Determining Grasping Regions using Vision

Marcus Lim
Department of Computer Science
Cornell University
Ithaca, NY 14850
limklm@gmail.com

Biswajit Biswal
Department of Computer Science
Cornell University
Ithaca, NY 14850
bb452@cornell.edu

Abstract

Object grasping is one of the fundamental tasks of personal robots. There are several approaches to this problem, some requiring a 3D model of the object. Saxena et al. [1] proposed a learning algorithm which does not require a 3D model; instead it identifies points on the object corresponding to good grasping locations based on computer vision.

Our approach identifies oriented rectangular regions instead of points as this representation is more suitable for a parallel-plate gripper. Since these rectangles can have various locations, sizes and orientations, such a representation is computationally expensive. To tackle this problem, we used histogram features that can be computed in (amortized) constant time for each rectangle. Our learning model is based on a linear SVM ranking model trained using SVM-rank [2].

Results from robotic experiments show that our algorithm performs well on both trained objects and novel objects, indicating that it generalized well to a wider class of objects.

1 Introduction

The goal of our project is to program a robotic arm pick up objects autonomously with computer vision. While this task may seem easy to a human being, programming a robotic arm to pick up objects is not a trivial task, especially if it is an object that the robot has never seen before.

Modern-day robots are capable of carrying out very complex and specialized manipulation tasks such as assembling a car, and they can perform these tasks accurately and precisely. Humans, on the other hand, might have some trouble with achieving the same level of precision, but are capable of performing a wider range of tasks. We are interested in designing robots that can perform general tasks such as picking up an object from the ground autonomously.

This is not an easy task since the robot has to be able to interpret its environment and identify objects, determine suitable grasping configurations and plan its motion from its current configuration to the desired configurations while carefully avoiding obstacles.

There are many related work in this area. If a full 3-d model of the object is available, then approaches such as ones based on form and force closure [3], friction cones [4], or other

methods can be applied. However these approaches cannot be applied to unknown objects for which we do not have a 3-d model of. 3-d reconstruction is difficult for objects without texture, and can only work on visible regions of the object.

In contrast to these approaches, we are working on a robust machine learning algorithm that does not depend on having a 3-d model of the object. It takes as input an image of the scene and an incomplete point cloud from a stereo camera and determines suitable grasping regions on the image. With the 2-d region, we can translate them into 3-d points in the point cloud and determine a suitable grasping configuration of the arm. A path planner can be used to plan a suitable path from the current configuration to the desired grasping configuration that carefully avoids any obstacles along the way. We are working with the Adept Viper robotic arm shown on figure 1.

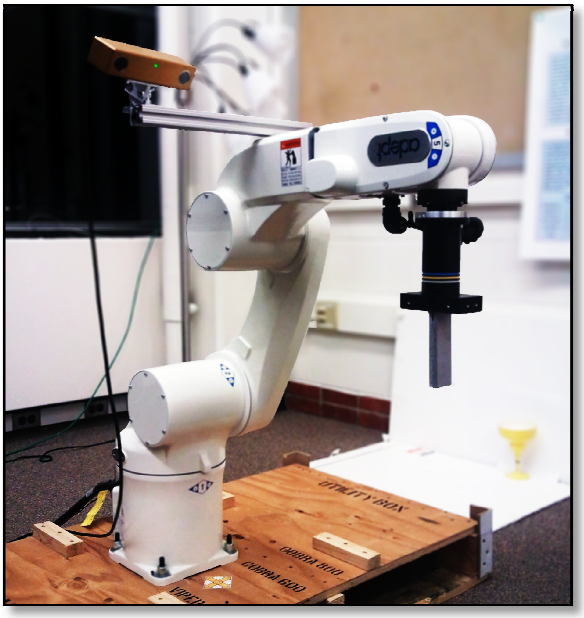

Figure 1: Adept Viper robotic arm with mounted Bumblebee stereo camera

Currently, we restrict grasping regions to oriented rectangular regions in the image as this representation provides enough information to derive a range of suitable grasping configurations [5]. However, if the algorithm performs fast enough, then we can consider extending our search space by including parallelograms or quadrilaterals.

2 Learning model

Instead of directly tacking the problem of finding suitable grasping regions in a given image, we decided to tackle an easier problem: does a given rectangular region on the image correspond to a suitable grasping rectangle? Given such a classifier, we can then extract all rectangles from an image and evaluate them with the classifier to find suitable grasping rectangles.

However, this approach is computationally expensive. Suppose that the classifier take O(k) amount of time to classify each rectangle. To evaluate all rectangles of all orientations (rotations), this would take $O(kn^2m^2p)$, where n and m are the dimensions of the image and p is the number of different orientations. This is a large search space, and for the algorithm to run in a reasonable amount of time, the classifier has to be computationally efficient.

While the set of all possible rectangles is very large, the number of good grasping rectangles is much smaller. Thus, the algorithm has to sieve through most of the negative examples and find the positive ones.

Since we are only interested in identifying the best rectangle, a traditional binary classifier would not be appropriate as it might classify more than one rectangle as positive. Also, with a binary model, we can only separate the data into two classes - good or bad. However, based on our judgment, we can distinguish the best rectangles from the good ones. For example, while it is possible to grasp a cup by its lip, it would be best to grasp it by its handle since that grasp is more stable and would permit easy manipulations if desired in a later stage. Using a binary model, we are unable to input such information into the model.

SVM-rank [2] is a variant of the traditional SVM that is designed to learn ranking models. This type of problem is called ordinal regression, where the label y_i of an example (x_i, y_i) indicates a rank instead of a nominal class. The goal of ordinal regression is to learn a function $h(x)$ such that for all pairs of examples,

$$h(x_i) > h(x_j) \Leftrightarrow y_i > y_j.$$

This model can be further refined to include such constraints only for pairs of rectangles from the same image / point cloud. Without this relaxation, a good example x_i in one image must have a higher $h(x_i)$ than $h(x_j)$ of all other examples x_j on all other images. This need not be true as we might find examples that are relatively weaker in one image may be better than the best examples on another image. During detection, we are only comparing between rectangles from the same image and thus, if the model is well trained, it should be able to distinguish the best rectangle from the rest of the rectangles. For the sake of computational efficiency, we use a linear kernel for SVM-rank.

3 Data collection

With the camera mounted on the robotic arm, we took images of 80 different objects, each from 2 different viewing angles to yield 181 examples. Based on our evaluation metric (see Evaluation section), we labeled all positive rectangles on each of these images such that there was at least 50% overlap. Figure 4 shows some of these manually labeled positive examples. The negative rectangles are generated randomly to avoid subjective bias. Table 1 shows the number of examples for each object type.

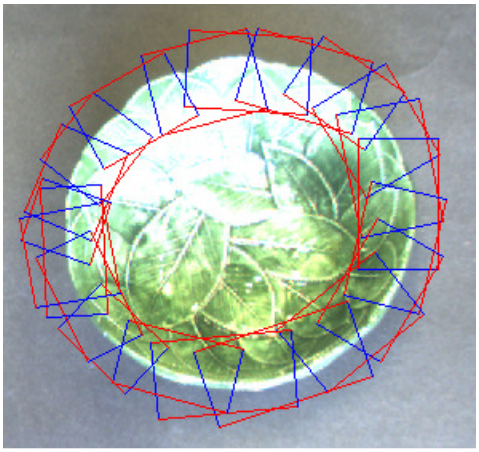

Figure 4: Manually labeled positive rectangles. Red lines indicate the orientation of the rectangle, which corresponds to how the parallel-plate gripper should be orientated to pick up the object.

We split the dataset into approximately 65% training and 35% evaluation, and the division is also shown in table 1.

Table 1: Number of instances of different object types

Object Type	# instances	# training instances	# test instances
Bowl	6	5	1
Cup	43	30	13
Martini	9	5	4
Misc	32	22	10
Plate	8	5	3
Rod	34	26	8
Tongs	8	4	4
Toy	29	21	8
Total	**169**	**118**	**51**

4 Background subtraction

Given the large search space, we decided to use background subtraction to help prune out regions on the image that do not contain the object of interest. Since background subtraction is not our area of expertise, we decided to take the simplest approach of taking two images, one with the object and one without. We then used the Mixture of Gaussians algorithm for background subtraction available in OpenCV [6]. Since we are also interested in the areas around the object, we dilate the result from the background subtraction to yield an object mask. From our experiments, we realized that the background subtraction is prone to excluding parts of the object, so we relaxed our criterion and considered all rectangles that have all four corners in the object mask, so this would also consider areas that would have otherwise been excluded due to the imperfect background subtraction.

5 Features

A. Image features

In order to extract visual cues such as edges, texture, and color, we first transformed the image into YCbCr space. Edge features are detected by convolving the Y (intensity) image with 6 oriented edge filters. Texture information is extracted by applying the 9 Laws' masks to the Y-image, and color information is extracted by applying the first Laws' mask to the color channels (Cb and Cr). The 9 Laws' masks and 6 oriented edge filters are shown in figure 5. As a final step, we normalized all filtered images to between 0 and 1. This is used to allow the features to be robust to illumination changes caused by different lighting situations.

Figure 5: The 9 Laws' masks and 6 oriented edge filters

In order to compute features for rectangles of all sizes, we decided to use normalized fuzzy histograms [7]. Histograms are appropriate since they can be calculated for inputs of all sizes and they describe the distribution of the inputs, and produce a short, fixed-length feature vector. We chose fuzzy histograms in place of normal histograms as they are more robust to small changes in input values. If an input value is near a bin boundary, a small

change in the value can cause a shift from one bin to another in the histogram. This means that values near boundaries are extremely sensitive to noise. To solve this problem, fuzzy histograms are calculated based on (linear) fuzzy partitions (figure 6). For each bin i, we define a bin center c_i, and we allocate each input value y_j to bins $i, i+1$ such that $c_i < y_j < c_{i+1}$ in the manner that bin i receives $1 - \frac{y_j - c_i}{c_{i+1} - c_i}$ and bin $i+1$ receives $\frac{y_j - c_i}{c_{i+1} - c_i}$. In this way, small changes in the value of y_j will lead to a commensurate change in the fuzzy histogram.

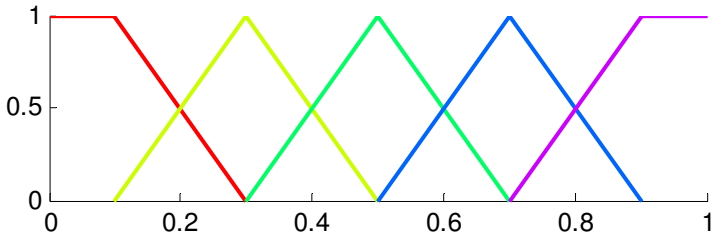

Figure 6: Fuzzy histogram partition functions

In order to capture details in the image, we decided to use 15 bin equally spaced between 0 and 1. We also partitioned each rectangular region into 3 equal-sized horizontal strips. This is useful as it is often the case that the center strip looks very different from the other two strips in positive examples. Thus, allowing this partition allows us to pick up such subtle differences that would otherwise be averaged away if combined into a single histogram for the entire rectangle. Using this partition and the 15 bins, we have a total of $3 \times 15 \times 17 = 765$ image features. Figure 7 shows examples of image feature histograms for positive and negative examples.

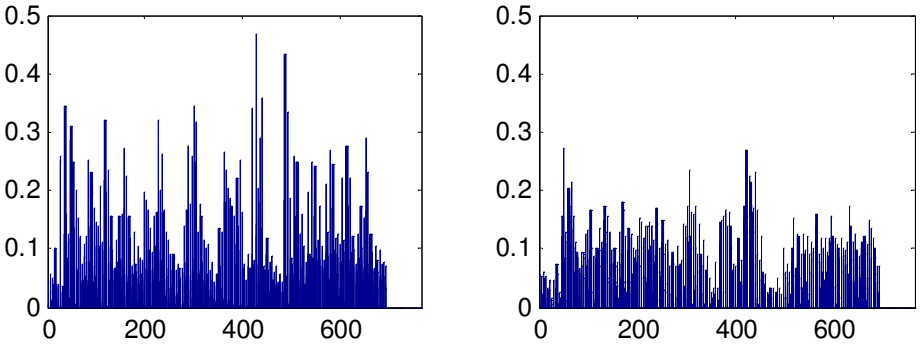

Figure 7: Image feature histograms, positive example on the left and negative example on the right

B. Point cloud features

From the point cloud collected, we calculate the normal vector and curvature at every point in the point cloud by fitting a surface through the point and its 50 neighboring points. Since we are most interested in the z-axis which corresponds to the camera's point-of-view, we ignore the x and y positional and normal information. Using the z position, surface normal in the z-direction and the local curvature, we apply the same 3-strip partitioning and 15-bin fuzzy histogram to yield a total of $3 \times 15 \times 3 = 135$ features (example in figure 8).

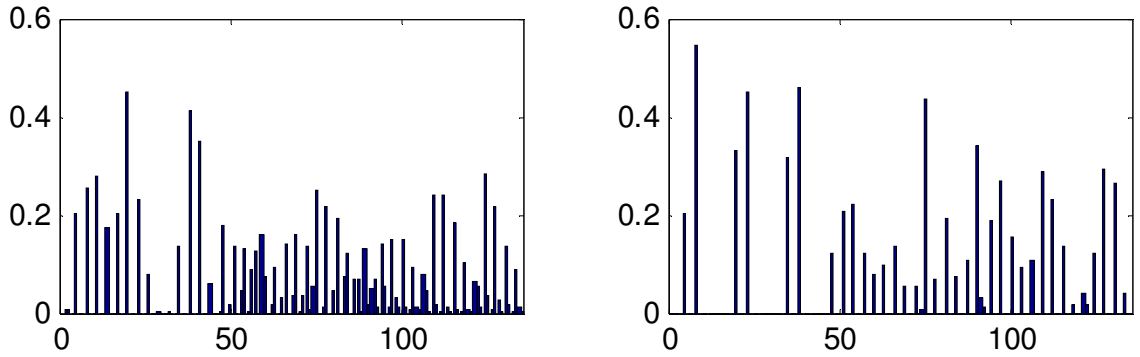

Figure 8: Point cloud feature histograms, positive example on the left and negative example on the right

In order to derive more information from the point cloud, we also calculated the Fast Point Feature Histogram (FPFH) [8] signature for every point. FPFH are informative pose-invariant local features that represent the underlying surface model properties at each point. They are computed based on certain geometrical relations between the point and its neighbors. The FPFH computation produces a 33-bin histogram vector at each point and we sum them up for every point in a given rectangle and normalize them by the number of points in the rectangle. Figure 9 shows an example of the FPFH feature histogram for a positive and a negative labeled rectangle. As we can see, the difference is not very significant, and we expect that this set of features is not very discriminative for our problem.

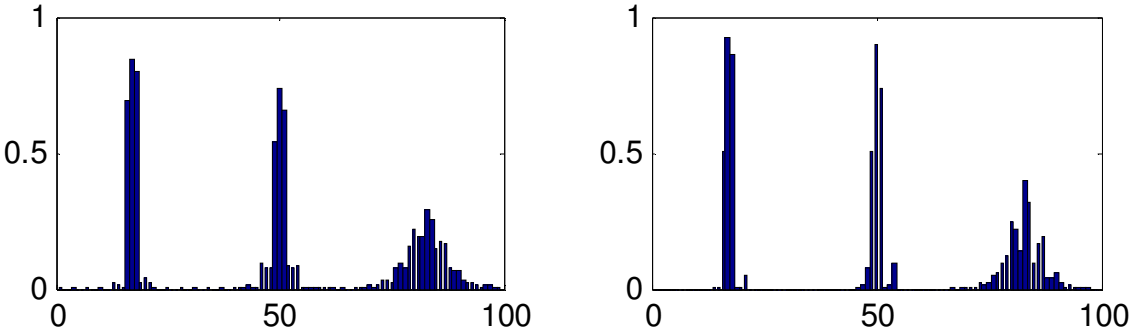

Figure 9: FPFH features, positive example on the left and negative example on the right

6 Evaluation metric

In order to determine the accuracy of our method, we used the metric proposed by Jiang et al [5]. Specifically, we compare our top predicted rectangle with a ground truth rectangle. If the orientation difference is less than 30^o and the ratio of the area of intersection of the two rectangles to the area of the predicted rectangle is more than 0.5, then we consider this as a correct prediction. If there are multiple ground truth rectangles, we repeat the comparison with all the ground truth rectangles and take the best score.

$$Score = 1_{\{\text{Orientation difference}<30^o\}} \times \frac{\text{Area of intersection}}{\text{Area of predicted rectangle}}$$

It is for this reason that our ground truth labels had to be mutually overlapping so that any of the good predicted rectangles will have at least a 50% intersection ratio with one of the ground truth labels.

7 Results and discussion

For comparison, we ran the algorithm by Jiang et al. [5] on our dataset and the results are shown in table 2. In order to gauge the influence of various features, we evaluated the results using various subsets of features. The baseline algorithm is the best score possible by predicting a fixed rectangle in every test instance. To do this, we enumerate all possible rectangles, and calculate the test scores for all of them, and pick out the best scoring rectangle. In addition to the features discussed above, we also calculated non-linear combinations of the features as suggested by Jiang et al. Since we subdivide our rectangles into three horizontal stripes, r1, r2 and r3, we get three equal-length feature vectors for each sub-rectangle. We for each feature in these vectors (f1, f2 and f3), we calculated three non-linear features: f1/f2, f3/f2 and f1×f3/f2.

Table 2: Summary of results of various algorithms

Algorithm	Train	Test
Baseline	-	22.60%
Previous work	40.63%	52.83%
Image features only	56.78%	52.94%
PC features only	50.00%	50.98%
Image + PC features	62.71%	54.90%
Image + PC with non-linear features	66.10%	60.78%
Image + PC + FPFH with non-linear features	66.10%	62.75%

As we can see, our algorithm shows a marked improvement over previous work. However, the test scores are still quite low, and we suspect that this might be because our dataset is more challenging as it contains a wide variety of objects. Table 3 shows the breakdown of the results on various object types. From the results, our trained model is adept at identifying good grasping rectangles in rod-shaped objects and toys, but performs less satisfactorily on bowls and cups.

Table 3: Summary of results for various object types

Object type	Train	Test
Bowl	60.00%	0.00%
Cup	40.00%	30.77%
Martini	40.00%	50.00%
Misc	59.09%	60.00%
Plate	40.00%	100.00%
Rod	96.15%	87.50%
Tong	50.00%	100.00%
Toy	90.48%	75.00%
Overall	66.10%	62.75%

We also conducted experiments with the robotic arm and the results on known objects (objects that were represented in the training set) are as follows:

Table 4: Known object experimental data

Object Type	# instances	Prediction	Grasping
Bowl	4	50.0%	100.0%
Cup	13	92.3%	72.7%

Misc	7	85.7%	66.7%
Plate	2	100.0%	100.0%
Rod	23	95.7%	73.9%
Tongs	4	100.0%	25.0%
Toy	5	80.0%	100.0%
Grand Total	**58**	**89.7%**	**73.1%**

The results on novel objects (objects that the algorithm has never encountered before during training) are as follows:

Table 5: Results on novel objects

Object Type	# instances	Prediction	Grasping
Cup	3	100.0%	100.0%
Misc	1	100.0%	100.0%
Measuring Tape	2	50.0%	0.0%
Rack	1	100.0%	0.0%
Pencil case	2	50.0%	0.0%
Foam	3	100.0%	100.0%
Shoe	5	100.0%	40.0%
Duster	2	100.0%	50.0%
Pump	1	100.0%	100.0%
Grand Total	**20**	**90.0%**	**61.1%**

Figure 10 shows some of the best predicted rectangles for a marker and a brush.

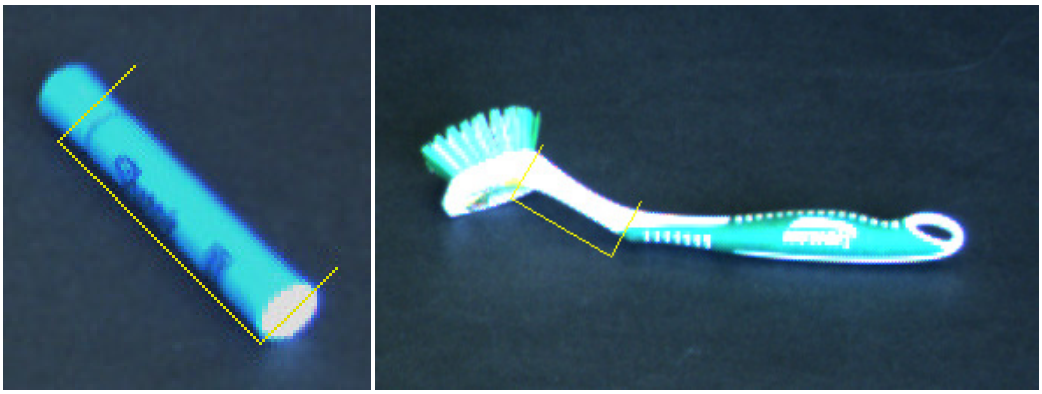

Figure 10: Test Results showing estimated grasping regions

We noted that the results from our robotic experiments were better than the results from our dataset. When we visually inspected our dataset, we realized that for many objects, the image tends to be overexposed, thus losing important image details. Hence, our algorithm had a hard time trying to identify good grasping rectangles on the dataset. During the experiments, we were careful to adjust the camera's exposure settings to ensure that the objects were correctly exposed, and this led to significantly better results.

During the grasping phase, for simplicity, we assume that letting our gripper approach the object along the vector parallel to the camera's viewpoint would be adequate. This assumption turned out to be inadequate for some cases where the object is very flat, and the best approach vector would thus be vertically downward. As a result of our assumption, our gripper failed to pick up many flat objects such a wooden spoon. To test that this is indeed

the reason for the failure, we placed these flat objects on a piece of foam which allows the gripper to push into the foam, yielding a greater inset into the object without hitting the ground plane. With the foam, the gripper had a markedly higher success rate, affirming our conjecture.

From the results on novel objects, our algorithm seems promising in identifying grasping rectangles for a wide variety of objects. We have shown that our algorithm, trained on a small set of objects from various classes, can produce results that generalize well to a wider range of objects.

7 Conclusion and future work

We have implemented an algorithm capable of identifying suitable grasping rectangles using stereo image and point cloud information. The combination of image and point cloud features allows our algorithm to perform significantly better than previous work and the results from robotic experiments show that our algorithm produces good results both on the trained objects and on novel objects that it has not encountered before.

We believe that our algorithm has more room for improvement. From table 2, we can see that non-linear features gave the greatest boost in our test accuracy, and hence, we could try to design more of such non-linear features. Another possibility is to use a non-linear kernel in SVM-rank, but this drastically increases the time needed to train the model as SVM-rank uses a specialized algorithm for training linear models.

We could also design simple heuristics to come up with better approach vectors for grasping objects in order to resolve the issue mentioned above. One way would be to use random sample consensus (RANSAC) [9] to find the ground plane in the point cloud, and use that to calculate the height of the object from the ground plane. If the height is low, it would be better to approach the object along the negative z-axis instead of the camera's viewpoint. We expect that implementing this simple heuristic would significantly increase our success rate for grasping.

References

[1] A. Saxena, J. Driemeyer, and A. Y. Ng, “Robotic grasping of novel objects using vision,” *IJRR*, 2008.

[2] T. Joachims, “Training Linear SVMs in Linear Time,” *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*, 2006.

[3] A. Bicchi and V. Kumar. "Robotic grasping and contact: a review," *ICRA*, 2000.

[4] M. T. Mason and J. K. Salisbury. "Manipulator grasping and pushing operation." In *Robot Hands and the Mechanics of Manipulation*. The MIT Press, Cambridge, MA, 1985.

[5] Y. Jiang, S. Moseson, and A. Saxena, “Learning to Grasp: What Should We Actually Learn?”. 2010.

[6] K. Loquin and O. Strauss, “Fuzzy Histograms and Density Estimation”, In *Advances in Soft Computing*, Springer Berlin, 2006.

[7] P. Kaewtrakulpong and R. Bowden. "An Improved Adaptive Background Mixture Model for Realtime Tracking with Shadow Detection," in Proc. 2nd European Workshop on

[8] R. Rasu, et al., “Fast Point Feature Histograms (FPFH) for 3D Registration”, *ICRA*, 2009.

Advanced Video-Based Surveillance Systems, 2001.

[9] M. A. Fischler and R. C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography." Commun. ACM 24, 6, 1981.