CS 4758/6758: Object Detection and Tracking

Rudhir Gupta

***Abstract*— Detecting an object in an image or a video is one of the foremost work before proceeding to understand the semantics of the scene. Tracking such target objects over sequence of frames provide us relevant information about where the object is. In this project, we are developing algorithms to help make the robot detect household/food items in the scenes captured through Microsoft Kinect Sensor. Also, we would be using the object tracking algorithms to carve out a reliable object classifier. The algorithms use rich set of features from both 2D and 3D frames.**

I. INTRODUCTION

Object detection and tracking is the first step in understanding the environment around the Robot. It is useful in detecting the activities and categorizing them based on the object it manipulates and interacts with. By tracking objects simultaneously we can know the association of objects with each other. For example, moving of hand with the milk container provides information that both objects were used for same activity 'pouring milk' and that they work with each other.

In 2D images, the two most used algorithm for detecting things are Haar Classifier [3] and the HoG [2] features based algorithm. But they have mostly been successful on very small subset of objects specifically in Face detection and pedestrian detection.

We used both 2D and 3D features of the object to classify target objects. In 2D, we used both HoG features and color histograms in our learning algorithm and used Viewpoint Feature Histogram from 3D point cloud data. We used Support Vector Machine for learning from 2D frames and k-Nearest Neighbour approach to learn 3D data.

For tracking the objects, we used Discrete Kalman Filter [4] with some heuristically learned parameters. Although linear, it was able to track the objects hovering with the human hand with good accuracy but was bounded by the false detections.

We ran the experiments on five objects, namely Can, Banana, Pear, Mug and Cereal Box with different type of environments, cluttered and uncluttered. For food type items we got an accuracy of around 70% and for household items it was around 55% average precision. Tracking these objects was nearly as accurate as the detection algorithms (reasons discussed in Section 2.3).

II. APPROACH

We now outline our approach. Our input is Kinect point cloud data of the scene. The scenes are typically table-top scenes but it could be any other scene too. We then, segment out the point cloud clusters of each object in the scene. These segmented clusters are the atomic units in our model. Our goal is to find the most probable cluster of the target category.

Next, we use the output from the detection module as the observations to our tracking module. The tracker uses the old observations and the current one to predict the least erroneous pose (state) of the object.

2.1 Object Segmentation

Segmentation is the step to get the probable clusters containing the target object. In 3D point cloud data, segmentation is done using the RANSAC algorithm to segment out the plane which could be table-top, floors and walls.
After getting the non-planar points from segmentation, clustering is done to get the probable clusters. We experimented with two different algorithms:

Euclidean Clustering: The algorithm uses nearest neighbor approach to divide an unorganized non-planar point cloud to smaller parts. This approach is more prone to over-segmentation.

Region-Growing[5]: This approach uses smoothness constraint for clustering. It uses normals from the surface of the point cloud as the method to align points to different clusters. This approach is more prone to under-segmentation.

Fig.1 Segmentation

2.2 Object Detection

Detection of the object from frames of the video is the most critical step for the tracking and thus scene understanding. We used both 2D and 3D features to capture the inherent local and global properties.

The properties we aim to capture through our model are:

Visual Appearance. Visual properties like color, texture and gradients of intensities provide us good local property of the object.

Shape and Geometry. The shape provides the signature of the object. The normals from the object surface in 3D data provide us such information. This also captures the pose/alignment of the object with respect to the viewpoint.

2.2.1 Features

We used Hue-Saturation color histograms and Histogram of oriented gradients as our primary features from the 2D data. These features are the local features and provide the intrinsic properties of each target object.

In 3D, we used the Viewpoint Features Histogram (VFH) features which captures the global orientation of the normals from the target's surface. This acts as the signature of the object.

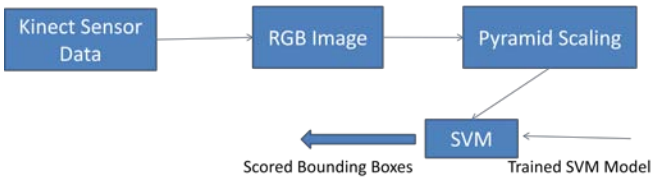

Fig.2 2D Classification

2.2.2 Learning Algorithms

Support Vector Machine[6]: We used the SVM implementation for classifying the 2D images based on features (which are 2D features here). We used SVM as it can work in higher dimension too and is a fast prediction algorithm as it produces a very short model (Support vectors) for the training data.
We used svm-light[1] implementation and used linear kernel function in a regression model.

The weight vector of the hyper-plane in svm is captured by summing the support vectors (whose lagrange multipliers (α) are greater than 0) using the following equation:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$
$$where, x_i = ith\ support\ vector$$
$$y_i = ith\ label$$
$$\alpha_i = lagrange\ multiplier\ for\ ith\ SV$$

Since, we were using the binary classifier, the regression values can be taken as the scores assigned to the detections of each cluster. Also, we used the **Platt's Algorithm** based on the LibSVM[2] implementation to get the probability scores for each detection by learning the parameters A and B for the MLE model using the logistic regression with 4-fold cross validation.

$$P(i|i\ or\ j, x) = {}^{1}/_{1 + e^{Af+B}}$$
$$where, f = decision\ value\ or\ label$$
$$x = test\ or\ train\ sample$$
$$i, j = class$$

k-Nearest Neighbor: The VFH features from 3D frames were fed into the k-NN algorithm. The algorithm focuses on finding the most probable class based on the closest training examples. The distance between target cluster and the trained object is taken as score and binned. We used Flann library with k = 6.
For example, if the distance between apple cluster and trained orange is 445, then binning it with 50 width, gives a category of 9.0 but a score of 1/0.9, equaling 0.1.

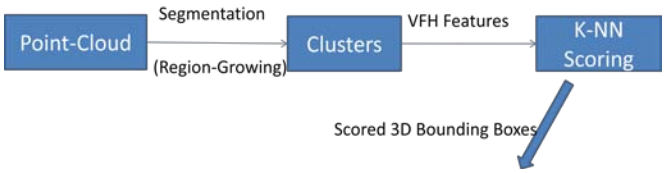

Fig.3 3D Classification

2.2.3 Training

2D Image Training: The 2D classifier is trained on both positive and negative images. The positive images are of some specific size which differs for each class of object like 64 x 64 pixels for apple and 64 x 96 for cereal box. The trained model from SVM is generated which can be used for testing test data. We train on the positive images of same window size. The negative samples can be of arbitrary size and thus can generate multiple negative samples by scanning the given image at different positions.

3D Image Training: The 3D classifier needs the VFH features of the images, thus the training is done on specific target objects. All the trained features are fed into kNN algorithm.

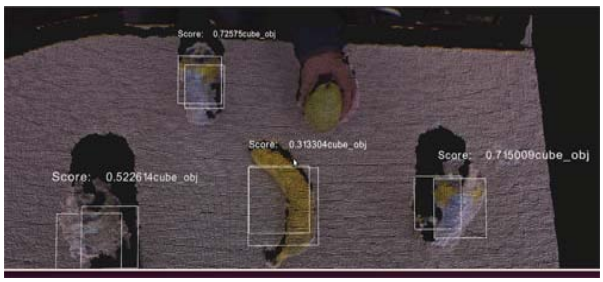

Fig 1.a: All three cans are found

[1] http://svmlight.joachims.org
[2] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

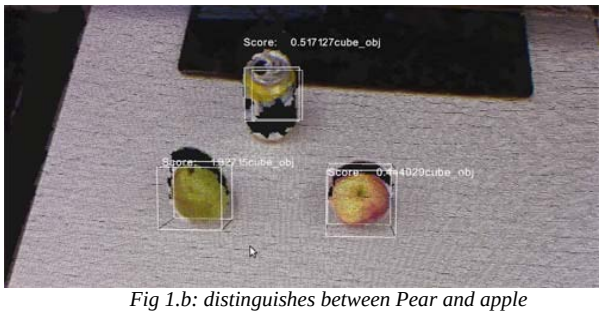

Fig 1.b: distinguishes between Pear and apple

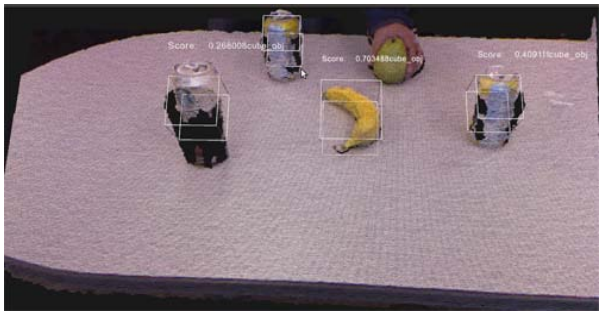

Fig 1.c: finds banana as most probable

2.2.4 Testing

Testing is similar to the Training procedure. In testing the image for the 2D classification, we traverse the image at specific size window but at different sizes. We used **Pyramid Scaling methodology,** where the target image is downscaled as a constant multiple and the window size is maintained the same. This procedure helps with finding the object at different sizes in the same image. Also, the window is moved some pixels and not at each pixel so as to not increase the runtime exponentially.

2.2.5 Mapping 2D to 3D

When testing, we change the 3D cluster to 2D cluster. After labeling using the 2D classifier, we need to back-project it back to its original coordinate. We implemented this mapping from 2D data points to 3D points.

For every cluster we have segmented, we first find all the VFH features and find the closest training example class (with distance). We score it with the inverse of the distance from the most probable training instance class. We, then get the 2D image representation of the cluster.

2D features described above from this cluster are then used to get the label using SVM, which is trained on positive and negative image of the specific target object.

2.2.6 Re-estimating Scores

Each classifier, 2D and 3D, gives the bounding box of the detected object. The bounding box is basically the cluster's 3D bounding cube while in 2D it's the rectangular box in which object was detected. Sometimes, the segmented cluster is not a perfect box around the object but is usually the box that contains the object as well as some other connected components involving the target object, which is generally part of human skeleton with the object like the hand holding the mug. To counter it, we do re-estimation of the cluster boundaries when the bounding box from 2D and 3D classifiers differs greatly, here 3.0 multiple of th 2D box width and height.

The re-estimation creates the new 3D bounding box based on 2D one projection with some added padding. And score is re-computed based on the new box.

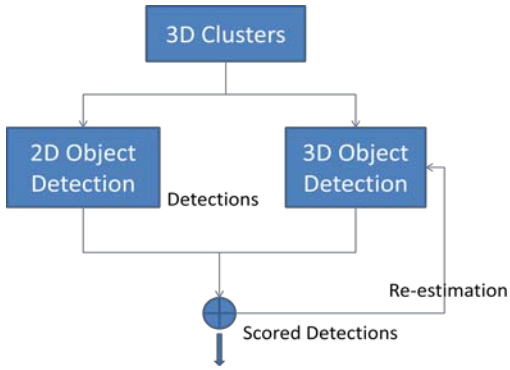

Fig.4 Over-all architecture

2.2.7 Inference

We used the scoring from both 2D and 3D features to build an accumulative inference model. The model is simply the sum of the both scores. The k-NN score is just the reciprocal of the distance from the closest training example.

2.2.8 Analysis

HoG versus HSV: To see which feature is important where, we tested the dataset on two types of features – only HoG and HSV + HoG. We did not see the result on just the HSV feature as HSV is just a complimentary feature. The results are as shown in Table II.

In all the cases, the precision decreased when used only HoG features (we are not using the VFH features here). For banana and pear, the precision did not decrease greatly as expected as the shape alone is not so discriminative. It might be because in the test set the other items as well as background was not so similar to pear and banana shape. Can, which does not have color as an important feature, did quite well without VFH feature.

Euclidean versus Region-growing Clustering: Clustering is an important step for object detection and thus choosing the right scheme is must for robust detection. We tested the some 3D scenes on both of the clustering algorithms separately to see which performs best. The results are shown in Table I.

From the results, the region-growing algorithm performs slightly better in cluttered space but not major difference in accuracy when used in uncluttered space. Thus, only difference favoring Region-growing is that it favors under-segmentation, which sometimes is helpful in bringing up the recall. Both of the test set involved the human skeleton mingling with the object.

TABLE I.

Class	Total Samples	Precision (Euclidean)	Precision (Region-grow)
Mug	57	66.7%	65%
Cereal-Box	70	40%	43%

Precision Table for different clustering methods

TABLE II.

Class	Total Samples	Precision (HoG)	Precision (HoG + HSV)
Pear	44	50%	52.27%
Banana	44	38%	40.9%
Can	44	77.2%	79.4%

Precision Table for different features

2.3 Object Tracking

Object tracking is done for robust detection as well as to counter occlusions. We implemented a Discrete Kalman Filter to track the object using the object detections output as the observations.

2.3.1 Kalman Filter

The Kalman filter addresses the general problem of trying to estimate the state $x \in R^n$ of a discrete-time controlled process that is governed by the linear equation.

$$x_k = Ax_{k-1} + w_{k-1}, \quad w \; is \; process \; noise \quad \textbf{(1)}$$

with a measurement that is,

$$z_k = Hx_k + v_k, \quad v \; is \; measurement \; noise \; \textbf{(2)}$$

The n x n matrix A in the difference equation equation (1) relates the state at the previous time step k-1 to the state at the current step k.

The Kalman filter estimates a process by using a form of feedback control: the filter estimates the process state at some time and then obtains feedback in the form of (noisy) measurements. The time update equations are responsible for projecting forward (in time) the current state and error covariance estimates to obtain the a priori estimates for the next time step. The measurement update equations are responsible for the feedback.

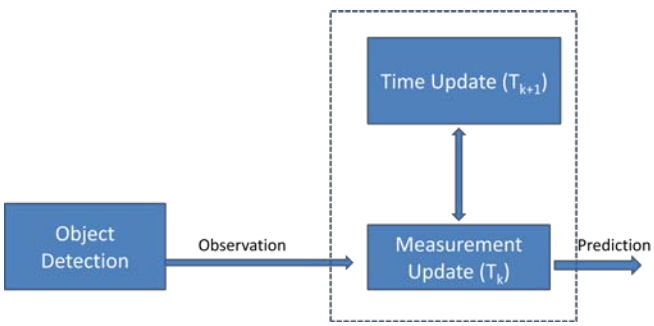

Fig.5 Tracking

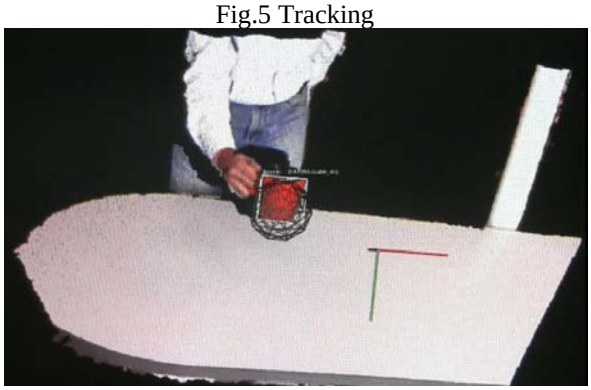

Fig.6a mug tracking

Fig. 6b Cereal-box tracking

Fig. 6c Cereal-box tracking

2.3.2 Model

The model of the Kalman Filter is defined by the State matrix A, the measurement matrix H, the process noise Q and the measurement noise R.

The state in our model is defined to be the x, y and z position of the object as well as the velocity component dx, dy and dz which brings the movement of the object into the model. Thus, A = $(x \; y \; z \; dx \; dy \; dz)^T$.

While the measurement matrix H is defined to be $(x \; y \; z)^T$. Since there could be two types of observation errors:

- Displacement error – which is the error in location of the bounding box containing the object in which the prediction is accurate.
- False prediction – which is the false detection from object detection module. Here the observation is completely off.

The error in observation could sometime be large and sometime be small. Thus the standard deviation chosen for the model is 50 pixels. The process noise, Q, based on this is 1.2 and so is the measurement noise, R.

2.3.3 Tracker

Prediction of the state is done in two steps- one with time update and the measurement update. The prediction step uses the state matrix to predict the next step using the equation (1). The measurement update steps corrects the prediction error by using the observation from the object detector. The kalman gain in our case is:

$$K_k = \frac{{P_k}^-}{({P_k}^- + R)},\ {P^-}_k \ is\ the\ error\ covariance$$

And the new measurement is given by,

$$x_k = {x_k}^- + K_k(z_k - {x_k}^-)$$

TABLE III.

Class	Total Samples	True Positives	Precision
Food Items			
Pear	42	36	85.7%
Banana	42	21	50%
Can	42	30	71.4%
Household Items			
Mug (uncluttered	57	37	65%
Cereal-Box (cluttered)	70	30	43%

Object Detection Precision Table

2.4 Dataset Collection and Object Labeling

2.4.1 Collection

Dataset collection is done by placing the target objects on table and taking snapshots using kinect camera at varying poses by rotating the object manually. Also, the tilt angle is also varied – 10 and 45 degrees.

2.4.2 Dataset Preprocessing

The collected dataset is preprocessed by segmenting out the background and stored as point cloud file. Also, the RGB image of the cluster is stored useful for 2D classifier training.

2.4.3 Labeling

Labeling of only RGB images is done and manual bounding box is defined by marker utility developed specifically.

III.

IV. EXPERIMENTS

3.1 Data

For training, we label all the positive images with their bounding boxes. Also, more negative instances are generated from negative sample by using multi-scale image scanning. We used the RGB-D object dataset from UWash as well as the self collected dataset.

For instance class, we trained it on around 1000 positive samples and 35000 negative samples.

TABLE IV.

Class	Total Samples	True Positives	Precision
Mug (uncluttered	57	38	66%
Cereal-Box (cluttered)	70	32	45%

Object Tracking Precision Table

3.2 Results

3.2.1 Object Detection

For the two classes – Food and Household items, involving five objects - Can, Pear, Banana, mug and cereal-box, the precision values are in Table III. Cereal-box had the least precision because of the cluttered environment.
Also, segmentation was problem in most of the frames in the experiment.

3.2.2 Tracking

Test set of tracking was the Mug and the Cereal-box. Mug had a good precision as the environment was not cluttered and the detection was good in most of the frames. Thus the tracking precision is mostly dependent on the detection rate. Kalman filter can, to an extent, detect the discrepancy but when false detections are at the extreme corners of a frame, it would fail.
The results are in Table IV.

REFERENCES

[1] Hema Swetha Koppula, A. Anand, T. Joachims, A. Saxena, “Semantic Labeling of 3D Point Clouds for Indoor Scenes”

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

[3] Paul Viola, Michael Jones. *Rapid Object Detection using a Boosted Cascade of Simple Features*. Conference on Computer Vision and Pattern Recognition (CVPR), 2001, pp. 511-518.N. Kawasaki, “Parametric study of thermal and chemical nonequilibrium nozzle flow,” M.S. thesis, Dept. Electron. Eng., Osaka Univ., Osaka, Japan, 1993.

[4] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. SIGGRAPH, 2001.

[5] T. Rabbania, F. A. van den Heuvelb, G. Vosselmanc , "Segmentation of point clouds using smoothness constraint", ISPRS Symposium

[6] Burges, Christopher J. C.; A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery