Robotic Grasp Pose Estimation

Bryant Har
bjh254

Nathaniel Chin
nlc62

Tony Yang
txy3

1 Introduction

Grasping objects is a fundamental skill required for many daily tasks such as cooking, eating, and cleaning. Humans perform this skill very well because we can perceive objects clearly and our hands are very dexterous. Through a lot of practice grasping a variety of objects for many different purposes, we are generally efficient in grasping objects in a way that is also convenient for us to perform a desired task with the object. Therefore, we want to be able to use our experience of grasping objects to teach a robot arm to grasp in a similar manner. The impact of this research is that robots can precisely grasp objects in a task-oriented manner. This opens doors for robots to accomplish tasks such as cutting vegetables with a knife or using a pepper shaker. These complex tasks require the robot to grasp the objects in a specific way so that they can be used for its intended purposes.

2 Problem Statement

The problem we tackled for this research project is pre-grasp pose estimation. More specifically, given a 3D point cloud of an object of interest, we sought to find a suitable pre-grasp end-effector pose for the Franka Emika Panda to successfully grasp the object. In this project, the object of interest is a simple bottle of pepper, though our methodology is scalable and can easily be generalized to any class of object or even to raw point cloud data.

3 Review of Pointnet++

Throughout the design process, we debated between several existing architectures for our network. In particular, there were a few existing frameworks that catered to similar and relevant tasks but were ultimately ill-suited for our specific goal. One notable paper was PoseFusion,[1] which proved to be too unwieldy and relied on methods we deemed to be excessive for our far simpler task.

The primary paper off which we based our model was the seminal Pointnet++ paper,[2] which developed a framework for learning hierarchical features, or features with a structural dependence on each other. The paper offered a strong insight into the architecture of their networks, which would prove invaluable in modifying the existing implementations thereof. For our purposes, we based our final network on two existing Pointnet++ implementations.[3] Critically, these implementations aggregated and incorporated a few updates and optimizations made to the original Pointnet++ paper since its release. As such, referencing these implementations and building off of them proved to be invaluable resources for the development of our architecture. These three resources formed the backbone of our methodology and enabled us to achieve reasonable results for our task.

[1]Reference 4
[2]Reference 1
[3]References 2, 3

4 Approach

Our goal is to teach a robot arm to grasp an object from a table and raise it off the surface. Specifically, we will teach a Franka Emika Panda robot arm to grasp a bottle and raise it using a PointNet++ network to imitate the behavior of an expert demonstration i.e. Nathaniel. In our training, we will use a point cloud of the bottle as inputs and train the network to output a 3-dimensional translation vector for the arm to be in pre-grasp. Once we outputted this vector along with a perpendicular rotation with the z-axis, we used a controller to actuate that pose. From the pregrasp pose, the object was grasped by just closing the gripper. We measured how successful the grasp is based on whether or not the robot grasped the object. To this end, we had three primary steps to our approach:

1. Data Collection: To train the PointNet++ model, first we collected a dataset of point clouds of a bottle in different positions on the table and different viewing directions. The labels for the dataset were the corresponding pregrasp pose on the bottle. To this end, we manually demonstrated a proper grasp, record the robot pose, and generated the point cloud for the object. This dataset will be used to train the model to recognize the bottle and how to approach grasping them.
2. Model Training: We will use this dataset as inputs to PointNet++. However, the PointNet++ architecture is originally meant for segmentation and classification tasks, but we wanted our task to be for regression. Then, we trained our network on 10 point clouds and ran the model on a test point cloud to get a predicted end-effector pose. Of the 6D end-effector pose, we only estimated the 3D translational vector, which served as our labels. We had access to computers with GPUs so that helped expedite the training process. This process is explained in further detail in the next section.
3. Deployment/Final Evaluation: We evaluated the model by creating a separate test dataset of the bottle in ten different locations. We then ran the point clouds of the object through the model and used the predicted grasp poses to execute a grasp on the bottle. To quantify the error, we calculated the L2 norm between each label and predicted pose and averaged them across the dataset.

5 Methodology

The overall structure of our network training methodology is fairly straightforward.

Our labels were the 3D translational coordinates of the ground truth end-effector poses truncated from the full 6D pose. The data itself was the raw point cloud data of the 10 samples we obtained, each of which approximately contained 20,000 to 30,000 points. In our preprocessing, we sparsified and downsampled the point cloud data so that our cloud only contained 1024 datapoints. Then, in our training loop, we fed the data into the Pointnet++ point cloud embedding networks. Then, this was passed into the feature network. In most of the implementations we found, including the ones we referenced, were designed with classification tasks in mind. Critically, the implementations we chose were written in Pytorch, making our work of modifying the architecture much easier.

The conversion to a regression network from a classification network only required removing a few batching and pooling layers from the forward pass and changing the last layer from a log softmax layer (which generated predicted class probabilities) to a linear layer. We then fed this into another linear layer that outputted three values, representing the estimated translational coordinates of the ideal end-effector pose. Since we were dealing with regression instead of classification, we also switched the loss function from negative log likelihood loss to MSE loss. We then ran the training loop for 25 epochs, processing a single point cloud in each iteration. Overall, training the model took around 1 minute at an average rate of 4.81 iterations per second. Importantly, at each iteration, each point cloud was downsampled randomly to reduce the chance of overfitting, since the network could now train on multiple possible different representations of the same point cloud.

6 Limitations

There were numerous limitations to our approach, limited by fundamental issues in our design.

- Data Acquisition: Our dataset was limited by the scale of our capabilities. Only around 30 to 40 datapoints could be acquired per hour by the nature of our setup. Therefore, we were only able to acquire a few dozen samplings of the robot for our dataset. This was further exacerbated by the pruning of our dataset for incorrect or invalid datapoints. We manually sorted through our data and had to remove all point clouds with obvious aberrations or incorrect labels. Overall, our final dataset only consisted of 10 usable samples.
- Data Coverage: The data samples we acquired for our training were generally very similar to one another. That is, we didn't try adding more varied examples like the bottle closer to the robot, farther from the robot, or sideways in order to expedite our data acquisition process. Collecting data such that the model is invariant to translations of the object and different viewing orientations for the point cloud is tedious. Our data collection also does not scale well to more objects in which the objects themselves can have different orientations. Therefore, constructing a dataset that has reasonable coverage of these possibilities is a difficult task. Having a greater coverage of datapoints nevertheless would make our model more robust to different poses of the bottle and even different types of objects. However, out of convenience, we only acquired samples of the bottle of pepper in very similar positions.
- Data Quality: Depending on the viewing direction of the robot, the point cloud of the object might not depict much of the full nature of the object the robot is trying to grasp. This means that the point cloud does not provide much information about the object it is trying to grasp. Therefore, when the model tries to learn from the incomplete point cloud how to grasp objects, there can be inaccuracies in the predicted grasp pose.

7 Results

Using our modified PointNet++ architecture, we trained our model using 10 different point clouds over 25 epochs. As seen in Figure 1 below, our MSE training loss initially started quite high during the first first few epochs, but sharply dropped down and converged to loss values ranging from 0 and 1. To gauge the potential of overfitting, we gathered two additional point clouds as a test dataset and fed the model prediction to the arm, which executed the end-effector pose. The results of these live experiments are shown in the video and in Figure 2.

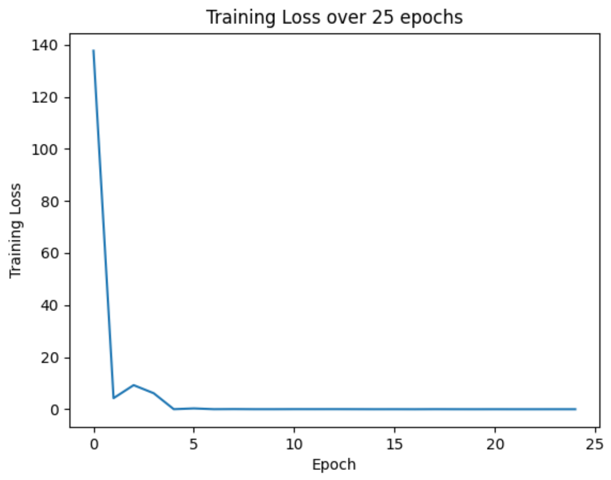

Figure 1: Graph displaying the training loss over 25 epochs

Observing the final end-effector pose of the Franka Emika robot in Figure 2, it is clear that the robot did not grasp onto the pepper bottle successfully. Although it did not meet our evaluation metric of whether or not the grasp is successful or not, our grasp position was only off by a couple of centimeters in each coordinate direction.

To further analyze our results, we calculated the L2 norm between our predicted x, y, z positions and the labels of 10 distinct point clouds from the point clouds we used for training. As seen in Figure 3 below we get an average L2 norm of 0.111997136. This meant that on the training data, the model predicts an end-effector pose that is only off on average around 0.112 meters from the object.

8 Conclusion

The inaccuracy of our predicted end-effector pose can be largely attributed to the fact that our model had very few examples to train on during the training loop. We only used a total of 10 point clouds for training, which is rather small, making it inevitable that the resultant performance would have high variance in performance. The fact that the arm missed the bottle of pepper suggests that the model slightly overfit to our data. However, the L2 norms calculated in Figure 3 seem to suggest that the overfitting may not be too severe, as the disparities between the model predictions and the ground truths in the training set were similarly significant and nontrivial at around 0.112 meters. Specifically, the training set also suffers from suboptimal predictions. Importantly, a critical reason why our model did not overfit as severely as we expected was because of the significant pooling layers added to the architecture of our network, which downsampled our features and hidden layers and partially controlled the overfitting.

Still, simply getting within centimeters of the bottle in the testing phase is nevertheless a promising result. These results hint that our architecture has the potential to generalize well to unseen data if only we had trained the model more rigorously and provided more samples. Of course, the model pose estimation accuracy missed our expectations, but the end result was understandable given how little data was fed into the network. Considering these results, we believe that a more robust model is very feasible with our approach, given a larger dataset.

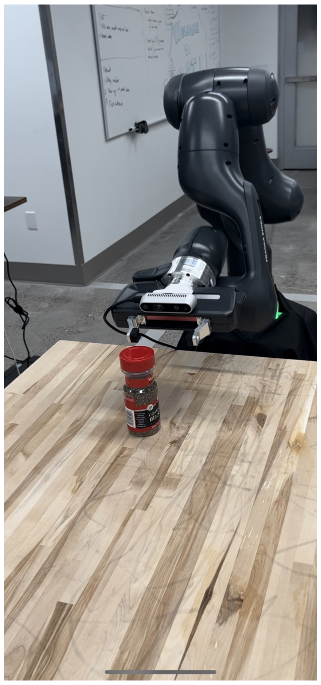
Figure 2: Final position of the Franka Emika robot when executing a grasp on the bottle

9 Contributions

Nathaniel Chin: Collected the dataset to train the model on. This involved setting up the realsense camera on the Franka Emika Panda robot and recording the point clouds of the detected bottle. For

Norm between the Prediction and Labels of 10 Point Clouds			
Point Cloud Number	Predictions (x,y,z)	Labels	L2 Norm
1	[0.4754, 0.0861, 0.2297]	[0.5068, 0.0176, 0.2525]	0.087474148
2	[0.4653, 0.0898, 0.2331]	[0.5506, 0.1267, 0.2552]	0.095463963
3	[0.4655, 0.0899, 0.2331]	[0.5527, 0.0107, 0.2550]	0.119860165
4	[0.4626, 0.0886, 0.2323]	[0.5527, 0.0107, 0.2550]	0.121228391
5	[0.4655, 0.0899, 0.2331]	[0.5506, 0.1268, 0.2552]	0.095275002
6	[0.4513, 0.0880, 0.2313]	[0.5527, 0.0107, 0.2550]	0.122089655
7	[0.4547, 0.0849, 0.2287]	[0.5527, 0.0107, 0.2550]	0.125714075
8	[0.4577, 0.0863, 0.2298]	[0.5527, 0.0107, 0.2550]	0.124029232
9	[0.46457, 0.0900, 0.2332]	[0.5674, 0.0062, 0.2542]	0.133374970
10	[0.4654, 0.0898, 0.2331]	[0.5506, 0.1267, 0.2552]	0.0954617533
Average Norm: 0.111997136			

Figure 3: Table of the L2 norm of the prediction and labels of many point clouds. After running the model on the test point clouds, we took the average of each individual point cloud's L2 norm

each point cloud, the robot was manually moved to a pre-grasp position and the end-effector pose was recorded for the labels. Cowrote the limitations, methodology, and problem statement sections.

Bryant Har: Developed the neural network and pruned the existing body of literature. Found the proper implementations in Pytorch and worked to convert that architecture into network suited for our regression task. Helped initiate the camera module and worked to troubleshoot parts of the robot. Trained the model and wrote the code for parsing the point clouds. Cowrote the methodology, approach, problem statement, review of Pointnet++ and results sections.

Tony Yang: Helped with modifying the neural network. Trained the model on the point clouds and ran the model through the test data to produce the end-effector pose. Responsible for creating the video and the filming. Cowrote the approach, introduction, conclusion, and results section.

10 References

- Charles R. Qi and Li Yi and Hao Su and Leonidas J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. URL `https://arxiv.org/abs/1706.02413`
- PointNet and PointNet++ implemented by pytorch (pure python) and on ModelNet, ShapeNet and S3DIS. URL `https://github.com/yanx27/Pointnet_Pointnet2_pytorch`
- Pytorch Implementation of PointNet and PointNet++ `https://github.com/yanx27/Pointnet_Pointnet2_pytorch`
- Yuyang Tu, Junnan Jiang, Shuang Li, Norman Hendrich, Miao Li and Jianwei Zhang. PoseFusion: Robust Object-in-Hand Pose Estimation with SelectLSTM `https://arxiv.org/pdf/2304.04523.pdf`