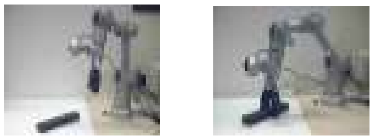
Learning To Pick Up a Novel Object



Chioma Osondu and Justin Kearns

1. Abstract

In order for an autonomous robot to successfully function in a natural environment it needs to be able to able to interact with objects it has never seen before. Specifically it needs to be able to manipulate objects in the environment that it has no prior knowledge of. The robot must not only be able to cope with uncertainty in the shape of the object, but also with uncertainty in the position and orientation of the object. This problem is one that can be approached with machine learning techniques. Our project is to teach a robot arm to pick up items it has never seen before. We will attempt this problem mainly with a supervised learning approach. We will train the arm by showing it a variety of objects. It will use monocular vision to extract features of the object and we will show it how to pick up the object up. Given this training set, it will learn the best way to pick up objects it has never seen before.
Other work in this area includes using uncalibrated stereovision to pick up an uncertain object at Cambridge University

2. Learning

This was done using linear regression.

2.1 Data Collection

We started off with images of 5 objects. In this case, markers of different colors and slightly different shapes. For each marker, we had 5 different orientations. For each orientation, we had 10 pictures taken from different positions, as the robot arm approached the marker. This process yielded a total of 250 images for training purposes. In addition to collecting images, the robot joint angles, operational coordinates and motor encoder values, corresponding to each image, were saved to files (one file per image.) Each image was 320 x 240 pixels in size (i.e. along the x-axis and y-axis.)

2.2 Initial Feature Selection

Our initial challenge in getting the robot arm to pick up unknown objects was to determine what features to use. Since as mentioned earlier, we had images and the corresponding robot position when the image was taken as well as the final position of the robot in the gripping position, we needed to determine the relationship between these images and the final orientation of the end-effector when the object was gripped.

The first attempt at feature extraction was done through a feature vector program written by Ashutosh Saxena. The program divides an image into patches of equal sizes and returns 17 attributes for each patch attained by using various filters (the number of patches could be modified to suit different training needs.) After careful inspection, we found that the optimal way of dividing up the image was to have 18 divisions along the x-axis and 14 along the y-axis. This guarantees that at least one edge of the object would be captured by one of the 9 patches, which we would be used in the next phase of the computation.

We chose to use the 8 patches surrounding the center of the object, as well as the center patch in a tic-tac-toe board configuration. This gave the total of 9 patches mentioned above. Determining the center of each object was done in following way; first, the image is converted to grayscale and then threshold values are used to separate the background (which was white) from the object. Second, given the points above the threshold, the largest of the clusters found in the image is noted. Finally, the midpoint in the x and y direction are used to determine the image center.

2.2.1 Training

For all of the learning described below, we used linear regression. We also added a bias column of all 1's to the training matrix in each of the attempts described below. For some of the tests, we introduced a regularization parameter, lambda with values ranging from .01 to .005. This did not significantly improve the results. In fact, most of the time, the results were identical to the results observed with the normal linear regression in closed form, so we reverted back to linear regression without regularization. 200 out of the 250 images were used for training and the remainder was used as the test set. The chi-squared error was used to analyze effectiveness/accuracy of our model; so all the errors reported below are chi-squared error values with formula (Y is actual orientation):

sum((Y - predictedY)^2) / sum((Y - mean(Y))^2)

2.2.1.1 Training (Take 1)

The 17 attributes corresponding to each of the 9 patches were used as a row in the training matrix. In the target matrix, each group of 9 rows, corresponding to 9 patches of the same image, would have identical values. In addition to these, each of the 10 images that were captured while moving to the final position was also associated with 1 outcome, i.e. the final orientation of the end-effector. This meant that 90 rows in the target vector had identical values. This proved to be a very bad approach to the problem as the both the training and test errors were very high.

2.2.1.2 Training (Take 2)

The training matrix was unchanged, but we decided to look at our target vector differently. Instead of having each image group of 10 images taken en route to the final position associated with the same value in the target vector, we used the difference between the gamma (the orientation of the wrist joint) at the time the image was taken and the final gamma. For testing, this gamma difference value tells us how much to increase or decrease gamma to match the object orientation. This approach was minimally better than the previous approach.

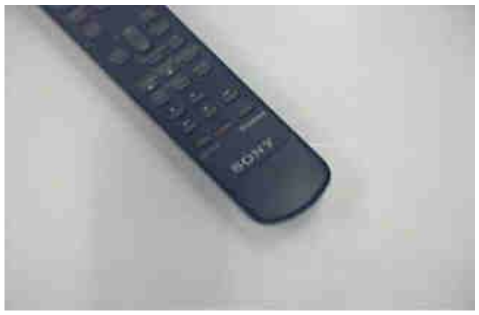
We came to the conclusion that drastic changes had to be made to our training model. Many of the 17 attributes were not informative with regard to our problem, so we decided to concentrate only on the 6 edge filters from the feature vector; ignoring the other attributes that deal with texture, et cetera. Also, having 9 rows corresponding to the same output value seemed inappropriate because it was unintuitive.

2.2.1.3 Training (Take 3)

The training matrix was set up in such way that the 6 edges corresponding to each of the 9 patches associated with one image were stored in 1 row. Therefore, there was 1 row per image and 54 attributes per row. This proved to be a better model, with significantly better train than test error. This also was a case of over-fitting. To improve the test error, we decreased the number of features, first by taking various combinations of patches, such as the 4 corners and center of our tic-tac-toe board as well as other combinations of more and less patches. As expected, this improved the test errors but had the adverse effect of increasing the training error.

2.2.1.4 Training (Take 4)

This approach, which seemed to be the winning approach, uses a single highly informative patch. The patch needed to have the most information about the orientation of the object. To select this patch we used different heuristics, for example, for each patch, we found the average of its edge scores and divided each edge score by the average. The patch containing the edge with the highest value was used as the best patch. The above heuristic used in isolation did not select the most informative patch.



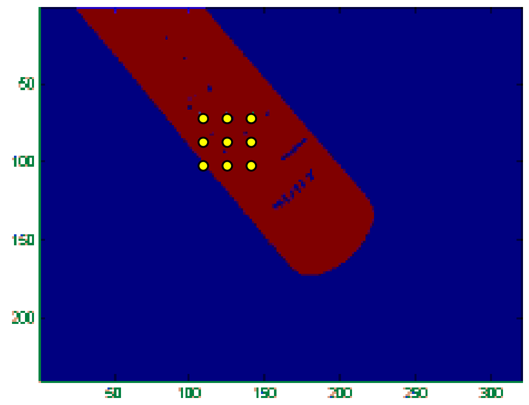


Figure 1. The object the robot is supposed to grasp is seen in the earlier picture. The estimated center of the object and the 9 patches around the center that are used in the heuristic computation below.

The heuristic that achieved the best results selected the best patch by combining the process described above with the following; the edge to average ratios computed above were summed along the edges for each of the 9 patches, i.e. the sum of all edge 1 ratios in the 9 patches and edge 2 ratios, etc. We considered the two greatest scores as the two best edges. Then for both of the edges, whichever edge was greatest in the most patches, we considered the dominant edge. The patch having the most difference between the dominant edge and the second best edge was then selected as the best patch. This heuristic was further reinforced by visual inspection.

Using this patch, we tried all possible subsets of the 6 edges and found that using 2 edges generated the lowest test error of 0.6926 with a training error of 0.7316. While the test error, was significantly better than before, it was still not good enough to use in practice. It was generally correct in regard to following the shape of the curve, but the magnitude was much smaller as can be seen in the graph below.

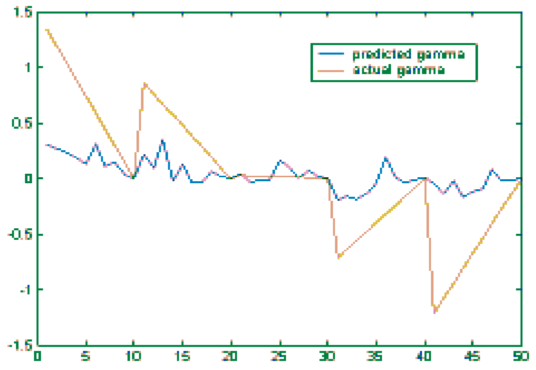


Figure 2. The actual end-effector orientation compared with the predicted end-effector position (using 2 edges as described above) on the test data.

2.3 Alternate Feature Selection

We decided to start from scratch on our own features. A lot of the collected images had bad shadows, making it difficult to determine the ends of the object from the threshold value. We used the saturation channel of HSV, which generally eliminated the background and returned only parts (non-white) of the object. We used the method described earlier to estimate the ends of the object based on the position of the largest cluster. With these points, we needed to determine the coordinates of the object ends. We knew an end of the object was either in the top left or bottom left, so we checked the region in the top left and if part of the object was there, the object was oriented top left to bottom right. If it was not, then it was oriented bottom left to top right.

The object orientation was determined by finding the angle between the vector pointing in the Y direction from the object center and the vector from the center of the object to the end point. This angle alone did not seem to be enough because the magnitude or the orientation value could be misleading. For example, in the case of a white pen with a red cap, the HSV would only pick up the red cap and the orientation angle would be large even if the cap angle was not, because of the small area.

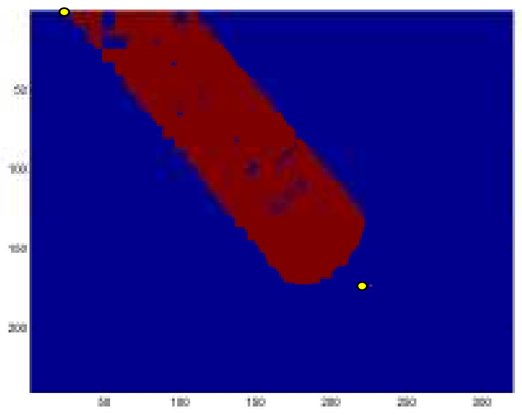


Figure 3. Saturation channel of object with estimated end points. The object used here is a remote control.

To fix the orientation magnitude problem, we added the area that we analyzed as a feature to enable it to learn when to give more and less weight to the magnitude of the value. It also seemed logical that the position of the camera when the picture was taken would affect the analysis, so the difference in the camera and object position as well as the pitch of the camera were included.

We tried all subsets of the 4 features and found the pitch did not help, so we narrowed it down to the 3 features: heuristic angle, camera to object distance, and area of the object analyzed. These 3 features generated the best test error, at that point in the project, of 0.372.

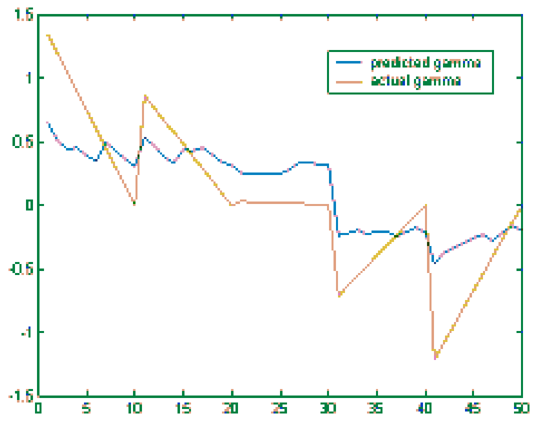


Figure 4. The actual end-effector orientation compared with the predicted end effector position using the features described above on the test data.

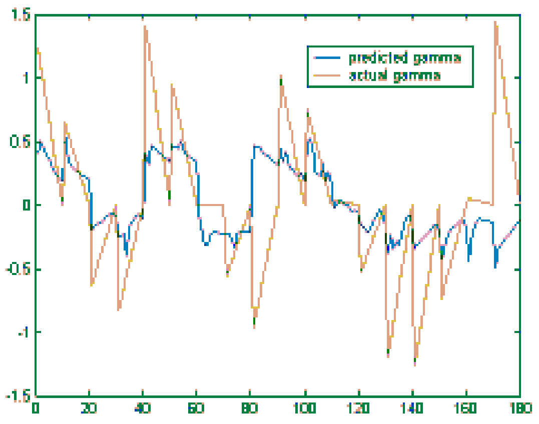


Figure 5. The actual end-effector orientation compared with the predicted end effector position using the features described above on the training data.

The test error was good compared to previous cases, but looking at the graph of actual output vs. predicted output, the magnitude and sign were better matches compared to the previous graph, but it did not fit the curve of the graph.

Since the previous features in the other training experiments fit the curve better, we combined the two sets of features, so that the number of features in the training matrix was 5. Therefore, the total number of columns in the training matrix was 6, including the bias column. The test error here was 0.29.
We tried training and testing on other parts of our 250 images using these features and found that our test error was between 0.5 and 0.6, which was worse than previously observed. This can be attributed to biasing our data toward the pens we trained on, so we mixed up our data so that we trained on all types of pens and tested on all types of pens, which resulted in test errors of about 0.54. However, the graph of predicted vs. actual was more promising, so we tried out the predicted values on the robot. The results of the experiments did not meet our standards.

2.3.1 Training (Final Take)

Since the best results described above were not much better than random data selection, we decided to collect more data. This time the selection of objects included a variety of long thin objects. The lighting was also better this time around as there was a light source directly above the robot, eliminating most of the shadow. The total number of images in the collection grew to 650. We ran similar tests and found this time, that it was more fitting to use 4 edges with the area, angle of orientation and distance between the camera and the object as features.

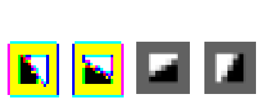


Figure 6. The 4 edge filters that produced the best test and training error.

Randomly selected rows of the entire feature matrix were used as training data and the remainder was used for testing. Of the 10 images for each object orientation, one image was randomly selected for test and the other 9 were used for training.

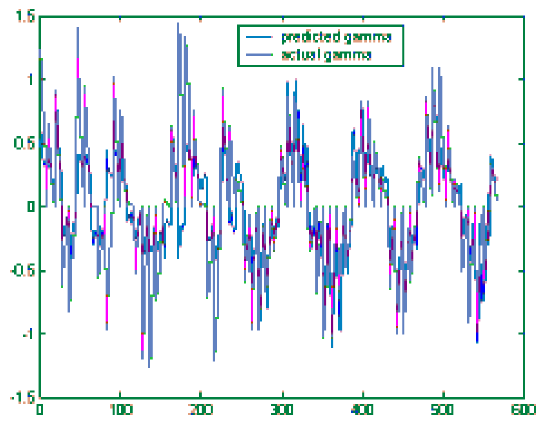


Figure 7. The actual end-effector orientation compared with the predicted end effector position using the edges from Figure 6 combined with the other three features mentioned above. This is on training data.

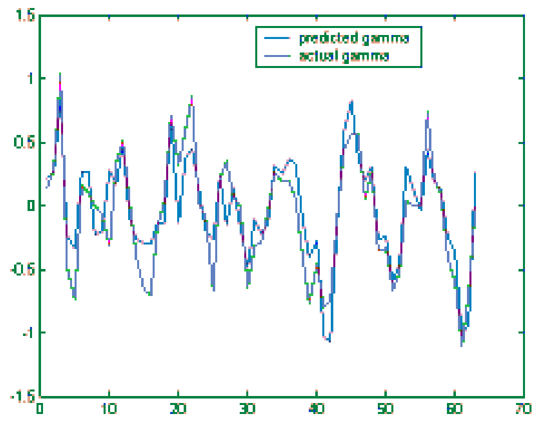


Figure 8. The actual end-effector orientation compared with the predicted end effector position using the edges from Figure 6 combined with the other three features mentioned above. This is on the test data.

As can be seen from the figures above, getting more data proved to be a worthwhile decision. The observed test error was 0.24 and the train error was

0.37, mostly from magnitude misses. While there are some magnitude misses, the robustness of the robot's gripper allows it to grasp objects with orientation values within range.

2.4 Testing

The robot takes a picture of an object and writes the image to one file, while the corresponding joint angles, operational coordinates, and motor encoder values are written to a separate file. This image is then analyzed and an end-effector orientation (which is equivalent to a single target value) is returned using the processes described above. Since all of the analysis and training was done in Matlab, to fully automate the robot motion, we needed to integrate the C++ code and the Matlab code. To this end, the Matlab code was compiled into a standalone executable. A call was then made from the C++ code to the Matlab executable after the files needed for analysis were generated. The Matlab executable would output a file containing the predicted gamma difference value (which would then be used to generate the final end-effector orientation) as well as the object center. The object center was needed to compute the values of some of the other robot operational coordinates. We use the camera intrinsic and fixed Z plane of the table to compute the objects location in the robot's coordinate system and move to that position with the learned gamma value. We make 3 assessments of the object location from different positions and choose the median. If the robot grips the object, it puts it in a box, otherwise it returns to the start position and tries again. Since the Z plane was fixed, the most important coordinate to facilitate the desired outcome was the orientation of the gripper. The gripper had to be turned the right way for the robot to successfully pick up the object in its view.

2.5 Experimental Results

We tested the robot with 5 objects in arbitrary orientations that were not included in its training set, 3 pens of various color and shape, 1 eraser, and 1 coiled USB cable. The robot correctly guessed the sign of the orientation on all attempts where the object was clearly oriented in 1 direction. When the object was oriented straight up, the robot guessed a small orientation in either direction. The match of the magnitude of the robot's orientation was not exact, but it was always in a range close enough for the robot to get the objects in its gripper on every attempt it made, with 1 exception on the coil of wires in a crescent orientation in which the robot arm was oriented correctly, but estimated the center poorly.

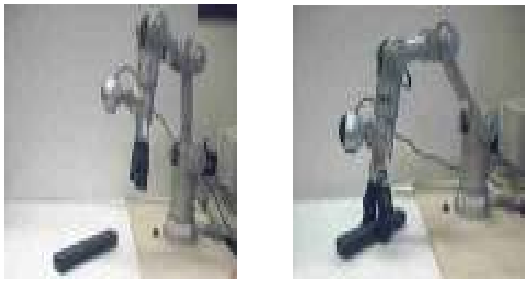


Figure 9. The harmonic arm in the start configuration, while analyzing the object (left). The harmonic arm in the final position, gripping the object with its end-effector, using the correct orientation to ensure that the object can be picked up.

3. Conclusions

Initially, it seemed highly unlikely that a linear classifier would be successful at this task. This was because. It was hard to imagine a linear relationship between the orientation of the end-effector and the features, which were used during the training. Our initial attempts were discouraging because even though we were able to drive the training error down, the test error only increased. After extensive analysis of our initial features and target variables, we were able to lower the test error, even though the values would not be good enough, in practice.
This led to the creation of an unrelated set of features as described earlier.
The major breakthrough here was combining the two sets of features giving us values for the orientation of the end-effector, which could actually be used with an amazing degree of success in practice.

Acknowledgements

Ashutosh Saxena was an invaluable resource and mentor throughout the project and we would like to say a special thanks to him. Also, many thanks to Jimmy Zhang and Anya Petrovskaya for help with the Harmonic Robot Arm.

References

1. Saxena, Ashutosh, Chung, Sung H., & Ng, Andrew (2005). Learning Depth from Single Monocular Images.