Identifying Grasping Points on Common Household Objects

Avtar Khalsa, Sajeev Krishnan

ABSTRACT

We worked on the problem of grasping objects based upon a single 2d representation of them as might be acquired by the camera on a robot. This was a challenge previously tackled by Professor Ashutosh Saxena while he was working at Stanford University. Our initial challenge was to get the code initially given to us to run properly on our systems. This was a significant challenge as the code was undocumented and heavily dependent upon the unknown file structure used in the previous implementation. Once we had resolved this problem, we attempted to improve accuracy by increasing the number of positive patches available to the training set. Another approach we used in an attempt to improve accuracy was to find a single weights vector for each type of object, and then find grasping points by first classifying the object and then selecting a weights vector.

INTRODUCTION

In the paper "Learning Grasp Strategies with Partial Shape Information" [1] Ashutosh Saxena along with several other researchers at Stanford were able to take various images of common items from around an office and kitchen and use machine learning to identify grasping points on the objects. These grasping points were locations for a robot arm to easily pick up the object. Unfortunately, the code as packaged did not run, and required extensive sifting through in order to be correctly executed. The basic structure of the code operated as follows:

1. Write a script to convert images into feature vectors
2. Take a subset of image feature vectors to use as training data.
3. In order to train, the images need to be mapped to the corresponding manually identified grasping points. These feature vectors need to be placed in correctly formatted arrays and then passed as inputs to the glmfit function
4. The output of the glmfit function is a 52x1 weights vector. This vector is used as the input of the matlab function glmval, along with the feature vector of the image to be tested. The output of this function contains the grasping point information which can then be formatted as appropriate.

Once these steps have all been accomplished, the grasping determined from glmval can then be compared to manually marked grasping points given to us in the training data. This allows us to determine the accuracy of our measurements. Once we got this far, it was important to find ways to improve the algorithm. We approached this from two angles.

One was to attempt to use more images in the training set. In our initial approach, we were limited to using 150 images in the training set. This is because the training set feature vectors need to be combined into a single extremely long array, and eventually run into the continuous memory restrictions of the computer the code is being run on.

We could increase the number of images by decreasing the size of each image used in

training. Since the images used in our analysis are mostly blank space surrounding the object in question, we could start by locating the object, then selecting a 400x400 box surrounding the object, and then use this for the training arrays. This would increase the number of images the algorithm could train with, and as such, increase the number of grasping points in the training set.

The other approach we took to improve the accuracy of the algorithm was to train it once for each object type. This yields a weights vector for each object type. When looking for grasping points on an object, we would first classify it as a specific object type and then use the associated weights vector in the glmval function to locate the grasping points. This would allow us to use 150 training images for each object type and could provide for the highest possible accuracy.

APPROACH

Initially we took 140 images, 20 of each object classification, and established them as our training set. We took the training set and organized them into their own folders. Each image was divided into patches for analysis. The 480x640 images were 67 patches wide and 54 patches long. The images typically consisted of a single object in a single color with a gray background. A few examples of original images can be seen on the left side of Figure 1 below. The goal of the matlab code was to determine which patches represent grasping points and which did not. For each image we also had a marked image which was a binary depiction of grasping points vs not grasping points in the corresponding original image. This corresponding grasping image was considered to be the ground truth in our analysis. A few examples of original images and their corresponding marked images can be seen in Figure 2 below.

To prepare the data for the glmfit function, we took the original images, and first passed them through the makeDepthFeatureVector function provided by Professor Saxena. The output of this function was a 3d array 67x54x51. For every single patch, there were 51 features. For every original image from the training set, we would generate this 3d array, and then we stacked them on top of each other in 2 dimensions. One row for every single patch from every single image, and 51 columns for the 51 features associated with each.

Some examples of the feature depth vectors can be seen in Figure 1 on the next page.

Ultimately, when all the feature vectors were combined, they formed a 2D matrix like the one seen in Figure 3 below.

We also took the corresponding grasping images, and simply stacked each patch one on top of the other with either a 1 or a 0 representing whether or not it was a grasping point.

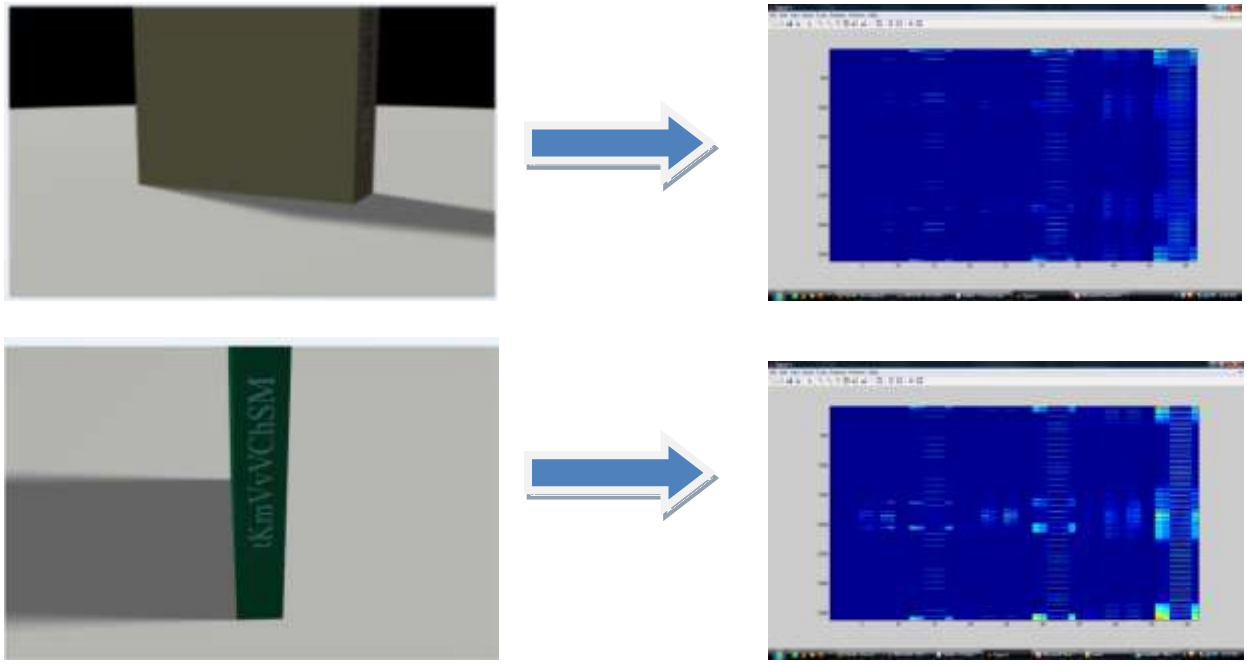


Figure 1. Original Images converted to Feature Vectors

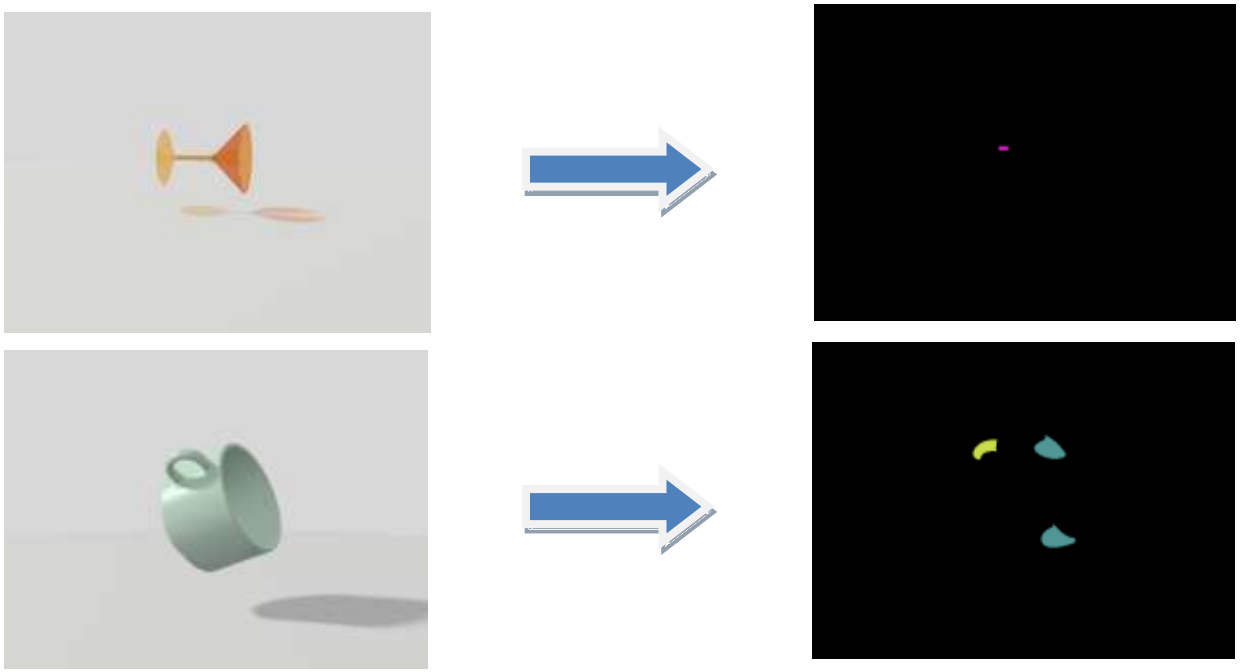


Figure 2. Original Images and their Grasping Points

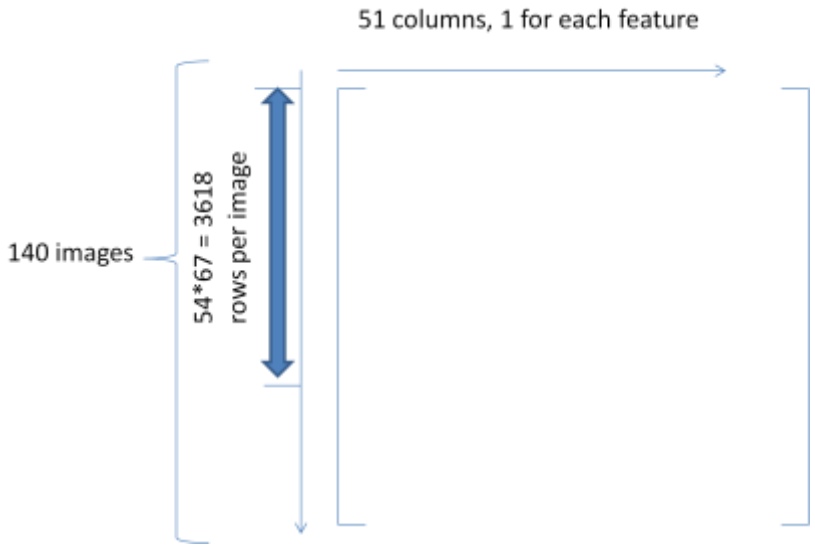


Figure 3. 2D Matrix representing all feature vectors for training images

EXPERIMENTS

With these 140 images setup, we called the glmfit function and were able to get a weights vector. With this vector we called the glmval function first for each of the images we used for training. We then took the top value coming returned by glmval and assumed that this was the grasping point selected. The results were adequate although not spectacular as can be seen in Figure 4 and 5 below. Figure 4 is an example of the algorithm working well. On the other hand, Figure 5, is an example of the image algorithm working quiet poorly.

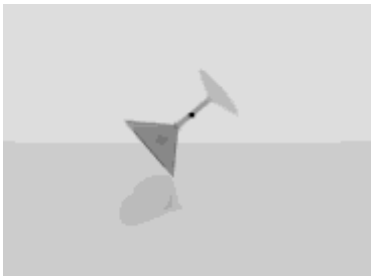


Figure 4

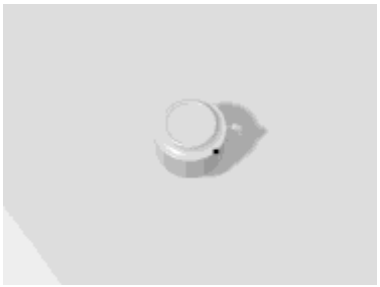


Figure 5

In order to get a reasonable grasp of how accurate our algorithm really was, we ran it across all the training images we had at our disposal. We then took the patch given the highest value, and checked to see if it was actually a grasping point from our ground truth grasping points. The results of this approach yielded only 10% of the top level grasping points as being actual grasping points. This is obviously an extremely low accuracy rate, and as such, we set out to improve it.

IMPROVEMENTS

The improvements we made were two fold.

Shrinking Images: we tried was shrinking the images from 640 x 480 to 400 x 400. This allowed us to decrease the number of patches in the image from 67x54 to 45x42 while maintaining roughly the same patch size. The benefit of shrinking the number of patches was that it allowed us to increase the number of training images without running out of memory.

In order to shrink these images, we used the grasping point images to locate the object, with matlab's regionprops function. Once we had located the center of the grasping points, we selected the 400 pixels surrounding it in both the original image and the grasping image. This created a new set of images, and corresponding grasping points all of which were smaller. These smaller images allowed us to go from a training set of 140 images to 240 images.

We expected to see a corresponding increase in the accuracy of the algorithm based on the increase in the number of training images.

With this increase, we did see a corresponding improvement in the number of correctly identified grasping points, as it went from roughly 10% to roughly 12%. An example of an improved grasping point can be seen in figures 6 A and B below.

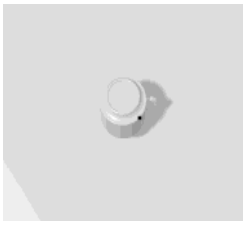


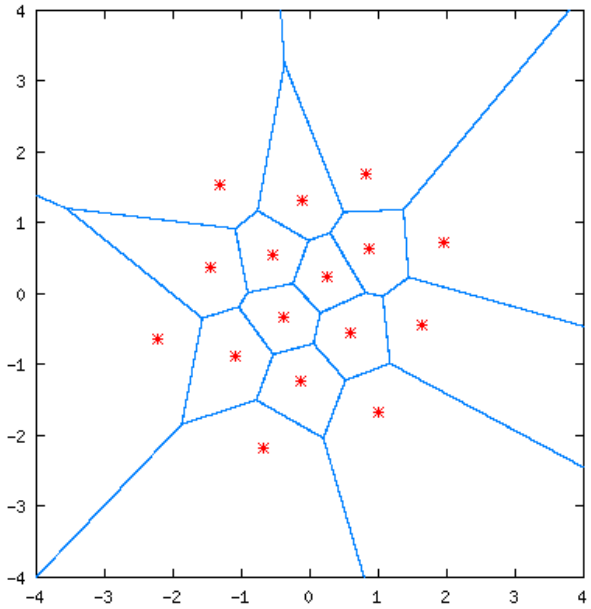
Figure 6 A

Figure 6 B

Unfortunately, this improvement was not very significant and as such, we tried a different approach for improvement.

Classification: A Vector Quantizer (VQ) is essentially an approximator, somewhat similar in nature to "rounding off" to the nearest digit. A simple example could be a number line, the set of numbers between 0 and +2 is approximated by 1 (the centroid), the set of numbers between -2 and 0 is approximated by -1, every number greater than +2 is approximated by +3, every number lesser than -2 is approximated by -3 and so on.

This notion can be extended to 2 dimensions by using centroids for defined regions.

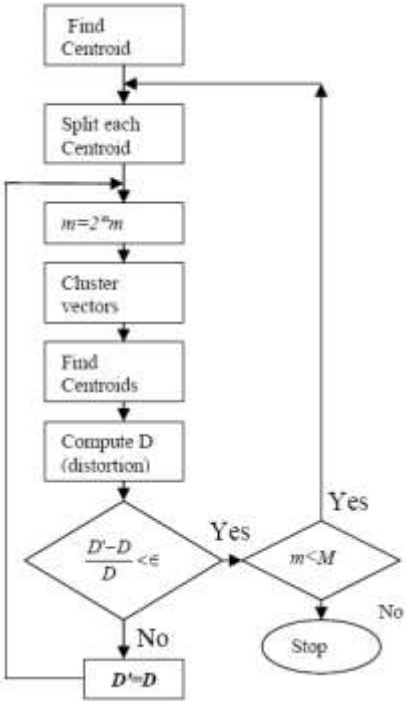


In the above example the red stars are the codevectors (or centroids) and the set of all the codevectors is known as a codebook. The complexity in the design of a VQ increases with the increase in the number of dimensions.

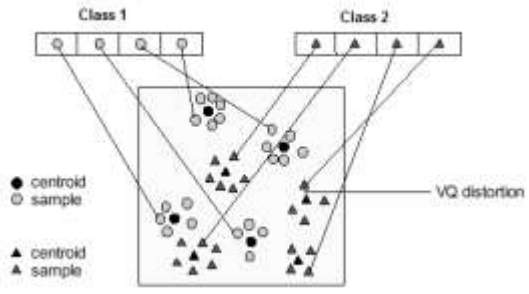
In 1980, Linde-Buzo-Gray proposed a VQ design algorithm, known as VQ-LBG, based on a training sequence. The initial codevector is obtained by finding the average of the whole cluster; two codevectors are obtained from the initial codevector by splitting the whole cluster into two regions. The iterative algorithm uses these two codevectors as the initial codevector to compute more codevectors, and till we get a codebook with the desired number of codevectors.

The algorithm is as follows:

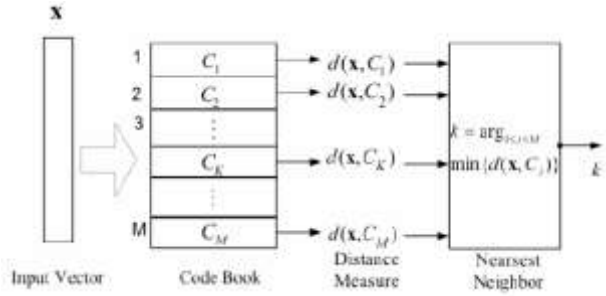
1. Design a 1-vector codebook; this is the centroid of the entire set of training vectors (hence, no iteration is required here).
2. Double the size of the codebook by splitting each current codebook yn according to the rule: where n varies from 1 to the current size of the codebook, and e is the splitting parameter. For our system, e = 0.001.(1)
3. Nearest-Neighbor Search: for each training vector, find the centroid in the current codebook that is closest (in terms of similarity measurement), and assign that vector to the corresponding cell (associated with the closest centroid). This is done using the K-means iterative algorithm.
4. Centroid Update: update the centroid in each cell using the centroid of the training vectors assigned to that cell.
5. Iteration 1: repeat steps 3 and 4 until the average distance falls below a preset threshold
6. Iteration 2: repeat steps 2, 3, and 4 until a codebook of size M is reached.



In the training phase, a class-specific VQ codebook is generated for each known class by clustering its training feature vectors. The resultant codewords (centroids) are shown in Figure 4 by circles and triangles at the centers of the corresponding blocks for speaker1 and 2, respectively. The distance from a vector to the closest codeword of a codebook is called a VQ distortion. In the recognition phase, an input utterance of an unknown voice is "vector-quantized" using each trained codebook and the *total VQ distortion* is computed. The speaker corresponding to the VQ codebook with the smallest total distortion is identified.



In the recognition phase the features of an unknown image are extracted and represented by a sequence of feature vectors {x1… xn}. Each feature vector in the sequence X is compared with all the stored codewords in codebook, and the codeword with the minimum distance from the feature vectors is selected as proposed command For each codebook a distance measure is computed, and the command with the lowest distance is chosen.



Using this methodology, we attempted to improve our accuracy by first classifying the image as one of the images we have used for training. Once the image is classified, we can then use a weights vector corresponding exclusively to that image type. This allows us to increase from roughly 20 images per object type in the original version all the way up to 140 images per object type in the training.

RESULTS

Our initial results at first looked quite appealing as on almost all the images, we were able to obtain at least one correct grasping point on almost all of the images, as can be seen in the attached excel file. Unfortunately, this algorithm almost never found only correct grasping points, nor did it frequently even find all correct grasping points. As such, we only considered the single grasping point that the algorithm considered most likely to be a grasping point. Unfortunately, this turned out to correspond to a correct grasping point only 10.61% of the time.

We adjusted the algorithm to only examine the 400x400 section of the centered on the grasping point. We expected the significantly greater number of training images to yield better results for the grasping points. Unfortunately, this modification only resulted in an accurate top grasping point 12.92% of the time.

Once we moved to the classifier based approach however, we were able to obtain significantly better results.

The classifier was tested with a set of 50 images and success rate was roughly 93%. However, we noticed some interesting behaviour; it classified objects by immediate appearance. Example: A martini glass when looked at from the top looks like a cerealbowl, and must be grasped at the rim like a cerealbowl. Speaking in absolute terms this would be a misclassification. But for the purposes of our grasping algorithm we would need the martini glass (top view) to be classified as a cerealbowl. Hence, we could venture as far as saying that the classifier has 100% success rate.

Once the items had been correctly classified, we could use the appropriate weights vector to determine the grasping point. With this approach, we were able to develop a weights vector using a training set of 140 images for EACH item, and as such expected it to be much more accurate.

Approach	Accuracy(%)
Original	10.61%
Shrunken	12.92%
Classifier	35.59%
Martini	14.29%
Mug	23.76%
teaTwo	33.00%
CerealBowl	71.29%

FUTURE WORK

Other approaches to improving the algorithm were to use segmentation to extract global features from the image. We ran into issues with this primarily because of two reasons. The region-growing algorithm takes a long time to run in matlab, making it infeasible for practical application. Also, the existing feature vector could not be augmented with the new features because of disagreeing dimensions.

While we were clearly able to demonstrate a significant improvement over the existing algorithm, we suspect the inaccuracies arise due to a more fundamental problem, that being extracting the feature vectors. Since the same feature vectors were used to classify the images, we are sure that they work reasonably for purposes of classification. But the classification would still work so long as they yielded the similar results for each class of object. This does not prove the validity of the feature vectors beyond that we are just calling the function correctly.

More techniques could be examined to include more global features of the object as opposed to just local features as this current program does. We also propose some extensive testing with the current feature vectors to determine their validity.

CONCLUSION

Our algorithm succeeds at finding the right grasping points and is certainly an improvement over the existing algorithm. However, the performance of the program is limited by one major factor; inability to use more images for the training phase, due to the space constraints imposed by Matlab. Our first improvement, i.e. shrinking input images to 400x400 so as to eliminate a significant number of negative patches allows us to use 240 images in training as opposed to 140 originally. Our second improvement allows us to make training more specific, i.e. use a training set based on the class of the object, which allows using 240 images of the same class against 240 generic images originally. There would definitely be an increase in performance if there were a way to include more images in training. Looking for solutions in this regard could be a potential source of work in the future. Also, some work could be done with image segmentation to isolate different objects from a given image. Currently the algorithm exploits only local features such as planarity, depth, symmetry and center of mass. This would allow us to incorporate some more global features of the object itself in the training phase and also eliminate a lot of false positives in the testing phase.

ACKNOWLEDGEMENTS

We sincerely thank Prof Saxena for his assistance, support and guidance and also for giving us confidence when we needed it most.

We would also like to thank the members of the personal robots group for giving us valuable timely suggestions, without which the project would not have been successful.

REFERENCES

Robotic Grasping of Novel Objects, Ashutosh Saxena, Justin Driemeyer, Justin Kearns, Andrew Y. Ng. In ***NIPS*** 19, 2006.

Learning to Grasp Novel Objects using Vision, Ashutosh Saxena, Justin Driemeyer, Justin Kearns, Chioma Osondu, Andrew Y. Ng, *ISER*, 2006.