Dinner Table Object Segmentation, Classification and Grasping by the PR2 Robot with a Head Mounted Kinect RGB-D Camera

Abhishek Sharma*, Bharath P. Bhat*, Rohan Kamath*

Abstract—**While robots have been used in an industrial setting with calibrated environments for decades, robotic manipulation in home environments still poses a challenge. The unconstrained nature of the environments is usually the cause of concern. The PR2 by Willow Garage Inc. has gained significant popularity in the robotics community. In this project we attempt to use a Microsoft Kinect sensor mounted on top of the PR2 to observe table top objects. Further the algorithm attempts to recognize general dining table objects and locate their positions. Finally, we attempt to make the robot pick up the objects and assemble them in different locations based on their type.**

INTRODUCTION

The goal of this project set is to recognize dining table top objects, their positions in the world frame and subsequently cause the robot to pick up and move them to different boxes/areas based on the object type. Thus the PR2 needs toestimate an object's type, its position, choose an appropriate grasp point and move it to the appropriate area based on its category. In our current implementation, the scope of the object set is restricted to three categories, namely - cups, bowls and plates.

A. About the Robot – the PR2

The PR2 robot is designed to do human-scale tasks in environments where people are present. Use of ROS (robot operating system) released by Willow Garage Inc. was made for running the programs to control the robot actions. The PR2 coupled with ROS provides a customizable platform for researchers to prototype ideas. We make use of the platform to implement our machine vision algorithm and get a working prototype of a robot to clear a dinner table.

II. TRAINING DATA COLLECTION

A Microsoft Kinect was mounted on the head of the PR2 to collect both RGB and depth images of the dinner table. As the Kinect is a very popular extension device for the PR2, we were able to find and use the stable *openni* Kinect driver available on the ROS wiki [2]. The depth image is returned as a pointcloud containing x, y and z estimates for each point. Knowing that the Kinect depth sensor has a range of approximately 1.2 m - 3.5 m [3], we fixed the height of the robot and its distance from the table. Subsequently, we built our training set by placing objects at different positions and orientations (whenever applicable) on the table. We obtained 3600 RGB images and point clouds in total, 1200 belonging to each class. 4 different bowls, 8 different coffee mugs and a soda glass (both belonging to super-category 'cups') and 3 different plates were employed. A snapshot of the objects used for training can be found in Fig. 1.

Fig 1. : A subset of training images used for this work.
First row are representatives from the plates class, the second row from cups and third row represents bowls.

The Kinect generates images with a resolution of 640 X 480 pixels. The following preprocessing steps were carried out to make the data ready for learning purposes. All of the steps were carried out in MATLAB unless otherwise specified.

A. Depth Data Smoothing

The data obtained from the Kinect Depth camera is unable to estimate depth of all points correctly. These points are returned as NaN (not a number) by the *openni* driver. In order to use the data for our other algorithms we used a nearest neighbor interpolation technique for estimating the depth of the missing points. We then used a 4x4 median smoothing procedure to reduce the noise in the depth image. Fig 2. shows a raw depth image from the Kinect sensor and

* In alphabetical order of first names. Denotes equal contribution by the authors.

Abhishek Sharma is first year MS, Electrical Engineering. Bharath P. Bhat is first year, MS, Mechanical Engineering. Rohan Kamath is first year, MS Computer Science.

the result of the smoothing process. Only the smoothed data was used for all subsequent operations.

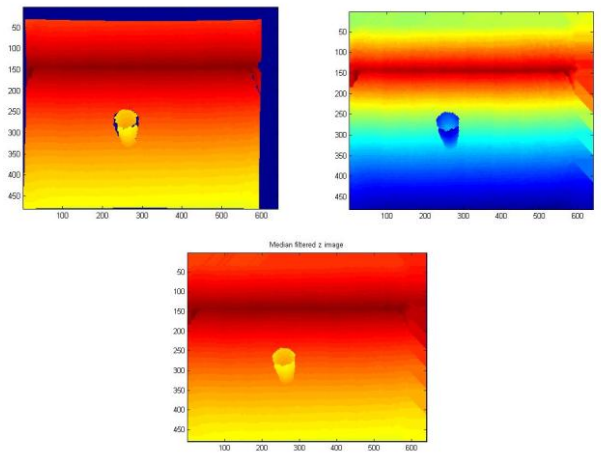

Fig 2 (Top to Bottom, Left to Right): First image shows a raw depth image obtained, second image is the result of the interpolation process, third image is after median smoothing. The visualization here is a scaling of the depth values.

B. Object Segmentation

In this section we explain the object segmentation techniques employed for segmenting the objects of interest in a given image frame. We used a sliding window approach to traverse the entire image and pick out windows which might contain objects. Our approach was based on the following two observations:
i) A window which contains an object is likely to have a good number of edge pixels
ii) We limit our windows of interest to those that completely enclose an object. This translates to the fact that our machine learning algorithm is trained to detect objects provided it sees the full object, and hence is not equipped to detect occluded objects. This simplification however makes it much easier to obtain a clean data set and a high accuracy on the object recognition front.

Our clustering algorithm was structured around the above two arguments to output object windows given the full field of view seen by the Kinect. More specifically, given an input image and associated point-cloud, the algorithm performs the following steps:

i) Obtain a Canny edge map [4] of the RGB image. Set threshold to 10% of the total number of edge pixels.

ii) A 120x120 window goes through the image, picking out windows that contain atleast as many edge pixels as the threshold calculated in step (i). This pre-filtering step greatly improves the speed of object segmentation.

iii) Carry out k-means clustering with k = 2 on each window selected in step (ii). The grayscale intensities and depth values are used as features for the k-means algorithm. This step differentiates the object from the background in the window. Pick out only those windows for which there exists a cluster that has less than 20 pixels along the boundary of the window. This step ensures that the object is fully enclosed by the window.

The process is repeated for a 200x200 window. The windows were selected based on the maximum size of our objects of interest. The smaller window helps to define a tight boundary around the smaller objects (bowls and cups). Fig. 3 shows the result of our segmentation algorithm on a full image.

Fig. 3 : Top image shows the input RGB image to the algorithm. The middle and bottom row images are grayscale images of the segmented objects along with the category they were assigned to by our learning algorithm

These segmented RGB and point cloud images enclosing the objects of interest were used to generate features for machine learning as discussed in the next section.

III. SHAPE CONTEXT FEATURES

Our decision to use shape context features was motivated by the fact that the shape of the object defined from the depth data was a clear differentiator for our object classes. Shape context features [5] provide a simple way of tracking the internal and external contours of an object that define its shape. In particular, the following steps were followed to generate the shape context features.

i) Obtain the canny edge map of the depth image. This extracts the internal and external contours of the object. Our

use of the depth data instead of the RGB data to define edges is motivated by the fact that the depth edge map is invariant to designs and textures on the object surface. (A very common occurrence for dinner table objects, and an argument against rich descriptors like SIFT [6]). Figure 4 shows representative canny edge maps for the three object classes.

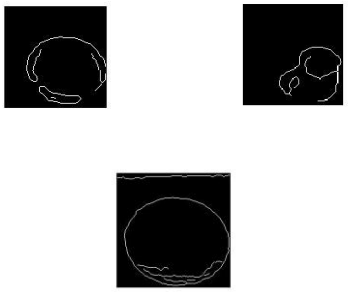

Fig. 4: Canny Edge map obtained on the depth data.
Top row shows edge map of a bowl and cup. Bottom row shows edge map of a plate.

ii) For each edge pixel, we divide the 10x10 pixel window around the pixel into 8 angular bins, and assign the current pixel to the bin that has the highest number of surrounding edge pixels. In other words, this roughly determines the direction of the edge around the current pixel. The process is repeated for every edge pixel, and hence, every pixel is assigned to an angular bin, and a histogram is constructed. This histogram forms an 8 dimensional feature vector that defines a given image. The feature vector is normalized so that the actual number of edge pixels does not have a bearing on the classification. Fig.5 shows representative histograms for the three object classes.

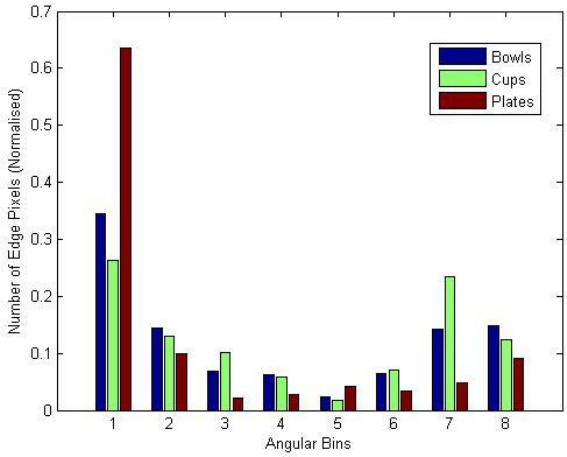

Fig.5.: Representative histograms for the three object classes.

The features so generated help us recognize objects irrespective of color and surface design/texture. Although it is not theoretically invariant to rotation, our training set contains several possible orientations of each object which should provide some robustness. The algorithm is scale invariant as long as the objects fit into the 200x200 window defined above, as it only extracts the relative number of edge pixels going in any direction.

These 8-dimensional feature vectors are used to train an SVM discussed in the next section.

IV. SVM Classification

We made use of the popular LibSVM [7] library for Matlab to implement a SVM model for object recognition. A multi-class SVM was used with a Radial Basis Function (RBF) kernel. The RBF kernel is given by the following equation :

$$K(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T \phi(x^{(j)}) = \exp(-\gamma \| x^{(i)} - x^{(j)} \|^2), \gamma > 0$$

and is equivalent to mapping the data to an infinite dimensional space [8]. For multiclass classification, LibSVM implements one against onemethod. n(n-1)/2 classifiers are employed that separate every pair of the n-labels.Thus we have 3 classifiers in our implementation. The final output class label is the class that wins the most number of binary decisions. LibSVM also provides a probability estimate P(y=i|x), i = 1,2,3 based on [9]. Our implementation uses these probability estimates to determine the best object window.

A. *Cross Validation and Grid Search*

The performance of SVM depends upon proper selection of the cost parameter C and the parameter γ of the RBF. We searched across a grid of possible values of C and γ to achieve the best possible results. Each model was compared based on the average accuracy obtained by carrying out k-fold cross validation with k = 10 on our dataset of 3600 images. Table 1 gives the results of the grid search. We were able to achieve a cross validation accuracy of 99.1% by setting C = 15 and γ = 500.
This model was then trained on the entire dataset and deployed on the PR2 for online implementation.

γ/C	1	100	50	20	10	15
0.125	84.1%	87.3%	-	-	-	85.2%
1	86.4%	90.0%	-	-	-	88.9%
10	90.1%	93.0%	-	-	-	91.7%
100	93.4%	98.1%	-	-	-	96.9%
1000	98.3%	98.8%	98.8%	98.8%	98.9%	98.8%
2000	98.1%	98.4%	98.4%	98.4%	98.4%	98.4%
600	98.1%	98.9%	98.9%	99.0%	98.9%	99.1%
400	97.0%	98.9%	98.9%	99.0%	98.8%	98.9%
500	97.6%	99.0%	99.0%	99.1%	98.9%	99.1%
900	98.3%	98.8%	98.8%	98.9%	98.9%	98.9%

Table 1 : Results of cross validation accuracy for different values for C and γ.

V. Implementation on the PR2

The implementation on the PR2 involved three steps:

i) Using the trained SVM model to recognize objects in a new scene

ii) Determining optimum grasp points for the PR2 to pick up a recognized object

iii) Defining trajectories for the PR2 to move to the grasp point and place it at its assigned location.

A. Online Object Classification

As and when the robot sees a new image, a scanning window approach similar to that described in Section II picks out possible object windows. These smaller windows are then fed into the SVM model for recognition. In case there are multiple windows that identify the same object with slight offsets, the algorithm selects the window with the highest confidence based on the probability estimates produced by LibSVM. Two windows of sizes 120x120 and 200x200 traverse the image. The choice of window sizes and the segmentation algorithm presented earlier provide some compensation for the lack of a noise class as they are likely to reject larger objects or objects that do not have a significant number of edge pixels on the depth edge map.

B. Grasping Point Determination

In order to determine the optimal grasp points for each object, we used the grasping library developed by Ellen Klingbeil et al. [10]. The algorithm is motivated by the observation that when a robot has a solid grasp on an object, the surface area of contact is maximized. Thus, the shape of the region being grabbed should be as similar as possible to the shape of the gripper interior. The input is the pointcloud for the segmented object. The output is the set of coordinate frames of the best grasp points. Fig. 6 shows the output grasp points on a bowl and a snapshot of the robot executing a grasp. The grasp points need to be converted from the frame of the head-mounted Kinect sensor to the frame of the robot.

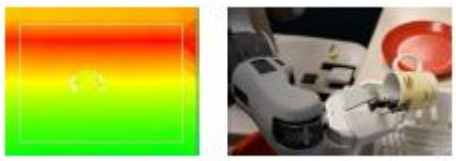

Fig. 6: L: Grasp points plotted on the point cloud, R: Gripper in action.

C. Trajectory Definition and Execution

After obtaining the grasp points we have used pre-defined ROS libraries [11] to define smooth trajectories for the arm through inverse kinematics. We call a ROS service that takes in desired Cartesian poses for the PR2 gripper (as given by the grasping library) and converts them to arm joint angles. We define a suitable time interval to execute a trajectory in order to ensure smooth motion and avoid jerks caused by inertia. The ROS node called Joint Trajectory Action then executes the trajectory so that different objects are placed in different trays.

A URL of the implementation video is at [1].

VI. Conclusion

We have implemented simple machine learning techniques to teach the PR2 to recognize dinner table objects. We were able to achieve highly accurate classification in our current implementation with 3 object classes using depth data from the Kinect. Further focus would be on making our algorithms more robust to overcome the limitations mentioned earlier, and to extend the range of objects considered.

VII. ACKNOWLEDGEMENT

The authors would like to thank Prof. Oussama Khatib and PhD student members from his group Ms. Ellen Klingbeil and Mr. Samir Menon for their guidance and support during the project. We are also thankful for the access to the PR2 robot.

References

[1] http://www.youtube.com/watch?v=5tJJFD8_XX8
[2] http://www.ros.org/wiki/openni_camera.
[3] http://jira.ai2.upv.es/confluence/download/attachments/12222467/Kinect+System.pdf?version=1&modificationDate=1301668437000
[4] Canny, John; , "A Computational Approach to Edge Detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on* , vol.PAMI-8, no.6, pp.679-698, Nov. 1986.
[5] Belongie, S.; Malik, J.; , "Matching with shape contexts," *Content-based Access of Image and Video Libraries, 2000. Proceedings. IEEE Workshop on* , vol., no., pp.20-26, 2000
[6] Lowe, D.G.; , "Object recognition from local scale-invariant features," Computer Vision, 1999. *The Proceedings of the Seventh IEEE International Conference on* , vol.2, no., pp.1150-1157 vol.2, 1999
[7] Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1--27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm
[8] http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=MachineLearning&doc=exercises/ex8/ex8.html
[9] T.-F. Wu, C.-J. Lin, and R. C. Weng. Probability Estimates for Multi-class Classification by Pairwise Coupling. *Journal of Machine Learning Research*, 2004..
[10] Klingbeil, Ellen; Rao, Deepak; Carpenter, Blake; Ganapathi, Varun; Ng, Andrew Y.; Khatib, Oussama; , "Grasping with application to an autonomous checkout robot," *Robotics and Automation (ICRA), 2011 IEEE International Conference on* , vol., no., pp.2837-2844, 9-13 May 2011
[11] http://www.ros.org/wiki/pr2_controllers/Tutorials/Moving%20the%20arm%20through%20a%20Cartesian%20pose%20trajectory#The_Joint_Trajectory_Controller