3D Object Detection using Depth Images

Christol Barnes, Pradeep Kumar, Sachin Mohla
University of California, Santa Barbara
{christol, sachinmohla, pkg}@umail.ucsb.edu

Abstract

Object detection is a classic computer vision problem and a challenging one. Through this project we explore object detection of 3D objects using depth images obtained using Kinect. We trained our object classifier by rendering 3D CAD models of regular household objects with different poses and orientations. For each model obtained we extracted VFH and GRSD feature descriptors. Our classifiers include a k-D tree model based nearest neighbor search and an SVM. For testing we performed segmentation and clustering on the captured indoor scene to obtain clusters that are similar to the trained data set. Using chi-square metric we were able to classify the objects and detect their pose.

1. Introduction

Object detection is a fundamental problem in computer vision. Being able to automatically identify objects in a given scene has lots of important applications, mainly in robotics, image retrieval or video classification among others. Tremendous progress has been made in the field of recognizing objects from a two-dimensional image (2D). However, with the advent of cheap 3D depth sensors such as Microsoft Kinect, Asus Xtion Pro and Intel Real-Sense, a new area of research has opened up which uses 3D information in areas where 2D doesn't perform too well. A perfect example for this is real time human pose recognition which is used in Microsoft Xbox.

The goal of our project is to develop a general machine learning framework based solution for detecting objects using Kinect RGB-D images. Literature shows that RGB-D based detection methods can outperform state of the art techniques like DPM and R-CNN which use RGB images due to several reasons, especially the illumination invariance of depth map images and their occlusion tolerance.

1.1. Kinect Sensor

The kinect camera is a sensor module consisting of one RGB camera and depth sensor which comprises IR Laser and a CMOS sensor. For our application we made use of the depth sensor alone, to obtain a depth map of the target object. Since it uses infrared light, it is robust to ambient lighting unlike the RGB cameras.

2. Existing Work

As part of the literature survey for 3D object detection using depth images we explored relevant papers and topics that motivated this project. Most of the recent work on 3D object detection using RGB-D images can be broadly classified into Image based detection and Semantic segmentation techniques.

2.1. Image based detection

The object detection in 3D is primarily addressed as a problem of localizing objects of certain classes. In case of image based detection most popular techniques train a classifier on an image area within a window and test using a classifier via a sliding window [1], [2], [3] or on selected areas [4], [5]. In most cases minimum bounding rectangles (MBRs) of the objects are the ideal outputs. The simplest approach here is to use a sliding window of varying size and classify sub-images defined by the window. Usually, neighboring windows have similar features, so each object is likely to be detected by several windows. Since multiple/wrong detections are not desirable, non-maximum suppression (NMS) is used. As for the classification task powerful machine learning concepts such as convolutional neural networks have been applied to object detection [6]. For object recognition problem the different classes of objects to be determined are typically trained using a linear SVM but in the case of object detection in 3D, Shuran Song et al. show that applying an ensemble of exemplar SVMs gives high accuracy in detection. For object detection, HOG (histogram of oriented gradients) templates based sliding window classifiers have also been shown to be robust and widely used [7], [8].

2.2. Semantic Segmentation

Most prior work on RGB-D perception have focussed on semantic segmentation i.e. the task of assigning a semantic label for each region of a depth map or a 3D mesh [9].

Because of the segmentation task the inherent issue with this technique is that the entire object in a scene is not captured. Feature descriptors are then developed based only on the part that is captured. Notably, the notion of an object instance is missing from such an output. However, [6] proposes a CNN model for object detection based on semantic segmentation as proposed in [10] and has shown to give state of the art results.

We now focus on our base paper. The paper by Shuran Song et al. [14] proposes a 3D detection window in 3D space for object detection. As part of their training model, a collection of 3D CAD models are rendered from hundreds of viewpoints at different scales to generate synthetic depth maps. From the 3D point cloud obtained, features such as point density, 3D shape, 3D normal and TSDF are extracted. These features help capture geometric shape and orientation of 3D objects which makes the detection more robust. With the features extracted an ensemble of linear Exemplar-SVM classifiers is learnt. Each SVM is trained with a rendered depth map from CG model as the single positive and many negatives from a labeled dataset of RGB-D maps. This model builds on the work by Tomasz Malisiewicz, et al. [2] that proposed the Exemplar-SVMs for object detection in RGB images and proved that the method improved the accuracy.

For the testing stage the paper [14] proposes a sliding 3D detection window to detect the object in space. The 3D space captured by the bounding box is exhaustively classified using all Exemplar-SVMs. Based on the detection score of each SVM it is evaluated whether the corresponding shape of the object to be detected exists or not. Finally non-maximum suppression is performed on all the detection boxes in 3D. This method claims to outperform the state-of-the-art algorithms for RGBD images, and achieves about $1.7\times$ improvement on average precision compared to DPM and R-CNN.

The focus of most papers on object detection has shifted towards detecting 3D objects using RGB-D images. Also with commercial hardware products such as kinect becoming more accessible to users, many new ideas have been developed in this field. We see that a lot of the recent work focused on rich feature extraction and emphasized on developing local and global descriptors in 3D that improve classification of objects. The object detection is now treated as a classic machine learning problem to improve the detection accuracy. CNNs and deep neural nets have been proved to be useful for standard RGB tasks like image classification and object detection. Many recent papers develop on the same idea to exploit these powerful techniques on RGB-D images that improve accuracy and provide robust 3D object detection [6].

3. Features

In order to match two different point clouds, we need to find a set of correspondence among them. The comparison needs to be unambiguous and reproducible. Some of the most regularly used features are normal, point density and shape descriptors. But these are not optimal enough. For a feature to be optima, it needs to be robust to transformations, noise and be resolution invariant. Therefore, we make use of feature descriptors, which are more complex and precise signatures of a point, and hence help in easy matching between a given set of points. There are various kinds of feature descriptors, as described in [11], with most of them being histograms describing a particular aspect of the object point cloud. Some of them encode the difference between the angle of normals between a point and others, while some others use distance between the two points. Due to this, the descriptor can be scale invariant and may be able to tolerate some amount of occlusions. The choice of the descriptor depends on the accuracy and computational requirements of a given application. Descriptors can be classified into two categories: local and global.

3.1. Local Descriptors

Local descriptors are computed for individual points or key points of a given point cloud. They just specify the local geometry around a given point. Hence, these descriptors are most ideal for multi-object or cluttered scenes where its difficult to obtain a single unique description of the scene using a global descriptor. We now look at one such descriptor which models the local surface geometry for each point.

3.1.1 Point Feature Histogram

This feature first identifies all pairs in the vicinity of a point and for each pair computes a fixed coordinate frame using their normal. Using this frame, the difference between the two points in terms of their normal is encoded with three angular variables. The calculation of the frame is done using the following vectors which are computed using the normals defined at the two points:

$$u = n_s \tag{1}$$

$$v = n_s \times (p_t - p_s) \tag{2}$$

$$w = u \times v \tag{3}$$

Once these vectors are obtained, the three angles α, θ and ϕ are calculated using the following equations:

$$\alpha = cos^{-1}(v.n_t) \tag{4}$$

$$\phi = cos^{-1}\left(u.\frac{(p_t - p_s)}{\|p_t - p_s\|}\right) \tag{5}$$

$$\theta = tan^{-1}\left(w.n_t, w.n_t\right) \tag{6}$$

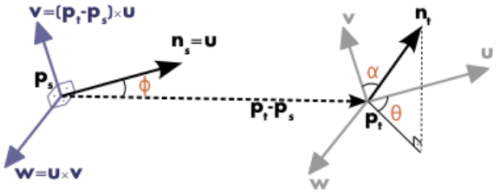

Figure 1. PFH vectors and angles

Figure 1 describes the computation of these three angles for two points p_s and p_t.

Once these angles are obtained, they are then binned into five equal bins. Since there are three dimensions (α, θ and ϕ), and five bins, the total dimensionality of this feature comes up to $5^3 = 125$. The original proposal for this algorithm [12] also considered using distance between the points as an additional parameter to encode, but, it was dropped from the final implementation since it was not considered discriminative enough. PFH provides highly accurate representation of the object point cloud, however it has a drawback: it is computationally too expensive to perform. With a cloud of n key points with k neighbors, the descriptor has a complexity of $O(nk^2)$. Hence, its not suitable for real time applications. Therefore, we decided to use global descriptors which even though provide less accurate representation of a point cloud, are computationally inexpensive.

3.2. Global Descriptors

As the name indicates, these are computed for a whole cluster instead of evaluating them at each point. Therefore, they tend to encode object geometry. Global descriptors are mostly used for classification, geometric analysis in terms of an objects type, shape and for pose estimation. We used two such global descriptors in our object recognition pipeline: viewpoint feature histogram and geometric radius-based surface descriptor.

3.2.1 Viewpoint feature histogram

This extends the idea of point feature histogram to global scale. Since, PFH is invariant to the objects pose, the viewpoint is added as an extra information to this descriptor. Therefore, we have two parts to this descriptor: a viewpoint direction component and an extended PFH component. The difference between the original PFH and the implementation of this descriptor lies in the fact that, here the PFH is calculated for a single point: the centroid.

Once the centroid is found out by averaging the x, y and z coordinates of the object, the vector between the position of the RGB-D sensor (viewpoint) and the centroid is computed. Following this, for all points in the cluster, the angle between this vector and this centroid is calculated and the result is binned into a histogram. This descriptor is scale invariant since the vector between viewpoint and centroid is translated to each point when computing the angle.

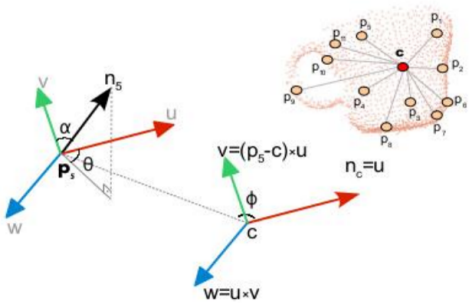

Figure 2. VFH vectors and angles

The second component is computed in the same fashion as done in case of PFH. Here, the three angles are computed only for the centroid with computed viewpoint direction vector as its normal and setting all points belonging to the cluster as neighbors. The method described here is illustrated in Figure 2.

Finally, we concatenate the four components (1 for viewpoint, and 3 for PFH component) and normalize the bins to build the final descriptor. Apart from these, the current implementation of VFH also includes a shape distribution component (SDC) which increases the size of the descriptor to 308. This shape distribution component encodes the information about the distribution of points around the regions centroid in terms of the distances. This helps in differentiating the objects with similar characteristics such as two planar surfaces from each other.

The PFH vectors and the angles for the centroid are computed as before using the following equations:

$$u = n_s \tag{7}$$

$$v = \frac{(p_i - p_c)}{\|p_i - p_c\|} \times u \tag{8}$$

$$w = u \times v \tag{9}$$

Using these vectors, the angles are calculated in a similar fashion.

$$\alpha = cos^{-1}(v.n) \tag{10}$$

$$\phi = cos^{-1}\left(u.\frac{(p_i - p_s)}{\|p_i - p_s\|}\right) \tag{11}$$

$$\theta = tan^{-1}\left(w.n, u.n\right) \tag{12}$$

The shape distribution component is calculated as follows:

$$SDC = \frac{(p_c - p_i)^2}{max[(p_c - p_i)^2]} \tag{13}$$

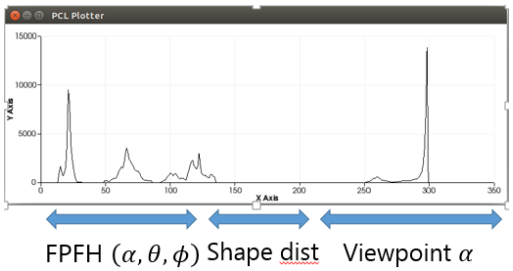

Figure 3. VFH vectors and angles

The final histogram obtained is shown in Figure 3

3.2.2 Global Radius Based Surface Descriptor

This descriptor is described in a work by Marton et al in [13] and is an extension of a local descriptor called radius based surface descriptor (RSD) which uses voxelization of space and simple feature estimation. It is based on the idea that, for a set of voxels which are labeled, a new global feature can be computed using the geometric and appearance relationships between the local labels. It basically counts the number of transitions between different types of voxels. The computation of this feature involves two steps: voxel annotation and object categorization.

For voxel annotation, we first construct a neighborhood based on the points belonging to a specific voxel and its surrounding voxels. Once this neighborhood is computed, the radius-based surface descriptor (RSD) which involves computation of their principal curves radii rmin and rmax, is computed in the local neighborhood as described in [13]. The RSD is basically computed using the distribution of normal angles using distances. After finding the RSD, we categorize the surfaces into

- Planes: large r_{min},
- Cylinders: medium r_{min} and large r_{max},
- Edges: small r_{min} and r_{max},
- Rims: small r_{min} and medium r_{max},
- Spheres: similar r_{min} and r_{max},

After annotation of voxels, a global feature space is computed that can produce a unique signature for each object cluster obtained after segmentation of a scene. The relationship between each of these local labels are computed and the individual histograms are summed to obtain final histogram. We can then categorize local household objects as plate, bowl, cylinder, box depending on their histogram characteristics. Figure 4 and Figure 5 illustrates the different histograms obtained for a flat box and for a coffee mug.

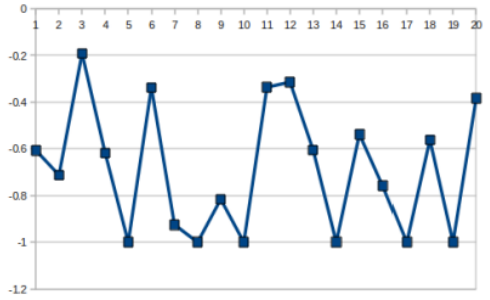

Figure 4. GRSD histogram for coffee mug

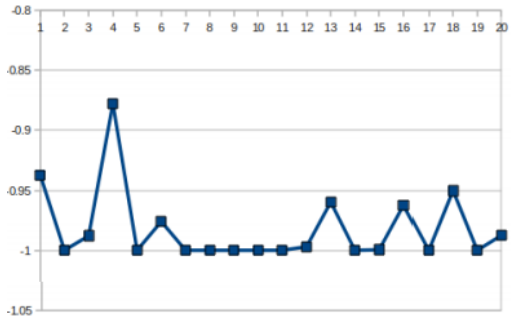

Figure 5. GRSD histogram for flat box

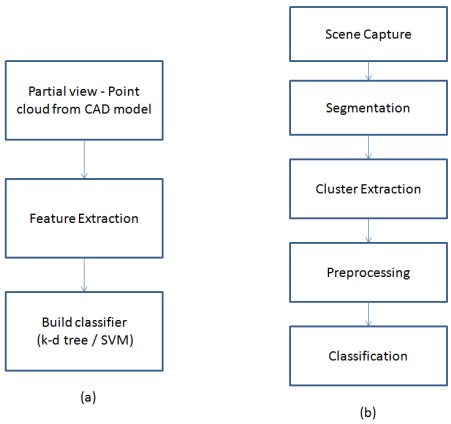

Figure 6. Object Recognition Pipeline: (a) Training (b) Testing

4. Object Recognition Pipeline

Our object recognition pipeline is described in Figure 6.

4.1. Dataset and Libraries used

Our dataset consists of 3D CAD models obtained from various sources online. We converted most of the CAD models to .PLY format because we use PCL (Point Cloud Library) for handling point clouds and PCL library supports the .PLY format. By integrating PCL with VTK (Visualization Tool Kit) library we can convert CAD models to point clouds. For implementing k-D trees we used the FLANN approach described in [15] and LibSVM library for Support

Vector Machine implementation. Our dataset consists of the following class of objects : chair, bowl, box, ball, bottle and mug.

4.2. Training

The scene that we analyze to detect objects consist of only a single view. So it is imperative that the training models that we generate are partial views of the CAD models, rather than using the point cloud of the full model.

We performed preprocessing using Blender and Meshlab softwares to edit the CAD models to suit the requirements of VTK, which is the Visualization toolkit associated with PCL. The processed CAD models are in the form of mesh and have to be converted to point clouds. We used a typical ray tracing technique in PCL, where the camera is fixed at a position somewhere around the CAD model. A view direction and Up vector of camera defines the view and the combination of all these with the camera position is called the view point. A view frustum is constructed in along the view direction starting from the camera position. The view point construction is made such that object position is somewhere in the view frustum. Now several rays are rendered with uniform spacing throughout the frustum starting from camera position and the points where these rays hits the mesh of the CAD model becomes the point cloud of the CAD model from that particular viewpoint.

We used 80 such viewpoints for our experiments. This gives us the set of partial view point cloud for each CAD model. Most of the camera viewpoints are set to focus on the centroid of the original point cloud and some account for translated views as well.

Once we have the point clouds extracted from the CAD models, we extract the VFH and GRSD feature descriptors from each of these models.

4.2.1 Building the classifier

We used two approaches for building the classifier: k-D trees for nearest neighbor search and SVM based classification.

K-d trees are one of the most efficient algorithms used for nearest neighbor search. A k-d tree is similar to a binary search tree but in k-dimensions. In our case, k-d tree, which is a space partitioning data structure organizes features in a k-dimensional space. The construction of k-d tree is such that every non-leaf node implicitly generates a hyperplane that splits the hyperspace into two parts, which are called as half-spaces. During every node traversal, the direction of hyperplane is chosen such that the hyperplane is set by the value of one dimension of the feature and the unit vector in that dimension is set as the normal of the hyperplane. The vector that is chosen for every level of the tree in a random manner and it is made sure that it covers

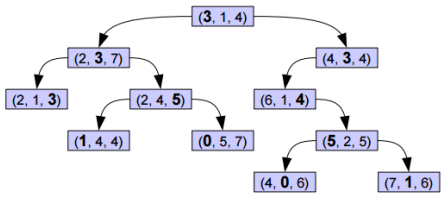

Figure 7. k-d tree example

the entire k-dimensions. Similar to binary search tree, the left sub tree has nodes that has lesser value than the parent node in that particular dimension of the feature vector and the right subtree has a greater value than the parent node in the same dimension on the feature vector. The k-d trees are very efficient in low dimensional space but the performance decreases exponentially for high dimensional space. So we used the FLANN [15] library for approximate nearest neighbor search in higher dimensional space. It contains different implementations for k-d tree which is optimized based on the type of the data set used. The one we were interested is the priority search k-means tree.

For SVM implementation, we used LibSVM library for performing SVM classification with polynomial kernels. Since it is a multi class classification, we used one-vs-all approach where we trained all views of the object as positive examples. This is different from the exemplar SVM based method used in [14] where each of the different viewpoints of the same object is modeled as different classes. Even though the method of using exemplar svm can provide very good result in terms of identifying the pose directly from the svm result, the time complexity increases exponentially as the number of viewpoint considered increases. Instead we tried to use all viewpoints as same class for SVM based classification.

4.3. Testing

4.3.1 Scene Capture

We used Kinect hardware along with openni library to capture the scene. The point cloud is then generated from the scene using the depth image. This method is similar to the way we obtained partial view from CAD models but with more noise.

4.3.2 Segmentation

For segmentation we used the model based approach implemented as SAC segmentation class in the PCL library. SAC in this case is similar to the RANSAC algorithm (Random Sample Consensus), a non-deterministic algorithm that given a data with included outliers returns the parameters of a model which explains the data and implicitly points out the outliers. The SAC class implementation implements

several generic shapes such as plane, cone, cylinder etc. This allows us to fit a mathematical model (shape in this case) to the points in a point cloud. The points that fit this model are the inliers while the rest of the points can be termed as the outliers.

For our purpose of 3D object detection over a given surface such as ground or table, we fit a plane model to the point cloud. Interestingly the points that are captured as inliers fit the plane model that we described while the outliers are all the remaining points that we are actually interested in segmenting out.

4.3.3 Convex Hull

In the next stage, in order to extract out points of interest we first construct a convex hull. A convex hull for a set of points in Euclidian space is the minimum convex set containing all those points. PCL again has a convex hull class implementation that allows us to determine these specific set of points on the plane model that were segmented out in the previous step.

4.3.4 Polygon Prism

Once the convex hull is obtained, we can now focus on the points in point cloud that are part of this region. To obtain all relevant points we define a polygon prism which captures them. Important parameter to define here is the height of the polygon (object dependent). By setting an appropriate minimum and maximum height of the polygon only relevant points of the objects to be detected can be extruded out while the points on the plane and other irrelevant parts of the point cloud can be excluded. We make use of the Extract polygon prism data class of the PCL library to achieve this.

4.3.5 Euclidian Clustering

From the polygon prism step we obtained the points which belong to different objects that are part of the scene captured. To be able to segment out objects as individual clusters we make use of the euclidean clustering extraction class, implemented as part of the PCL library. Clustering is the process of examining a collection of points in the point cloud and grouping them into clusters based on a distance measure. The goal is to segment out different clusters based on the fact that points in same cluster have a small distance from one another while points in different clusters are at a large distance from one another. Euclidian clustering hence achieves this by defining the euclidian distance in space as the distance measure. Using this technique we are able to segment out different objects defined in space, however the distance tolerance that we determine is critical. A low distance tolerance is more suitable for scenes that include multiple objects within close vicinity, example - objects on a table. A high tolerance is more suitable when detecting large objects such as chair or sofa in an indoor scene given the inherent geometric nature of the object to be detected.

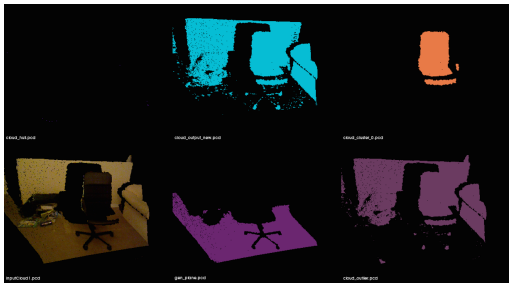

Figure 8. Object Recognition Pipeline

The results of the segmentation flow is shown in Figure 8. The figure in the bottom left corner is the input point cloud of a scene containing an office chair. The image on the top right is the final segmented cluster obtained.

4.3.6 Classification

The k-d tree or SVM classifier that we generated in training stage is used to detect the class of the object. In case of k-d tree, we stored the features of different poses of same object in the k-d tree. Thus using k-d tree helps in finding the orientation along with the specific class of the object. In case of SVM, we only detect the class but not the pose. Our implementation had the flexibility to construct SVM for class labeling and k-d tree for finding the pose of the object. This combination of classifiers that we call the hierarchical classifier relies heavily on SVM results. This will be successful only with a larger training dataset. From our experiments with a smaller training set, it will be unfair to comment on the efficiency of this hierarchical classifier. So we used only k-d trees as it gives the best results for smaller data set.

5. Online Training

We also implemented online training where we captured objects using Kinect and extracted features in a way similar to that of the testing pipeline, till the feature extraction part. Once the features are extracted, building the classifier is similar to that of the CAD model based training.

6. Results

We first used SVM for detecting chair as explained in training section 4.2 and were able to classify chairs almost every time with this classifier. However, this was not in real time.

Hence we settled with only k-d tree as the classifier and obtained the best match results in around 1 second on average for detection. Again, this depends on the size of the k-d

tree and we used around 25 models belonging to 6 classes with each model rendered in 80 different views. It is quite difficult to find an exact CAD model for house hold items especially for complex objects like chairs, sofa. In case of smaller and lesser complex items like mug, cereal bowl the CAD models matched our test objects better than the complex objects and for items like ball the CAD models was almost same as the ones we used for testing.

6.1. Evaluation metric

Comparison is done using the histograms obtained after feature extraction. We use chi-square metric to compare these histograms. The chi-square metric for comparing two different histograms H_1 and H_2 is defined in equation 14.

$$d(H_1, H_2) = \frac{\sum_n (H_1(n) - H_2(n))^2}{H_1(n)} \qquad (14)$$

The Table 6.1 shows the results that we obtained for the class models we considered. The classification accuracy shows how accurately our classifier can label the object. In our experiment, there are different types of objects in the same class. k-D tree based nearest neighbor search returns the closest match against the training dataset we have. In case of average Chi-square distance, the lower the value, better is the match. This means, the partial view and the CAD model object that defines the feature vector that the k-d tree returned as the best nearest neighbor, is matching the view and the test object. Also, the chi - square mentioned in table 6.1 is calculated only from the positive results of object classification. If the detected object is wrong, then it makes no sense to check the chi-square value. Chi-square distance of the best nearest neighbor CAD model returned by k-d tree is averaged over different poses of various items belonging to same class.

We can note that even though the ball is very simple to detect since it is the same from every pose, the noise in the depth image captured sometimes results in it getting wrongly classified as bowl. So, adding noise to our CAD models can improve the Chi-square metric and make the pose detection from k-d tree more accurate. But the problem with adding noise is that the noise amplitude is highly object variant. The noise level for a chair should be high because it is captured from a farther distance, when compared to a cup, which can be detected only if it relatively closer to the camera. Therefore, we didn't make use of noise addition in our experiments.

Also, in case of box we used online training as mentioned in section 5 where we captured only four different poses for building the k-d tree, it still gave good results. This is due to the fact that the training model matches the test model, except for pose. So, the accuracy is high but the chi-square distance metric is worse, because there were very less number of poses to compare with.

Class	Classification Accuracy	Distance
Ball	93.33	3673
Coffee Mug	70.67	6321
Bowl	68.67	9134
Box	80	12185
Bottle	53.33	14435
Chair	41.33	34425

Table 1. Experimental results

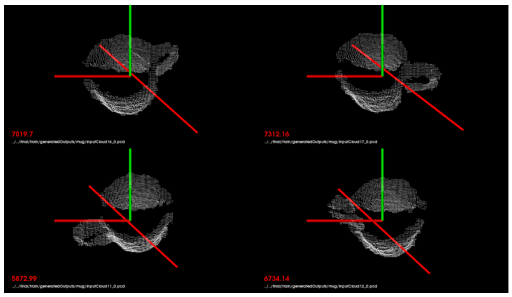

Figure 9. Matching obtained for a coffee mug

7. Conclusion and Future Work

We were successfully able to classify the test objects and detect their pose in real time. However, this work can be improved using hypothesis verification. Also, using local descriptors, we can improve the estimated pose and this can be used to improve the understanding of the scene captured. For the learning stage, exemplar SVMs can provide better classification results but this will incur a penalty in terms of time complexity.

8. Acknowledgements

We would like to thank Dr. Victor Fragoso for providing the necessary hardware and guidance during this project.

References

[1] Felzenszwalb, P.F., Girshick, R.B., McAllester, D., Ramanan, D.: Object detection with discriminatively trained part based models. PAMI (2010)

[2] Malisiewicz, T., Gupta, A., Efros, A.A.: Ensemble of exemplar-svms for object detection and beyond. In: ICCV. (2011)

[3] Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: CVPR. (2005)

[4] Wang, X., Yang, M., Zhu, S., Lin, Y.: Regionlets for generic object detection. In: ICCV. (2013)

[5] Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: CVPR. (2014)

[6] S. Gupta, R. Girshick, P. Arbelaez, and J. Malik. Learning rich features from RGB-D images for object detection and segmentation. In ECCV, 2014.

[7] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In Proc. of CVPR, 2005.

[8] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained partbased models. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2010

[9] Ren, X., Bo, L., Fox, D.: Rgb-(d) scene labeling: Features and algorithms. In: CVPR. (2012)

[10] Banica, D., Sminchisescu, C.: CPMC-3D-O2P: Semantic segmentation of RGB-D images using CPMC and second order pooling. CoRR abs/1312.7715 (2013)

[11] L.A. Alexandre, 3D descriptors for object and category recognition: a comparative evaluation, in: Workshop on Color-Depth Camera Fusion in Robotics at the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vilamoura, Portugal.

[12] Rusu , Z.C. Marton, N. Blodow, and M. Beetz, Persistent Point Feature Histograms for 3D Point Clouds, In Proc. of the 10th Int. Conf. on Intelligent Autonomous Systems (IAS-10), 2008.

[13] Z.-C. Marton, D. Pangercic, N. Blodow, J. Kleinehellefort, and M. Beetz, General 3D modelling of novel objects from a single view, in Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, Taiwan, 2010.

[14] S. Song and J.Xiao, Sliding Shapes for 3D Object Detection in Depth Images, Proc. of 13th European Conf. on Comp Vision, ECCV2014.

[15] M. Muja and D. G. Lowe, Scalable nearest neighbor algorithms for high dimensional data, IEEE Trans. Pattern Anal. Mach. Intell., vol. 36, no. 11, pp. 22272240, Nov. 2014