Multi-view Self-supervised Deep Learning for 6D Pose Estimation in the Amazon Picking Challenge

Andy Zeng\(^1\) Kuan-Ting Yu\(^2\) Shuran Song\(^1\) Daniel Suo\(^1\) Ed Walker Jr.\(^3\) Alberto Rodriguez\(^2\) Jianxiong Xiao\(^4\)
\(^1\)Princeton University \(^2\)Massachusetts Institute of Technology \(^3\)Google \(^4\)AutoX

Abstract— Warehouse automation has attracted significant interest in recent years, perhaps most visibly by the Amazon Picking Challenge (APC)\(^1\). Achieving a fully autonomous pick-and-place system requires a robust vision system that reliably recognizes objects and their 6D poses. However, a solution eludes the warehouse setting due to cluttered environments, self-occlusion, sensor noise, and a large variety of objects. In this paper, we present a vision system that took 3rd- and 4th- place in the stowing and picking tasks, respectively at APC 2016. Our approach leverages multi-view RGB-D data and data-driven, self-supervised learning to overcome the aforementioned difficulties. More specifically, we first segment and label multiple views of a scene with a fully convolutional neural network, and then fit pre-scanned 3D object models to the resulting segmentation to get the 6D object pose. Training a deep neural network for segmentation typically requires a large amount of training data with manual labels. We propose a self-supervised method to generate a large labeled dataset without tedious manual segmentation that could be scaled up to more object categories easily. We demonstrate that our system can reliably estimate the 6D pose of objects under a variety of scenarios. All code, data, and benchmarks are available [2].

I. INTRODUCTION

Over the last two decades, retail firms have increasingly automated their warehouse operations to fulfill growing demand from e-commerce and to provide faster, cheaper delivery. However, many tasks still hard to automate. Amazon has led a recent effort to define two such tasks: (1) picking: given an product ID, pick one instance out of a populated shelf and place it into a tote; (2) stowing: given a tote of products, pick and stow all of them into a populated shelf.

In this work, we describe our computer vision system that took 3rd place in the stowing and 4th in picking at the 2016 Amazon Picking Challenge (APC), and provide experiments to validate our design decisions. To successfully compete, our vision algorithms estimates the 6D poses of objects robustly under challenging scenarios:

1. Cluttered environments: both shelf bins and totes may have multiple objects and could be arranged to deceive vision algorithms (e.g., on top of one another).
2. Self-occlusion: due to limited camera positions, the system may only see a partial view of an object.
3. Missing data: commercial depth sensors are often unable to capture reflective, transparent, or mesh-like materials, which are all common in product packaging.
4. Small or deformable objects: small objects provide fewer data points while deformable prevent alignment to a rigid model.

II. RELATED WORK

Robots designed to manipulate objects typically require their vision algorithms to provide 2D bounding boxes, pixel-level segmentation\([4, 5]\), or 6D poses\([6, 7]\). The choice depends primarily on the manipulation task. For example, a 2D bounding box or pixel-level segmentation may be sufficient for a suction strategy, but a grasping strategy requires the 6D pose of an object.
Object segmentation. While last year’s winning team used a histogram backprojection method [8] with manually defined features [5, 4], recent work in computer vision has shown that deep learning considerably improves object segmentation [3]. In this work, we extend the state-of-the-art deep learning architecture used for image segmentation to incorporate depth and multi-view information.

6D pose estimation. Two primary approaches exist for estimating an object’s 6D pose. The first aligns 3D CAD models to 3D point clouds obtained from depth sensors using algorithms such as iterative closest point [9]. The second uses local descriptors such as SIFT keypoints [10] for color data or 3DMatch [11] for 3D data. The former approach is mainly used with depth-only sensors, in scenarios where lighting changes significantly, or on textureless objects. Highly textured and rigid objects, on the other hand, benefit from color descriptors. Meanwhile, many existing approaches such as LINEMOD [12] are based on the assumption that a single object sits on a table top and do not perform well when confronted with the limited visibility and clutter imposed by the APC scenario [13].

Benchmark for 6D pose estimation. In evaluating our algorithms as a component independent from the larger robotic system, we have produced a large, manually-labeled benchmark dataset for APC 2016 with objects’ 6D poses and segmentations. Previous efforts to construct benchmark datasets include Berkeley’s dataset for a number of objects both from and beyond APC 2015 [14] and Rutgers’s semi-automatically labeled data [15].

III. APC 2016

In a typical Amazon warehouse, millions of products are spread across nearly 1 million square feet and must be picked out of bins before shipment. The APC 2016 posed a simplified version of the general picking and stowing tasks. In the picking task, robots sit within a 2x2 meter area in front of a shelf populated with objects, and autonomously picks items from a list of 12 desired items and places them into a tote. In the stowing task, robots pick all the 12 items inside a tote and place them inside the shelf.

Before the competition, teams were provided with a list of 39 possible objects along with 3D CAD models of the shelf and tote. At run-time, the robots were provided with the initial contents of each bin on the shelf and a work order containing which items to pick. After picking and stowing the appropriate objects, the system had to report the final contents of both shelf and tote. Details of the competition can be found in [1].

IV. SYSTEM DESCRIPTION

Our vision system takes in RGB-D images from multiple views, and outputs 6D poses and a segmented point cloud for the robot to complete its picking or stowing task.

The camera is located on the end-effector, right below the two-finger gripper (Figure 1). This location allows the robot to place the camera in almost any pose inside the workspace. We can also use the camera in this location to confirm if the object has been successfully grabbed or sucked. A minor drawback of the camera location is that the gripper fingers partially blocks the camera view, occupying about 10% of the view frustum. We adjust the camera viewpoints used to capture data such that the non-blocked portion covers the entire scene. We chose the RealSense F200 RGB-D camera for two reasons. First, it has the close depth range that we need for warehouse object manipulation (0.2m–1.5m). Second, it is a consumer-level range sensor with a decent amount of flexibility from the data capture. We use kinematic data from the robot arm to fuse the point cloud from multiple views.

V. 6D OBJECT POSE ESTIMATION

Our algorithm estimates the 6D pose of all objects in a scene in two phases (Figure 2). First, we segment RGB-D point clouds captured from multiple views into different objects using a deep convolutional neural network. Second, we align pre-scanned 3D models of the identified objects to the segmented point clouds to estimate the 6D pose of each object. Our approach is based on several well-known methods. However, our evaluations show that when used alone, they are far from sufficient. As such, we present brief descriptions of these methods followed by in-depth discussions of how we combine and enhance these methods to formulate a robust vision system.

A. Object Segmentation with Fully Convolutional Networks

In recent years, ConvNets have made tremendous progress for computer vision tasks [16, 3]. We leverage these advancements to segment camera data into the different objects in the scene. Explicitly, we train a VGG architecture [17] Fully Convolutional Network (FCN) [3] to perform 2D object
segmentation. The FCN takes an RGB image as input and returns a set of 40 densely labeled pixel probability maps—one for each of the 39 objects and one for the background—of the same dimensions as the input image.

**Object segmentation using multiple views.** While information from only one camera view of an object may be limited, we can combine information from multiple views to more confidently distinguish and segment object surfaces for the model-fitting phase. We feed each of the RGB-D images taken from multiple views (18 of the tote in the stowing challenge; 15 of the shelf bin in the picking challenge) forward through our trained FCN, returning 40 class probabilities per pixel in each RGB-D image. First, we ignore all probability maps associated with objects that are not in the scene. From among the remaining probabilities maps, we compute a threshold per remaining object class (three standard deviations above the mean probability across all views) and ignore any pixels whose probabilities are all below these thresholds. Using the depth component of the RGB-D images, we project the remaining pixels into 3D space, and combine them with the kinematic information from the robot arm to give us a single segmented point cloud (segmentations can overlap each other).

Reducing noise. Fitting pre-scanned models to the segmented point cloud directly often gives poor results because of noise from our sensor and noise from the segmentation. We address this issue in three steps: First, we eliminate spatial outliers from the labeled point cloud caused by noisy depth data from the RGB-D sensor. We remove a point if the average distance to its k-nearest neighbors is above a threshold. Second, to combat noise from the segmentation algorithm, we remove background points. By aligning the 3D point cloud of the scene to the point cloud of the empty tote or shelf bin We remove both points too close to the background point cloud and those positioned outside the shelf bin or tote. Finally, we remove outlier points from each segmented group of points by finding the largest contiguous set of points along each principal axis (computed via PCA) and remove any points that are disjoint from this set.

Handling duplicate objects. Warehouse shelves commonly contain multiple instances of the same object. Naively segmenting RGB-D data will treat two distinct object with the same label as the same object. Because we know the inventory list in the warehouse setting, we know the number of identical objects we expect in the scene and use k-means clustering to separate the mass of pixels into the appropriate number of objects. Each cluster is then treated independently during the model-fitting phase of the algorithm.

B. 3D Model-Fitting

Using the fused point cloud generated from the first phase, we use the iterative closest point (ICP) algorithm [18] to fit pre-scanned 3D models of objects to their corresponding segmentations to estimate the objects’ poses. However, applying the vanilla ICP gives nonsensical results in many scenarios. We describe several such pitfalls along with our solutions.

Uniforming density for point clouds. In a typical point cloud obtained from a RGB-D sensor, surfaces perpendicular to the sensor’s optical axis are often has denser point cloud distribution. There are other examples where the density of points does not accurately reflect the surfaces they attempt to capture. In these cases, the ICP algorithm often optimizes for denser areas of the two point clouds. By applying a 3D uniform average grid filter to the point clouds, we are able to give them consistent distributions in 3D space.

Pose initialization. Because ICP takes an iterative approach to estimating the final transforms, it is very sensitive to initial pose. To initialize the rotation of the object, we observe that the principal directions of the segmented point cloud...
To automatically obtain pixel-wise object labels, we separate the target objects from the background to create an object mask. There are a 2D and a 3D component in this data process. Both use color and depth information. The 2D pipeline is robust to thin objects and objects with no depth, while the 3D pipeline is robust to an unstable background.

Computed from PCA give the approximate orientation of the object with an uneven aspect ratio. We also observe that the choice of orientation of objects with relatively even aspect ratios have little impact on the final result. To initialize the position of the object, we observe that a scanner often capturing a partial view of the object, and running ICP on this point cloud with a point cloud of the entire object directly could bring closer the centers of mass of the two point clouds instead of the partial surface from the scan with the appropriate surface on the complete object. To address this, we push back the initial pose of the pre-scanned object back along the optical axis of the RGB-D camera by half the size of the object’s bounding box. This initialization allows ICP to bring the correct surface of the complete model to the partial view, rather than iterate to a local optimum.

**Coarse to fine ICP.** Even after reducing noise in the segmentation step, a segmented object’s point cloud may still have noise (e.g., mislabeled points from adjacent objects). We address this with two passes of ICP. We differentiate the passes by the points we include in the algorithm: we define the inlier threshold of an ICP iteration as the percentile L2 distance above which we ignore. ICP with a 90% inlier ratio keeps the closest pairs of points between the two point clouds up to the 90th percentile.

In this approach, we assume that points with the correct label are denser than points with an incorrect label. After the first pass with a high inlier threshold (90%), we move the pre-scanned complete model closer to the correct portion of the partial view than the noisy portion. Given that the pre-scanned model should be closer to the correct portion, the second pass uses a lower inlier threshold (45%) to ignore the noisy portion of the point cloud.

**C. Handling Objects with Missing Depth.**

Many objects in both the APC and typical retailing warehouses have surfaces that challenge infrared-based depth sensors. Reflective material returns depth from reflected objects, whereas transparent or mesh materials may not register at all. Our pose estimation algorithm performs poorly for these objects; the resulting point cloud is noisy and sparse.

Our solution leverages our results from the multi-view segmentation to estimate a convex hull of the object. We first discretize the relevant space (e.g., shelf bin or tote) into a large 3D grid and project the fused segmentation results into voxels. Voxels that are inside of the segmentation and have been seen from multiple views are considered part of the object. We then compute a convex hull from the 3D occupied voxels, and the centroid of the hull is the object’s center of mass, while the shape of the hull determines the orientation of the object (assuming that the object is axis-aligned).

**VI. SELF-SUPERVISED TRAINING**

By bringing deep learning into our approach, we attempt to gain robustness at the expense of amassing quality training data. This is necessary to learn high-capacity models with many parameters. However, gathering and labeling such large amounts of training data manually can be expensive. The existing large-scale dataset used by deep learning (e.g., ImageNet [19]) are mostly Internet photos, which has very different object and image statistics from our warehouse setting.

To automatically label large amounts of training images with pixel-wise labels, we propose a self-supervised method. Our method is based on three observations: 1) batch-training on scenes with single object can train a deep model to perform well on scenes with multiple objects [16] (i.e., simultaneous training on cat-only or dog-only images enables successful testing on cat-with-dog images); 2) we can move the camera mounted on a robot arm precisely; 3) in single object scenes, with known background and known camera motions, we can automatically obtain precise foreground masks as segmentation training labels. In total, the resulting training dataset contains 136,575 training images of 39 objects, all of which are automatically labeled by our framework to train the network.

**Semi-automatic data gathering.** To semi-autonomously gather immense quantities of training data, we place objects
inside the shelf bins and tote in arbitrary poses, and configure
the robot to move the camera and capture RGB-D images
of the objects from a variety of different viewpoints. Each
individual shelf bin and tote contains only one object, which
is known and manually labeled. The position of the shelf
bin and tote is known to the robot, and so these object
labels are automatically propagated to the collected RGB-
D images based on camera view direction. After capturing
several hundred RGB-D images, the objects are re-arranged
to different poses in the bins and tote, and the process
is repeated again several times. In this process, human
involvement sums up to re-arranging the objects and labeling
which objects correspond to which bin/tote. On the other
hand, moving around different viewpoints, capturing sensor
data, and labeling each image by object is done automatically
by the robot. We also collect RGB-D images of empty shelf
bins and tote from the same exact camera viewpoints, in
preparation for the automatic data labeling.

Automatic data labeling. To obtain pixel-wise object labels,
we separate the foreground objects from the background to
create an object mask. There are two components in this data
labeling process: one operates in 2D and another operates in
3D. The 2D pipeline is robust to thin objects (objects that
cannot be reliably detached in 3D when placed too close to
a side or lying flat) and objects with no depth (i.e. no point
cloud), while the 3D pipeline is robust to large misalignments
between the pre-scanned shelf bin and tote. Results from both
pipelines are combined to automatically label an object mask
for each training RGB-D image.

In the 2D pipeline, we do pixel-wise comparisons between
RGB-D images with the object and background images
(empty shelf and tote) in order to identify the foreground
mask. We first fix minor image misalignments by using
multimodal 2D intensity-based registration to align the two
RGB-D images [20]. We then convert the aligned color image
from RGB to HSV, and then do pixel-wise comparisons of
the HSV values and depth values. If the difference in HSV
values or depth values is small, then the pixel is labeled as
background, otherwise labeled as foreground.

In the 3D pipeline, we align a pre-scanned 3D model of the
empty shelf bin or tote to the point cloud of the training data
and do point-to-point comparisons to identify the foreground
mask. We first capture multi-view RGB-D images of an
empty shelf bin and tote to create their pre-scanned 3D
models. We then use ICP to align these background models
to the projected point clouds of the RGB-D training images,
and remove points too close to the background. We project
the foreground points back to 2D to retrieve our object mask.

Training neural network. To leverage features trained
from a larger image domain, we use the sizable FCN-VGG
network architecture from [17] and initialize the network
weights using a model pre-trained on ImageNet for 1000-way
object classification. We fine-tune the network over the 40-
class output classifier (39 classes for each APC object and 1
class for background) using stochastic gradient descent with
momentum. Due to illumination and object viewpoint biases,
we maximize performance by training two such segmentation
networks: one for shelf bins and one for the tote.

VII. IMPLEMENTATION

All components of the vision system are modularized into
reusable ROS packages, with CUDA GPU acceleration. Deep
models are trained and tested with Marvin [21], a ROS-
compatible and lightweight deep learning framework. Training
our models takes up to 16 hours prior to convergence.

Our robot is controlled by a computer with an Intel E3-
1241 CPU 3.5 GHz and an NVIDIA GTX 1080. The run-
time speeds per component are as follows: 10ms for ROS
communication overhead, 400ms per forward pass of VGG-
FCN, 1200ms for denoising per scene, and 800ms on model-
fitting per object. On average, pose estimation time is 3-
5 seconds per shelf bin and 8-15 seconds for the tote.
Combined with multi-view robot motions, total vision time
with multi-view takes 10-15 seconds per shelf bin and 15-20
seconds for the tote.

VIII. EVALUATION

We evaluate variants of our method in different scenarios
on our benchmark dataset to understand (1) how segmentation
performs under different input modalities and training
dataset sizes and (2) how the full vision system performs.
A. Benchmark Dataset

Our benchmark dataset, ‘Shelf & Tote’, contains over 7,000 RGB-D images spanning 477 (Figure 7) scenes at 640 × 480 resolution. We collected the data during practice runs and competition finals for the APC and manually labeled 6D object poses and segmentations using our online annotator (Figure 8). The data reflect various challenges found in the warehouse setting: reflective materials, lighting conditions, partial views, and sensor limitations (noisy and missing depth) over cluttered environments.

Our experimental results in Tables I, II highlight the differences in performance over subsets of the dataset with scenes captured:

cptn: during competition at the APC finals
environment: in an office (off); in the APC competition warehouse (whs)
task: picking from a shelf bin or stowing from a red tote
clutter: with multiple objects
occlusion: with % of object occluded by another object, computed from ground truth
object properties: with objects that are deformable, thin, or have no depth from the RealSense F200 camera

B. Evaluating object segmentation

We test several variants of our FCN on object segmentation to answer two questions: (1) can we leverage both color and depth segmentation? (2) is more training data useful?

Metrics. We compare the predicted object segmentations from our trained FCNs against the ground truth segmentation labels of the benchmark dataset using pixel-wise precision and recall. Table I displays the mean average F-scores ($f = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$).

Depth for segmentation. We use HHA features [22] to encode depth information into three channels: horizontal disparity, height above ground, and angle of local surface normal with the inferred direction of gravity. We compare AlexNet trained on this encoding, VGG on RGB data, and both networks concatenated in Table I.

We find that the FCN performs significantly better when trained on color data than on depth data, with the largest disparity for deformable objects and thin objects, whose textures provide more discriminative power than geometric.

More training data. Deep learning models have seen significant success, especially if given large amounts of training data. However in our scenario—instance-level object segmentation on few object categories—it is not clear whether such a large dataset is necessary. We create two new datasets by randomly sampling 1% and 10% of the original and use them to train two VGG FCNs (Table I). We confirm marked improvements in F-score across all benchmark categories going from 1% to 10% of training data.

C. Evaluating pose estimation

We evaluate several key components of our vision system to determine whether they increase performance in isolation.

Metrics. We report the percentage of object pose predictions with rotations within 15°, and the percentage with translations within 5cm. The metric recognizes the structural invariance of several objects, some of which are axially-symmetric (cuboids), radially-symmetric (bottles, cylinders), or deformable (see web page [2] for details). Under this metric, correctly localized object poses will certainly be sufficient for picking with sensor-guarded motions.

Multi-view information. By capturing images from multiple views, the system can observe more object surface information that would otherwise be missing from self-occlusion or inter-object occlusion or clutter. Multi-view information also help to reduce the chance for problems caused by illumination on reflective surfaces.

We test the full vision system over the benchmark using three different subsets of camera views: (1) 15 views for shelf bins and 18 views for the tote (10cm apart between camera poses), (2) 5 views for shelf bins and 10 views for the tote (sparse arrangement, wide-baseline view angles), and (3) 1 view for shelf bins and 2 views for the tote (minimal number of views required to capture the scene). Assuming camera viewpoints are zero-indexed in row-major order ($5 \times 3$ for shelf bins and $3 \times 6$ for the tote, see Figure 3), the camera arrangements are as follows: $a_{shelf} = \{0,1,\ldots,14\}$ and $a_{tote} = \{0,1,\ldots,17\}$, $a_{shelf} = \{0,4,7,10,14\}$ and $a_{tote} = \{2,4,6,8,9,11,13,15,17\}$, $a_{shelf} = \{7\}$ and $a_{tote} = \{7,13\}$. Our results show that multiple views robustly address occlusion and heavy clutter in the warehouse setting (Table II [clutter] and [occlusion]). They also present a clear contrast between the performance of our algorithm using a single view of the scene, versus multiple views of the scene (Table II [Full] v.s [1v-2v]).

Denoising. The denoising step described in Section V proves important for achieving good results. With this turned off, our method attains less accuracy for both rotations and translations: 4.4% and 6.0% drop (Table II).

ICP improvements. Without the pre-processing steps to ICP, we observe less accurate pose predictions from our method: 0.9% and 3.1% drop in trans. and rot. accuracy (Table II).
Performance upper bound. We also evaluated how well the model-fitting part of our algorithm alone performs on the benchmark by using ground truth segmentation labels from the benchmark as the performance upper bound.

D. Detecting failure modes

Failure modes. Here we summarize the common failure modes of our system, which are illustrated in Figures 10: (1) Objects in the face of heavy occlusion or clutter are likely to have incomplete or missing segmentation from the FCN, resulting in suboptimal pose predictions (Figure 9.e), and completely undetected objects towards the back of the bin (Figure 10.m and p). (2) Objects textures may be confused with each other. In Figure 10.r algorithm combined object textures from dove bar on scotch mail and be mistaken it as outlet plugs. (3) Model fitting is not precise with cuboid objects (wrong corner alignments and insufficiently observation for the marker boxes in 10.o). However, this inaccuracy are still within the range of tolerance for robot picking with sensor-guarded motions.

Detecting failure modes by confidence score. Since mistakes are often very costly, we compute a confidence score per object pose prediction that favors high precision for low recall. Specifically, the final confidence score of a pose prediction for object equals the mean value of confidence scores over all points belonging to the segmentation of object. We find that erroneous poses (especially those of very occluded objects) more often have low confidence scores, and therefore are ignored by the robotic system. We investigate the usefulness of our confidence scores by evaluating the subset of predictions with confidence scores larger than 10% and 70% respectively (see Table II). These confidence percentages are important thresholds because in the full robot system, prediction with < 10% confidence (conf-10, at 78% recall) will be ignored during planning, while prediction with > 70% confidence (conf-70, at 23% recall) will allow a pick attempt.

IX. DISCUSSION

Despite tremendous advances in computer vision, it is apparent that many well-known approaches are not sufficient for even relatively simple and common scenarios. Below, we describe our two primary observations in constructing a vision system for robotic scenarios.

Making the most of every constraint. External constraints limit what our systems can do, but they can also simplify our implementation or provide assumptions we rely on to make them more robust. Occasionally, constraints create unexpected solutions. As an example, in the picking task, each team received a list of items, their bin assignments, and a model of the shelf. All teams used the bin assignments to rule out objects from consideration and the model of the shelf to calibrate their robots. These optimizations are straightforward and profitable. However, further investigation yields more opportunity. By using these same constraints, we were able to construct a self-supervising mechanism to train a deep neural network with significantly more data. As our evaluations show, the volume of training data is strongly correlated with the performance.

Designing robotic and vision systems hand-in-hand. Vision algorithms are often designed in a vacuum. However, vision is one component, in a robotic system. Typical computer vision algorithms operate on a single image for segmentation and recognition. Robotic arms free us from this constraint, allowing us to precisely fuse multiple views and improve our performance in cluttered environments that obstruct single views. Typical computer vision algorithms have fixed outputs (e.g., bounding boxes or 2D segmentation maps), but a robotic system with multiple manipulators might require different outputs. Such a system may be more complex,
but affords the robot more options in completing its task. For example, suction cups and grippers perform better on different objects, but the former uses an object’s segmented point cloud and the latter uses the object’s 6D pose.

X. CONCLUSION

In this paper, we present the vision system of Team MIT-Princeton’s 3rd- and 4th-place entry in the 2016 Amazon Picking Challenge. To address the challenges posed by the warehouse setting, our framework leverages multi-view RGB-D data and data-driven, self-supervised deep learning to reliably estimate the 6D poses of objects under a variety of scenarios. We also provide a well-labeled benchmark dataset of APC 2016 containing multi-view images of 477 scenes.

REFERENCES

