Abstract—We want to build robots that are useful in unstructured real world applications, such as doing work in the household. Grasping in particular is an important skill in this domain, yet it remains a challenge. One of the key hurdles is handling unexpected changes or motion in the objects being grasped and kinematic noise or other errors in the robot. This paper proposes an approach to learning a closed-loop controller for robotic grasping that dynamically guides the gripper to the object. We use a wrist-mounted sensor to acquire depth images in front of the gripper and train a convolutional neural network to learn a distance function to true grasps for grasp configurations over an image. The training sensor data is generated in simulation, a major advantage over previous work that uses real robot experience, which is costly to obtain. Despite being trained in simulation, our approach works well on real noisy sensor images. We compare our controller in simulated and real robot experiments to a strong baseline for grasp pose detection, and find that our approach significantly outperforms the baseline in the presence of kinematic noise, perceptual errors and disturbances of the object during grasping.

I. INTRODUCTION

Recently, deep neural networks have been used to learn a variety of visuomotor skills for robotic manipulation including grasping, screwing a top on a bottle, mating a mega-block, and hanging a loop of rope on a hook [1]. Grasping is a particularly useful and ubiquitous robotics task. A number of researchers have recently proposed using deep learning for robotic grasping systems that perform well for novel objects presented in dense clutter [2, 3, 4]. However, these systems still do not perform as well as we would like, achieving maximum grasp success rates of approximately 85% to 93% in ideal conditions [2]. The question is how to learn robotic grasping or manipulation behaviors that are robust to perceptual noise and kinematic inaccuracies in realistic conditions.

A major problem with many existing approaches is that they perform one-shot grasp detection and thus cannot learn dynamic correcting behaviors that respond to changes in the environment. One promising solution is to learn a closed-loop visuomotor controller. In contrast to one-shot grasp detection, closed-loop controllers have the potential to react to the unexpected disturbances of the object during grasping that often cause grasps to fail. The recent work by Levine et al. [5] used supervised deep networks to learn a closed-loop control policy for grasping novel objects in clutter. However, their approach has two important drawbacks. First, it requires visual data that observes the scene from a specific viewpoint with...
This paper proposes an approach to closed-loop control for robotic manipulation that is not subject to either of the two limitations described above. We make three key contributions. First, in order to eliminate the dependence on a particular viewing direction, we mount a depth sensor near the robot end-effector as shown in Figure 1. In this configuration, the same visuomotor controller can be used to grasp objects from any direction, because the camera to gripper configuration is fixed. Second, we train the system completely in simulation, thereby eliminating the dependence on enormous amounts of real-world robotic training experience. The key to training in simulation is our use of depth sensors rather than RGB cameras. While depth data is potentially less informative than RGB data, it can be simulated relatively accurately using ray tracing (we use OpenRA VE [6]). Third, we propose a novel neural network model that learns a distance-to-nearest-grasp function used by our controller. Our convolutional neural network (CNN) is similar in structure to that of Levine et al. [5], but takes images at a lower resolution and has many fewer layers. Instead of learning a policy directly, we learn a distance function, i.e., distance to grasp, using CNN regression with an L1 loss function. This function provides direct feedback about how viable a grasp is and allows us to use a simple controller to move the robot arm. We evaluate the performance of the system both in simulation and on a UR5 robot in our lab. Our major finding is that in the absence of motor or sensor noise, our closed-loop grasp controller has similar performance to a previously developed grasp detection method [2] with very high grasp success rates. However, under realistic motor, kinematic and sensor errors, the controller proposed here outperforms that method significantly.

II. RELATED WORK

Recent work in grasp perception has utilized deep learning to localize grasp configurations in a way that is analogous to object detection in computer vision [3, 7, 8, 4]. Such methods take potentially noisy sensor data as input and produce viable grasp pose estimates as output. However, these grasp detection methods typically suffer from perceptual errors and inaccurate robot kinematics such as in our previous work [2].

Visual servoing methods use visual feedback to move a camera to a target pose that depends directly on the object pose. While there are numerous methods in this area [9], only a small amount of previous work addresses using visual feedback directly for grasping [10, 11, 12]. In contrast to our work, the existing methods require manual feature design or specification. An active vision approach by Arruda et al. acquires sensor data from different view points to optimize surface reconstruction for reliable grasping during grasp planning [13]. However, the actual grasping does not use sensor feedback.

Levine et al. were one of the first to incorporate deep learning for grasp perception using visual feedback [5]. However, their approach requires months of training on multiple physical robots. Moreover, they require a CNN with 17 layers that must be trained from scratch. In addition, their use of a static camera makes it difficult to adapt to different grasping scenarios, e.g., a different table height or a different grasp approach direction. Because we generate training data in simulation and our CNN has only a few layers, our approach is simpler. In addition, since we mount the camera to the wrist of the robot arm, our approach is more flexible because it can be applied to any grasping scenario – not just those with a particular configuration relative to the camera.

III. APPROACH

We propose a new approach to the problem of learning a visuomotor controller for robotic grasping inspired by the
method of Levine et al. [5]. We mount a depth sensor near the wrist of our robot as shown in Figure 1. On each control step, the system takes a depth image of the scene directly in front of the gripper and uses this sensor information to guide the hand. The controller converges to good grasp configurations from which the gripper fingers can close and pick up the object. The approach is based on a convolutional neural network that learns a distance function. It takes the depth image in conjunction with a candidate hand displacement as input and produces as output an estimate of the distance-to-nearest-grasp. Figure 2 shows an overview of the approach. The key elements are: 1) the convolutional neural network that is used to model the distance function (Section III-A); 2) the approach to generating the training set in simulation (Section III-B); 3) the implementation of the controller (Section III-C).

A. CNN Model

The core of our work is a convolutional neural network (a CNN, see Figure 2) that learns a distance function that is used by our grasp controller. The network takes as input a depth image, \( I \), and an action, \( a = (x, y, \theta) \in \mathbb{R}^2 \times S^1 \). The action denotes a candidate planar pose offset relative to the depth sensor to which the robotic hand could be moved. It learns a real-valued function, \( d(I, a) \in \mathbb{R}_{\geq 0} \), that describes the distance between the hand and the nearest viable grasp after displacing the hand by \( a \). We interpret this distance to be the remaining cost-to-go of moving to the nearest viable grasp after executing action \( a \). Distance is measured in meters in the \((x, y, \theta)\) pose space by weighting the angular component (by 0.001 meter/degree) relative to the translational parts.

Our CNN is based on the LeNet network designed for handwritten digit classification [14]. It consists of two convolutional layers (Conv1 with 20 and Conv2 with 50 filters, kernel size 5, and stride 1) with leaky RELUs, max pooling and 2 inner-product (IP) layers with leaky RELUs. Inspired by Levine et al. [5] we apply an IP layer to the input pose vector (action) and then tile the resulting output over the spatial dimensions to match the dimensions of the Pool1 layer and sum element-wise. The output layer predicts the distance-to-go for the grasp pose action. Since we are learning a real-valued distance function, our CNN is solving a regression problem. We use L1 loss function.

B. Generating training data

We create a dataset in simulation using OpenRAVE [6] comprised of image-offset pairs and the corresponding distance-to-nearest-grasp labels. The way that OpenRAVE simulates the depth images is of particular interest. If the simulated images are sufficiently different from the images generated by an actual depth sensor, then this would produce a gap that would make it difficult to transfer the learned model onto the real robot. Fortunately, we found that this was not the case. The model learned on depth images generated by OpenRAVE (using ray tracing) seems to transfer well.

In order to train the CNN, we generate a large number of image-action pairs, each associated with a distance-to-nearest-grasp label. We accomplish this using OpenRAVE as follows. First, we generate 12.5k different scenes with a random selection of multiple objects placed under the sensor. The objects were derived from CAD models contained within the 3DNet database [15]. In particular, we have selected 381 graspable objects from the following 10 categories: mug, hammer, bottle, tetra pak, flash light, camera, can, apple and toy car. There are between 1-5 CAD objects in each scene. Each object is placed with a random position and orientation.

For each scene we generate 40 depth images by placing the camera randomly in \((x, y, z, \theta)\) above the objects, where \(x,y\) are the directions parallel to the table and \(z\) is the direction towards the table. This results in a total of 500k depth images. Each depth image has one color channel and has a size of \(64 \times 64\) pixels. For each depth image we uniformly sample 10 offset poses within the camera view and calculate the distance to the nearest grasp for each pose as follows. First, using the mesh model of the scene, we sample a large number of grasp candidates by filtering for robotic hand poses that are collision free and that contain parts of the visible environment between the robotic fingers (see [16]). Then, we test each candidate for force closure using standard methods [17]. Finally, after pruning the non-grasps, we evaluate the Euclidean distance to the nearest sampled grasp (see Figure 3).

C. Controller

Our controller takes actions that descend the distance function that is modelled by the CNN described in Section III-A. Its basic operation is outlined in Algorithm 1. The controller starts with the hand at a fixed initial height above the table in the \(z\)-direction. In Step 3, the controller acquires an image from the wrist-mounted depth sensor. In Step 4, it samples a set of candidate actions and selects the one with the minimum distance-to-nearest-grasp. In Step 5, the controller moves by a constant fractional step size in the direction of the selected action. In Step 6, the controller approaches the object in the \(z\)-direction by one step. This process repeats until the controller converges and the hand reaches the final hand height.

An important point is that we constrain the sampling to a region around the origin in the image. This enables us to capture the gradient of the distance function in the neighborhood of the current hand pose. The distance function may be...
a multimodal function in the space of offsets. For purposes of
stability, it is important for the controller to follow a gradient.
This is illustrated in Figure 4. Although the global minimum
in the distance function is on the left, the hand will follow
the gradient to the right. The controller thus grasps the object
closest to the current hand pose, regardless of its identity. If our
goal was to grasp a desired target object, our approach could
be extended to first run object detection and then sample grasp
candidates near the target object, e.g., within a bounding box
around it.

**Algorithm 1 Controller**

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r$ (step ratio to target pose)</td>
</tr>
<tr>
<td>$z_{step}$ (step size towards table)</td>
</tr>
<tr>
<td>$h_{max}$ (initial camera height)</td>
</tr>
<tr>
<td>$h_{min}$ (final camera height)</td>
</tr>
<tr>
<td>$\text{min}<em>\text{dist}</em>\text{obj}$ (min distance to object)</td>
</tr>
</tbody>
</table>

1: Initialize camera at $z = h_{max}$ above table
2: While $z > h_{min}$ AND $\text{dist}_\text{obj} > \text{min}_\text{dist}_\text{obj}$
3: Acquire sensor depth image
4: Sample actions; select min-cost offset, $\text{pose}_\text{top}$
5: Move camera to $r \cdot \text{pose}_\text{top}$
6: $z := z - z_{step}$
7: EndWhile
8: Execute fixed grasp motion (robot)
9: Evaluate distance to closest true grasp (sim)

Fig. 4. Illustration for how the controller works in a 1-dimensional case with
two objects (control in x-axis direction). Although the global best prediction
for the grasp pose belongs to the object on the left, the controller moves to the
closer object on the right, because it follows the direction of the local
gradient near the center of the image.

IV. SIMULATION EXPERIMENTS

We perform a series of experiments in simulation to evaluate
our new grasp controller (CTR) relative to grasp pose detection
(GPD), a recently proposed one-shot method that also learns
in simulation and achieves high success rates [2]. We perform
this comparison for two scenarios: one where the manipulator
moves exactly as commanded and one where the desired
manipulator motions are corrupted by zero-mean Gaussian
noise. All of the following simulation data are averages over
400 trials. In each trial, we generate a scene in OpenRAVE
with a random selection and placement of objects from the test
set as described in Section III-B. The initial camera position
is set to 0.3 m above the table. At each iteration the camera
height is reduced by a constant step until height 0.15 m is
reached. We run the controller for a total of 75 iterations,
using $r = 0.2$ as the step ratio to a target pose, and plot the
distance of the final gripper pose to the closest true grasp.

We use the deep learning framework Caffe [18] for training
the network. We run 900k iterations of stochastic gradient
descent with a learning rate of 0.001, a momentum of 0.9,
and a batch size of 1k instances. The dataset is described in
Section III-B. We split the training and test sets on object
instances. Both the training and test sets contain all 10 object
categories. However, the same object instance does not appear
in both sets. The training set contained 331 object instances
and the test set contained 50. We use the same network for all
experiments, including experiments on the robot in Section V.

A. Comparison with GPD baseline in the presence of kinematic noise

We compare the following scenarios: 1) **INIT**: No camera motion, distances measured from the initial position; 2) **CTR**$_{\text{no noise}}$: Run CTR starting from the initial position, without kinematics noise; 3) **CTR**$_{\text{with noise}}$: Run CTR from the initial position, with kinematics noise; 4) **GPD**$_{\text{no noise}}$: Move to top GPD prediction, without kinematics noise; 5) **GPD**$_{\text{with noise}}$: Move to top GPD prediction, with kinematics noise. The “with noise” scenario simulates the situation
where zero mean Gaussian noise ($w \sim \mathcal{N}(0,1)$) is added to
the robotic hand displacements on each control step:
\[
\Delta(x, y, \theta)_{\text{noisy}} = \Delta(x, y, \theta) + 0.4w + \|[\Delta(x, y, \theta)]
\] (1)

While this noisy-motion scenario is not entirely realistic,
we nevertheless think this is a good test of the resilience
of our controller to kinematic errors. The final distances to
the closest true grasp for the 5 scenarios above is shown in
Figure 5. Note that we only consider the distance in $(x, y, \theta)$
and not in $z$, because we assume that the distance to the object
can be retrieved easily from the depth image. We convert the
distances for $\theta$ from degrees to meters as described in III-A.

![Distances of grasps to closest true grasp](image)

Fig. 5. Histogram for distances of grasps to closest true grasp for 400 sim-
ulated trials for various scenarios. This shows that our approach compensates
well for movement noise of the gripper, where a GPD baseline method fails
to compensate.
is comparable. However, CTR does much better than GPD in scenarios with motion noise. This makes sense because the controller can compensate to some degree for motion errors while GPD cannot.

**B. Correction for perceptual errors made in single-shot prediction using the controller**

Next we compare the following two scenarios to characterize the advantages of the closed-loop controller versus one-shot detection: 1) **CNN only**: Move to the top one-shot global prediction using the CNN regression model; 2) **CNN + CTR**: Move to the top one-shot global prediction and then run the controller.

![Fig. 6](image)

Figure 6 shows the controller improves the performance of our approach even in a scenario without kinematic noise. This suggests that the controller can compensate for perceptual errors made in a single depth image.

**V. ROBOT EXPERIMENTS**

We evaluate our grasp controller on the UR5 robot in three experimental scenarios: (i) objects in isolation on a tabletop, (ii) objects in dense clutter, and (iii) objects in dense clutter with a shift in position after a few controller iterations. In these scenarios, we compare our controller (CTR) to grasp pose detection (GPD), a strong baseline [2]. A summary of our experimental results is given in Table I.

![Fig. 7](image)

**Figure 7.** Experimental setup with UR5 robot and SR300 sensor.

![Table I](image)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>CTR</th>
<th>GPD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objects in isolation</td>
<td>97.5%</td>
<td>97.5%</td>
</tr>
<tr>
<td>Clutter</td>
<td>88.9%</td>
<td>94.8%</td>
</tr>
<tr>
<td>Clutter with rotations</td>
<td>77.3%</td>
<td>22.5%</td>
</tr>
</tbody>
</table>

We run the controller with a step ratio of $r = 0.5$ and a $z$ step size of 1 cm. Each depth image from the sensor is post-processed to remove invalid depth readings near the edges of objects by applying an iterative, morphological dilation filter that replaces the missing values with a neighborhood maximum. IKFast [6] is used to convert the selected action (i.e. a Cartesian displacement) into target joint angles for the arm to move to. The controller runs until the depth data indicates that an object is within 14 cm distance from the sensor or the robot hand is too close to the table. To execute the grasp, we move the robot hand according to a predefined motion and close the fingers. The GPD baseline runs as follows. We move the robotic hand to a fixed point above the table pointing directly down and take a depth image. Then, we select and execute one of the detected grasps based on the heuristics outlined in prior work [2].

**A. Grasping objects in isolation**

In this experiment, each of the ten objects from our test set is presented to the robot in four different configurations. Only one out of the 40 grasp attempts results in a failure (97.5% grasp success rate). This failure happens on the bottle in an upright configuration where our controller is not able to converge to the correct pose. Given more iterations, the attempt would have been successful. In this scenario, GPD achieves the same success rate.

**B. Grasping objects in dense clutter**

We also evaluate our grasp controller in dense clutter. Here, we put the ten objects from our test set into a box, shake the box to mix up the objects, and empty it onto the table in front of the robot. An example of such a scenario is shown in Figure 8. A run is terminated when three consecutive executions result in grasp failures or when the remaining
over the ten runs of this experiment, the robot attempts 74 grasps using our controller, where eight are failures (88.9% grasp success rate). Five out of the eight failures are caused by the object slipping out of the fingers during the grasp, two are caused by slight misplacements in the final pose, and one is caused by a collision between a finger and the object. In comparison, the GPD method attempted 96 grasps over ten runs of this experiment. Only five were not successful (94.8% grasp success rate). Out of the five failures, two are perceptual errors and three are caused by the object slipping out of the hand while closing the fingers. While GPD achieves a higher grasp success rate than our controller in this scenario, we think that the controller would achieve a similar performance if it were given more iterations and the correction movement during each iteration was smaller (such as we did for the simulation).

C. Grasping objects with changing orientations

This experiment evaluates the performance of our controller versus the GPD baseline for a dynamic scenario where the human experimenter manually shifts the positions of the objects once during each grasp trial. To accomplish this, we pour the pile of cluttered objects onto a piece of paper and then shift the paper by a random amount after the third controller iteration. Over the ten runs of this experiment, the robot attempted 75 grasps using our controller. 17 attempts were failures (77.3% grasp success rate). In comparison, GPD only attempted 49 grasps, out of which 38 were failures (22.5% grasp success rate). The better performance of our controller makes sense because it is able to react to the shift whereas GPD cannot: it simply proceeds to execute the grasp as if the object pile had not been shifted. This is a general advantage of a closed-loop controller relative to typical grasp perception methods [4, 8, 3, 7, 2].

VI. DISCUSSION

We developed a visuomotor controller that uses visual feedback from the depth sensor mounted on the gripper to dynamically correct for misalignment with the object during grasping. We trained a deep CNN model with simulated sensor data that directly learns the distance function for a given depth image and grasp pose action. Generation of training data in simulation was more efficient than generation on a real robot. We found that the CNN model trained with simulated depth images transfers well to the domain of real sensor images after processing images to correct invalid depth readings.

Our controller was able to react to shifting objects and to inaccurate placement of the gripper relative to the object to be grasped. In simulation experiments, our approach compensated for significant noisy kinematics while a one-shot GPD baseline did not. Moreover, our controller using the CNN model corrected for perceptual errors present in one-shot prediction. Real world experiments demonstrated that our method also works well on the robot with noisy sensor images. Our performance was comparable to the GPD baseline. We expect to improve our controller further, such that we can execute more and smaller corrections while moving the gripper faster during grasping. Results in simulation showed that, by using a controller with fine-grained adjustments, we can exceed the performance of the GPD baseline, especially in the presence of kinematic noise. The UR5 robot used in our experiments has fairly precise forward kinematics. For experiments on a robot with more noisy kinematics we expect to see a significant advantage of our method.

ACKNOWLEDGEMENT

This work has been supported in part by the National Science Foundation through IIS-1427081, NASA through NNX16AC48A and NNX13AQ85G, and ONR through N000141410047.
REFERENCES


