue by preprocessing the scene in ways that are appropriate for the task. For example, if the landmark is in a tabletop scene and the tabletop is not a relevant landmark, the scene could be preprocessed to filter it out.

C. Limitations

Other than speed, *CustomLandmarks* has two main limitations, illustrated in Figure 5. The first is that the system relies on landmarks having a unique shape in the scene. The system works best in semi-structured environments where there are few distractions in the scene. To avoid this limitation, users can redesign the landmark so that it is unique, although it may not always be possible to do so.

The other main limitation is that the system can be brittle to viewpoint changes, such as when an asymmetric object is rotated. This is because we only capture landmarks from a single view of a point cloud. One way to mitigate this limitation is for users to create multiple landmarks representing rotated versions of the same object.

V. Robot experiments

We ran a set of experiments to evaluate how well the results from our offline experiments transfer to real manip-

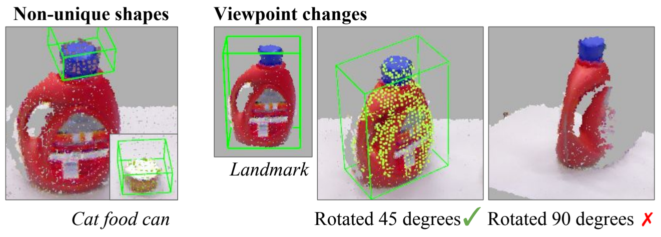


Fig. 5: Examples of limitations to the system. The left example shows how a landmark of a can food can could be confused for the cap of a laundry detergent bottle. The example on the right shows how a landmark of the bottle can be detected after a small viewpoint change (a 45 degree rotation), but not after a larger change.

ulation tasks. To this end, we used *CustomLandmarks* in conjunction with programming by demonstration (PbD) to program six manipulation actions on a PR2 robot.

A. Programming by demonstration with CustomLandmarks

To program manipulation actions on the PR2, we modified the PbD system in [23]. In this system, users program manipulation actions by guiding the robot's arms through a sequence of end-effector poses. Poses are defined either relative to the robot's base or relative to a landmark. Once the poses have been saved, the robot can repeat the action later. First, it searches for the necessary landmarks, then it adjusts the poses defined relative to those landmarks. Finally, the robot's arms move through the sequence of poses, opening or closing the grippers as programmed. Using *CustomLandmarks* allows users to define perceptual landmarks for their task, and PbD lets users create manipulation actions based on those landmarks, all without writing any code.

B. Task descriptions

The tasks were based on the case studies described in Section IV-A. Each action we programmed was tested in 5 variants of the task, in which we moved objects and the workspace around, or we added or removed objects.

1) Tool rack: We programmed the robot to either remove a tool from a tool rack or place a tool in an empty spot on the rack. The rack consisted of three 3D-printed wall mounts, and all of the tools had the tool adaptor. Both are shown in Fig. 2(b, d). The tasks were varied by changing where the tools were placed. We also included scenarios where the tool rack was full or empty and the correct action was to not act. All but one of the tests succeeded. In the failed test, the robot located an empty wall mount but used too much force when placing the tool on it.

2) Grasping a chair: Previously, the PbD system we used only worked in tabletop scenes. PbD could work in non-tabletop scenes using our system, we programmed the robot to detect and grasp a chair (Fig. 2g). We varied the scenes by setting the chair to different heights and moving it around. One of the five tests failed because our system could not locate the chair, showing how our system could fail if the viewpoint changed too much.

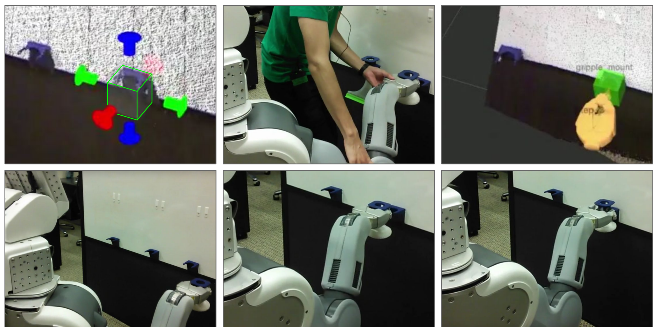


Fig. 6: The PR2 robot using *CustomLandmarks* to place a tool on a tool rack. The top row shows the user defining the landmark and demonstrating the action and the robot localizing the landmark in a new scene. The bottom row shows the robot executing the task.

3) Picking bowls from a shelf: In this task, the robot picked bowls from a shelf. The tasks were varied by placing bowls on different levels of the shelf, placing them in stacks, or removing all the bowls entirely. Because the shelf was at eye level and some bowls were stacked, it was necessary to create a bowl rim landmark, as illustrated in Fig. 2(a). The robot successfully grasped a bowl in all five scenarios.

4) Waste bins: For this task, we wanted the robot to either deposit an item into the trash bin or the recycling bin shown in Fig. 2(n). In the different scenarios, we varied where the bins were placed and which bin the robot needed to drop the item into. In one scenario, we removed the bin the robot needed to drop the item to see if it would mistakenly do so. All five scenarios we tested worked as expected.

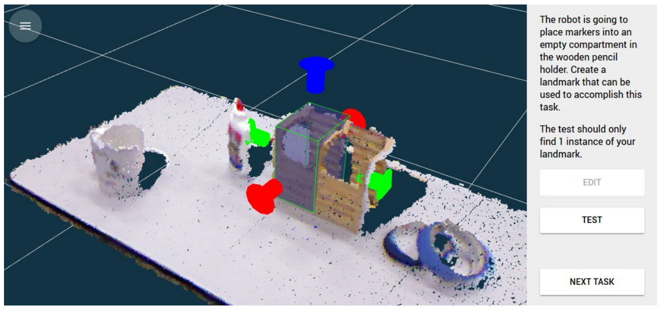
5) Cluttered tabletop: In this task, the robot needed to drop an item onto an empty spot on a table (Fig. 2l). We varied the items and their positions on the table. One of these scenarios failed because we placed a large, flat box on the table, which our system thought was an empty patch on the table. This failure shows how our system needs landmarks to have a unique shape in the scene.

6) Control panel: In this task, the robot pushed buttons on a mockup of a control panel (Fig. 2f). Instead of locating the buttons directly, the robot instead located a dial, and pushed the buttons based on the location of the dial. We varied the scenarios by moving the control panel around and increasing its incline angle. The system failed to locate the dial in one scenario where the control panel was inclined.

C. Discussion

Overall, the robot succeeded in 26 out of the 30 total scenes it was evaluated in. Our search algorithm detected all of the different kinds of landmarks described in Section IV-A. It also correctly recognized all of the cases where needed landmarks were missing, which avoided having the robot execute an action it was not supposed to.

The actions we programmed using PbD did not require writing any code. However, adding programming logic to the system could have helped us avoid some of the failure cases we encountered. For example, when searching for an empty


The robot is going to place markers into an empty compartment in the wooden pencil holder. Create a landmark that can be used to accomplish this task.

The test should only find 1 instance of your landmark.

Fig. 7: The experiment interface in editing mode. The interface to create landmarks is shown on the left and a user study-specific display is shown on the right.

patch on a table, we could have filtered out all detections that were not located at the table's height.

VI. User evaluation

Although *CustomLandmarks* can be used to design non-standard landmarks, it is important to know if users can actually understand and utilize the system in such a way. To address this question, we conducted a user study in which participants created landmarks for different robot tasks.

A. Study design

We asked users to create landmarks that would enable a robot to execute a set of three tasks. Tasks were assigned in random order. For each task, users had to create their landmark in one scene but test it in a different scene. This was to emulate real-world usage of the system. In the experiment interface (Fig. 7) users could switch between editing and testing modes as many times as needed. An experimenter observed their actions and informed users when they had solved a task correctly.

1) Tasks: The robot tasks were:

a. Placing an object on an empty spot on a cluttered table.
b. Picking bowls out of an eye-level shelf (including from the top of a stack of bowls).
c. Placing an object in an empty compartment of a two-compartment pencil holder.

Our main research question was whether users could design unconventional landmarks using *CustomLandmarks*. The correct landmarks were either object parts or scene parts. The first task required users to create a landmark of an empty patch of the table surface. To prevent trivial solutions with patches that were too small or too large, we required that the landmark be found in the test scene at least six times with no false positives. In the second task, users initially saw unstacked bowls, so the obvious solution was to make a landmark of a bowl. However, in the test scene, we turned one of the bowls into a stack, which required users to modify the landmark to represent a bowl rim. In the third task, we showed users an empty pencil holder, but in the test scene, one of the sides was filled with objects. As a result, users had to figure out that the landmark should be a single side of the container, as shown in Fig. 7.

TABLE II: Mean task performance measures from the user evaluation. Task numbers are as given in Section VI-A.1.

Task	Total time	Edit mode time	# of edits
Task A	5:55	3:16	5.64
Task B	4:15	2:23	4.5
Task C	3:38	2:33	3
All tasks	4:34	2:44	4.35

TABLE III: NASA-TLX survey results.

NASA-TLX subscale	Mean workload rating	Stdev
Mental demand	54.67	4.13
Physical demand	25.00	4.28
Temporal demand	29.00	3.99
Performance	29.00	4.49
Effort	54.33	3.48
Frustration	32.67	5.38
Raw TLX score	44.93	15.75

2) Training: To train users, an experimenter read from a script to explain the concept of landmarks. Users performed one training task where there was a line of cans on a table. Users practiced creating a landmark of a can. They were then introduced to the concept of a landmark representing empty space by modifying their landmark to only match with the rightmost can. The training session took approximately five minutes to complete.

3) Participants: We recruited participants through email and social media posts targeting undergraduate and graduate engineering students at the University of Washington. All participants were required to be novices to *CustomLandmarks*. This represented the population that would benefit from the system the most: technical, college-educated people who are not necessarily experts in computer vision or robotics. We recruited 15 participants, 9 male and 6 female. Their ages ranged from 18 to 32 and averaged 23.3. All participants received a $10 gift card for completing the study.

4) Measures: Because an experimenter told users whether each task had been completed correctly, we measured task performance using time spent per task and number of tasks completed within the 30 minutes allotted for the study. The experiment interface also measured the time spent editing and testing landmarks. Finally, we administered the widely-used NASA Task Load Index (TLX) survey [27] to measure the subjective workload of the tasks.

B. Results

14 out of the 15 participants correctly completed all of the tasks within the time limit of the study. The average time to complete a task correctly was 4:34 minutes, with 2:44 minutes of that time spent in editing mode. Performance measures for each task are summarized in Table II.

There are many ways to interpret TLX survey ratings [27], so Table III reports the average ratings for each workload category. The Raw TLX score, an average of all the categories, was 44.9, which is in the 40th percentile of TLX scores according to one meta-analysis [28].

C. Discussion

Our results show that users can learn to use *CustomLandmarks* and design unconventional landmarks in a short amount of time. Additionally, users were able to generalize from the training task, in which they created a landmark of a whole object, to tasks in which the landmarks represented parts of objects or parts of the scene.

In the study, users could rapidly switch between editing and testing modes to refine their landmark. However, when actually using *CustomLandmarks*, there may be a larger time gap between when a landmark is created and when it is used. As a result, we suggest that any tool for creating custom landmarks should incorporate a testing function that lets users try out the landmark on pre-recorded scenes.

VII. Conclusion

We present a system that enables roboticists and non-roboticists alike to rapidly create perceptual detectors for robot manipulation tasks. Our novel representation of *landmarks* allows users to complete perceptual tasks involving object parts or parts of a robot's workspace—tasks that traditional detectors are not designed to solve. Through experiments on over a dozen radically different tasks and 30 task executions, we showed that the system is effective at locating landmarks and can be deployed on robots in the real world. Finally, through a user study with 15 novices to the system, we showed that our user interface, landmark representation, and search algorithm are all intuitive to learn and use. In future work, we will improve on the system's accuracy and speed and apply the concept of user-defined landmarks to domains where color information is important.

Acknowledgements

This work was supported by the National Science Foundation, Awards IIS-1552427 "CAREER: End-User Programming of General-Purpose Robots" and IIS-1525251 "NRI: Rich Task Perception for Programming by Demonstration."

References

[1] A. G. Buch, D. Kraft, J.-K. Kamarainen, H. G. Petersen, and N. Krüger, "Pose estimation using local structure-specific shape and appearance context," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 2013, pp. 2080–2087.

[2] B. Drost, M. Ulrich, N. Navab, and S. Ilic, "Model globally, match locally: Efficient and robust 3D object recognition," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 998–1005.

[3] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (FPFH) for 3D registration," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, 2009, pp. 3212–3217.

[4] A. Collet, M. Martinez, and S. S. Srinivasa, "The MOPED framework: Object recognition and pose estimation for manipulation," *The International Journal of Robotics Research*, vol. 30, no. 10, pp. 1284–1306, 2011.

[5] C. Papazov, S. Haddadin, S. Parusel, K. Krieger, and D. Burschka, "Rigid 3D geometry matching for grasping of known objects in cluttered scenes," *The International Journal of Robotics Research*, vol. 31, no. 4, pp. 538–553, 2012.

[6] G. Anthes, "Deep learning comes of age," *Communications of the ACM*, vol. 56, no. 6, pp. 13–15, 2013.

[7] D. Maturana and S. Scherer, "VoxNet: A 3D convolutional neural network for real-time object recognition," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, 2015, pp. 922–928.

[8] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D ShapeNets: A deep representation for volumetric shapes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1912–1920.

[9] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser, "3DMatch: Learning local geometric descriptors from RGB-D reconstructions," *arXiv preprint arXiv:1603.08182*, 2016.

[10] K. Hsiao, L. P. Kaelbling, and T. Lozano-Perez, "Grasping POMDPs," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 4685–4692.

[11] A. Saxena, J. Driemeyer, and A. Y. Ng, "Robotic grasping of novel objects using vision," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 157–173, 2008.

[12] M. Gualtieri, A. ten Pas, K. Saenko, and R. Platt, "High precision grasp pose detection in dense clutter," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 598–605.

[13] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 705–724, 2015.

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

[15] K. Hsiao and T. Lozano-Perez, "Imitation learning of whole-body grasps," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, 2006, pp. 5657–5662.

[16] A. Herzog, P. Pastor, M. Kalakrishnan, L. Righetti, T. Asfour, and S. Schaal, "Template-based learning of grasp selection," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 2379–2384.

[17] A. Blake, C. Rother, M. Brown, P. Perez, and P. Torr, "Interactive image segmentation using an adaptive GMMRF model," in *European Conference on Computer Vision*. Springer, 2004, pp. 428–441.

[18] W. Yang, J. Cai, J. Zheng, and J. Luo, "User-friendly interactive image segmentation through unified combinatorial user inputs," *IEEE Transactions on Image Processing*, vol. 19, no. 9, pp. 2470–2479, 2010.

[19] R. Robotics, "Baxter user guide," http://mfg.rethinkrobotics.com/wiki/Support_Resources.

[20] S. Niekum, S. Chitta, A. G. Barto, B. Marthi, and S. Osentoski, "Incremental semantically grounded learning from demonstration." in *Robotics: Science and Systems*, 2013.

[21] M. Wächter, S. Schulz, T. Asfour, E. Aksoy, F. Wörgötter, and R. Dillmann, "Action sequence reproduction based on automatic segmentation and object-action complexes," in *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2013, pp. 189–195.

[22] S. Niekum, S. Osentoski, G. Konidaris, and A. G. Barto, "Learning and generalization of complex tasks from unstructured demonstrations," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012, pp. 5239–5246.

[23] S. Alexandrova, M. Cakmak, K. Hsiao, and L. Takayama, "Robot programming by demonstration with interactive action visualizations," in *Robotics: Science and Systems (RSS)*, 2014, pp. 48–56.

[24] B. Akgun and A. Thomaz, "Simultaneously learning actions and goals from demonstration," *Autonomous Robots*, vol. 40, no. 2, pp. 211–227, 2016.

[25] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," in *Robotics-DL tentative*, 1992, pp. 586–606.

[26] Z. Xu and M. Cakmak, "Enhanced robotic cleaning with a low-cost tool attachment," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, 2014, pp. 2595–2601.

[27] S. G. Hart, "NASA-task load index (NASA-TLX): 20 years later," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 50, no. 9. Sage Publications Sage CA: Los Angeles, CA, 2006, pp. 904–908.

[28] R. A. Grier, "How high is high? A meta-analysis of NASA-TLX global workload scores," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 59, no. 1. SAGE Publications Sage CA: Los Angeles, CA, 2015, pp. 1727–1731.