3D Vision for Indoor Robotic Manipulation using Kinect V2 Sensor with Point Cloud Library (PCL)

Author

Noaman Mehmood

Reg. Number: NUST201362032MSMME62113F

Supervisor

DR. YASAR AYAZ

DEPARTMENT OF ROBOTICS AND INTELLIGENT MACHINE

SCHOOL OF MECHANICAL AND MANUFACTURING ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

August, 2016

3D Vision for indoor Robotic Manipulation using Kinect V2 Sensor with Point Cloud Library (PCL)

Author

Noaman Mehmood

Reg. Number: NUST201362032MSMME62113F

Thesis submitted in partial fulfillment of the requirements for the degree of

MS Robotics and Intelligent Machine Engineering

Thesis Supervisor

DR. YASAR AYAZ

Thesis Supervisor's Signature: ____________________________

DEPARTMENT OF ROBOTICS AND INTELLIGENT MACHINE

SCHOOL OF MECHANICAL AND MANUFACTURING ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

August, 2016

Declaration

I certify that this research work titled *“3D Vison for indoor Robotic Manipulation using Kinect V2 with Point Cloud Library (PCL)”* is my own work. The work has not been presented elsewhere for assessment. The material that has been used from other sources it has been properly acknowledged / referred.

Signature of Student

Noaman Mehmood

NUST201362032MSMME62113F

Language Correctness Certificate

This thesis has been read by an English expert and is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the University.

Signature of Student
Noaman Mehmood
NUST201362032MSMME62113F

Signature of Supervisor
Dr. Yasar Ayaz

Copyright Statement

Acknowledgements

In the beginning, the name of ALLAH, the creator and the nurturer of the all the universes. All praises are for HIM who being REHMAAN has given everything without asking and who being RAHIM has forgiven every time. Everything which will follow in this research, every bit of word and every part of knowledge is due to HIM.

I am profusely thankful to my beloved parents and Family who gave me confident to be here and be the part of such knowledge.

I would also like to express special gratitude to my supervisor Dr. Yasar Ayaz for his help throughout my thesis. He has been a great motivator. In every moment of the research he has guided me. I have learnt too many things from him a part from this project.

I would also like to express my special thanks to Dr. Muhammad Naveed and Mr. Fahad Islam for their guidance.

Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance to my study.

Dedicated to my exceptional parents and my supervisor whose tremendous support and cooperation led me to this wonderful accomplishment

Abstract

In order to perform any robotic task object recognition plays a very vital role. In this research we propose a real time recognition system that is able to recognize single object using the latest Kinect v2 depth sensor. This sensor captures RGB images with depth information for each pixel and is also low cost. The recognition system consists of two domains i.e. training and testing. For training, a point cloud including the single object is retrieved, processed and its global feature named as Clustered Viewpoint Feature Histogram (CVFH) using Point Cloud Library (PCL) is calculated and for all the required objects a database is made. For testing a point cloud from Kinect V2 is obtained and after preprocessing (CVFH) is calculated for each cluster obtained and using K-Nearest Neighbor (KNN) method nearest feature match is searched from database. If a match is found, then object is considered as recognized. The system was tested successfully. CVFH has performed well than VFH in the case of partially occluded objects and noisy environments and since it considers geometry of object so its computational cost is also low as compared to other local features so it is quite suitable for the situation where computation time is main concern.

Key Words: Clustered viewpoint feature histogram (CVFH); Point cloud library (PCL); K-Nearest Neighbor; Depth; Vision

Table of Contents

List of Figures

Chapter 1: Introduction

1.1. Introduction

Object recognition system finds objects in real world from an image of the world using the object models which are known already. Normally there are two domains in machine vision dealing with Object recognition.

1.1.1 **2D Vision**

In 2D Vision light intensity at each pixel is captured and whole scene is projected on a plane to retrieve 2D image as shown in Figure 1. There are many techniques available which are being used in 2D object recognition i.e. Speeded Up Robust Features (SURF), Scale Invariant Feature Transform (SIFT). In 2D vision depth information is not available which is main drawback of the 2D vision as far as grasping of objects is concerned.

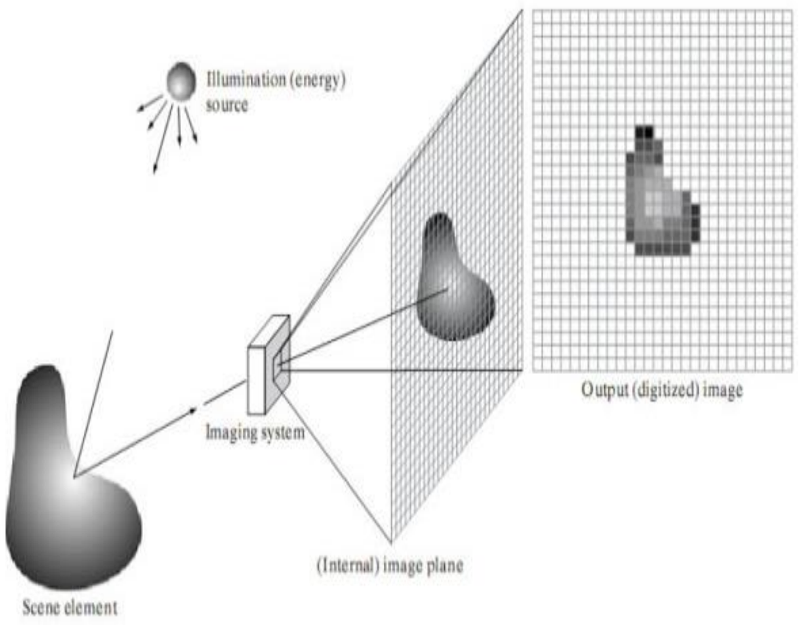


Fig. 1.1.1.1 2D Image Acquisition [1]

1.1.2 **3D Vision**

In 3D vision points coordinates of points in a scene are captured, which is called Point Cloud. 3D vision uses depth information so this is why it is quite useful. Initially there was not that much research in 3D vision as depth sensors were quite expensive. With the invent of cheap depth sensor like Kinect Sensor V2, research in the field of machine vision has increased rapidly. 3D vision has several advantages over 2D vision.

1.1.3 Advantages of 3D vision over 2D vision

Because of depth information 3D vision has several advantages over 2D vision. Using 3D vision exact position and orientation of object can be calculated relative to sensor. Moreover, using 3D one can determine exact shape of objects so it is quite useful as far as object recognition is concerned.

Research community in the field of machine vision classify object recognition in two main types i.e. instance level and category level. In instance level recognition one tries to recognize the instance of different objects while in category level recognition general category of object is determined.

1.2. Problem Statement:

Being human we can identify different objects quite easily and even can grasp different objects easily but this task is not easy for a robot. As the human also have memories for different objects in brain so in order to recognize an object we also need object models of general objects that a robot may meet while performing any task so that we can train our robot. So if we want a robot to recognize an object then we have to train the robot so that it can know what is the object.

1.3. Aims & Objective of the Thesis:

The aim and objective of the thesis to study existing 3D recognition techniques and then by using one most useful technique in order to recognize a single object using Kinect V2 Sensor with Point Cloud Library. This single object can be partially occluded as well.

1.4. Thesis Overview

Chapter 2 – Background

The chapter will give a brief overview of existing 3D object recognition techniques being used for object recognition using a point cloud. The techniques include Point Feature Histogram

(PFH), Fast Point Feature Histogram (FPFH), Viewpoint Feature Histogram (VFH) and Clustered Viewpoint Feature Histogram (CVFH).

Chapter 3 – Methodology

This chapter describes the algorithm devised to achieve the objective. It contains block diagram of both phases i.e. testing and training.

Chapter 4 – Depth Sensors

In this chapter available depth sensors are highlighted briefly i.e. Stereo Cameras, Time of Flight Cameras, Structured Light etc.

Chapter 5 – Training

In this chapter we discussed how training was performed. All the steps of testing phase were discussed. This chapter discusses the feature used in order to identify an object. Moreover, this chapter discusses all the steps involved in order to recognize a single object using Kinect V2 Sensor I.e. Retrieval of Point Cloud using Kinect V2 Sensor, Filtration of the Point Cloud obtained, Computation of the CVFH Feature etc.

Chapter 6 - Testing

In this chapter we discussed how testing was performed and what was the algorithm adapted to achieve goal.

Chapter 7 - Experimental Results

In this chapter experimental results of different objects are under different conditions were discussed.

Chapter 8 – Conclusion & Future Work

The research will be concluded with theses chapters followed with the future recommendations

Chapter 2: Background

The basis of 3D object recognition is to find set of correspondences between two point clouds, one of them containing object which we want to recognize. So far researchers have worked with points to store XYZ coordinates and color information (RGB) but it was found that two points cloud may share same coordinates although they belong to different surfaces. Likewise using color information takes us back to light related issues. So there was a definite need to devise new feature in order to recognize an object effectively. So the idea of descriptor was proposed.

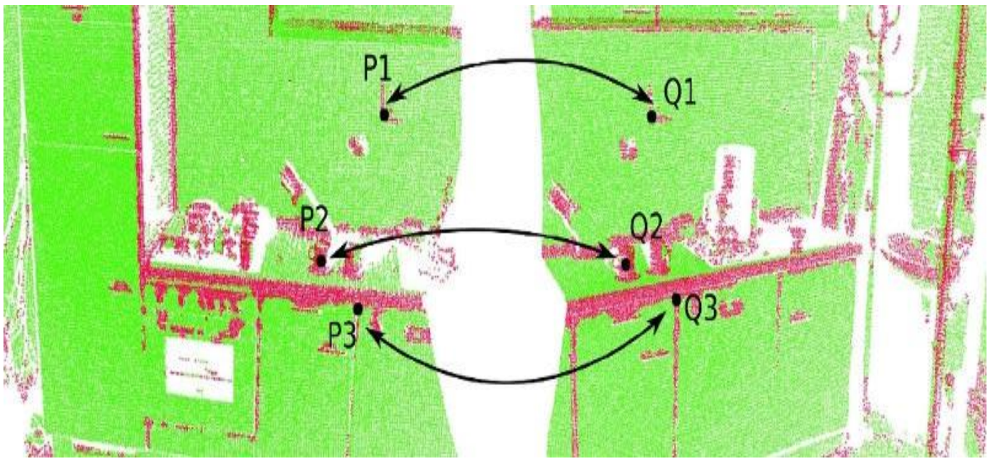


Fig. 2.1 3 Point Feature correspondence between two clouds [2]

Descriptors are precise signature of a point that carries a lot of information about the local geometry of a point. Many 3D descriptors both local and descriptors are proposed so far and each one has its own method of computing the local information of point geometry. Some compute difference between point and its neighbor and some compute difference between angles of normal of a point and its neighbors.

One more step which is performed further to reduce the size of descriptor is binning the result in to a histogram. On x axis there are no of bins and on y axis there are no of samples falling in each bin.

Local descriptors are computed for individual points in a point cloud. They give us information about the local geometry around that point. The advantage of local descriptor is that it can provide us more detailed information about an object and can be used for object recognition however it has one major drawback that its computation cost is very high.

2.1 Point Feature Histogram (PFH)

Point Feature Histogram (PFH) [3] gives us geometry information of a point by computing the difference between directions of normal. It makes pair of all point lying in vicinity and then for each pair a fixed coordinate frame is calculated from their normal.

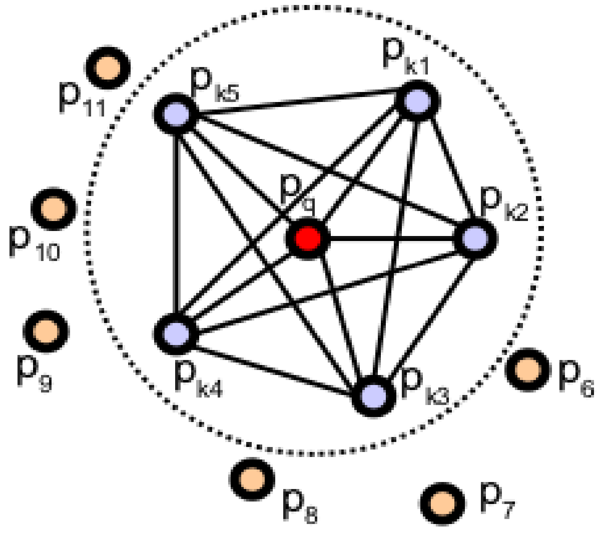
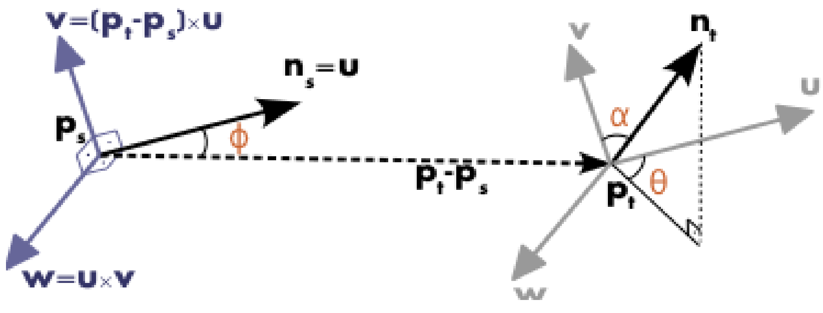


Fig. 2.1.1 Points Pair Established [2]



$\mathsf{v}=(\mathsf{p}_t-\mathsf{p}_s)\times\mathsf{u}$
$\mathsf{n}_s=\mathsf{u}$
p_s
ϕ
$\mathsf{p}_t-\mathsf{p}_s$
$\mathsf{w}=\mathsf{u}\times\mathsf{v}$
v
n_t
u
α
θ
p_t
w

$$\mathsf{u} = \boldsymbol{n}_s \qquad \alpha = \mathsf{v} \cdot \boldsymbol{n}_t$$

$$\mathsf{v} = \mathsf{u} \times \frac{(\boldsymbol{p}_t - \boldsymbol{p}_s)}{\|\boldsymbol{p}_t - \boldsymbol{p}_s\|_2} \qquad \phi = \mathsf{u} \cdot \frac{(\boldsymbol{p}_t - \boldsymbol{p}_s)}{d}$$

$$\mathsf{w} = \mathsf{u} \times \mathsf{v} \qquad \theta = \arctan(\mathsf{w} \cdot \boldsymbol{n}_t, \mathsf{u} \cdot \boldsymbol{n}_t)$$

Fig. 2.1.2 Fixed coordinate frame and angular Features [2]

Using the frame difference between normal can be calculated in terms of three variables. These three variable along with the fourth variable named as Euclidean distance between these two frames are saved and then the result is binned into a histogram after computing this for all pairs.

Normally the fourth variable i.e. Euclidean distance is neglected as it varies with viewpoint so three variables are used and concatenated to make histogram.

One more version of PFH is also available which includes color information as well and that is named as PFHRGB.

2.2 Fast Point Feature Histogram (FPFH)

PFH was found very accurate but its computational cost was too high which was a main hurdle as far as its implementation for real time application is concerned. So in order to reduce the computation cost another feature named as Fast Point Feature Histogram (FPFH) [4] was devised.

It only takes into account the direct connection between current key point and its neighbor. It doesn't consider neighbors of neighbor so it significantly reduces the computational cost. If we have a cloud with n key points and n neighbor, then complexity of Point Feature histogram was found to be $O(nK^2)$ which was reduced to $O(nK)$ and this new resulting Histogram is named as Simplified Point Feature Histogram (SPFH).

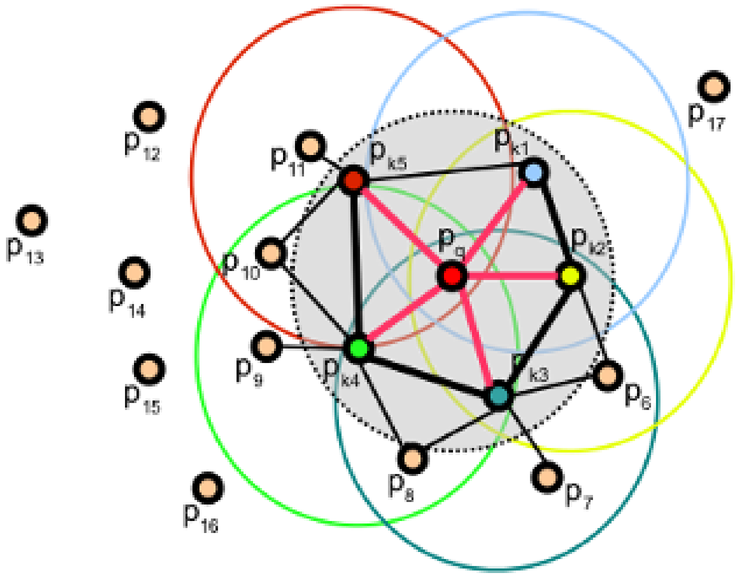


Fig. 2.2.1 Point pairs established when computing the FPFH for a point [2]

One more step is performed in which for each point its K neighbors are redetermined and neighboring SPFH values are used to weight the final histogram of query Point P_q. This descriptor performed very well. It reduced computational cost retaining the accuracy of the point feature histogram.

2.3 Signature of Histogram of Orientation (SHOT)

This descriptor [5] uses information about surface within a spherical structure which is further divided in thirty-two volumes.

After this for each volume one-dimensional histogram is calculated. To achieve final histogram, all local histograms are stitched together.

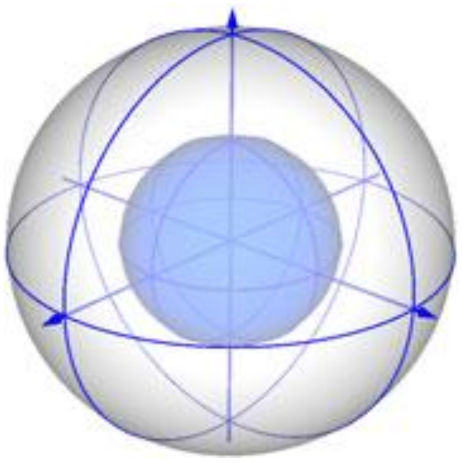


Fig. 2.3.1 Support structure to compute SHOT [5]

2.4 Spin Image (SI)

Spin Image [6] is very old descriptor. It was devised in 1997 and still it has useful applications. There are some surfaces which are made by vertices, edges, this descriptor was meant for these types of surfaces. In this support structure of cylinder is used with a given radius, height and aligned with normal. Volumes are also obtained out of this cylinder.

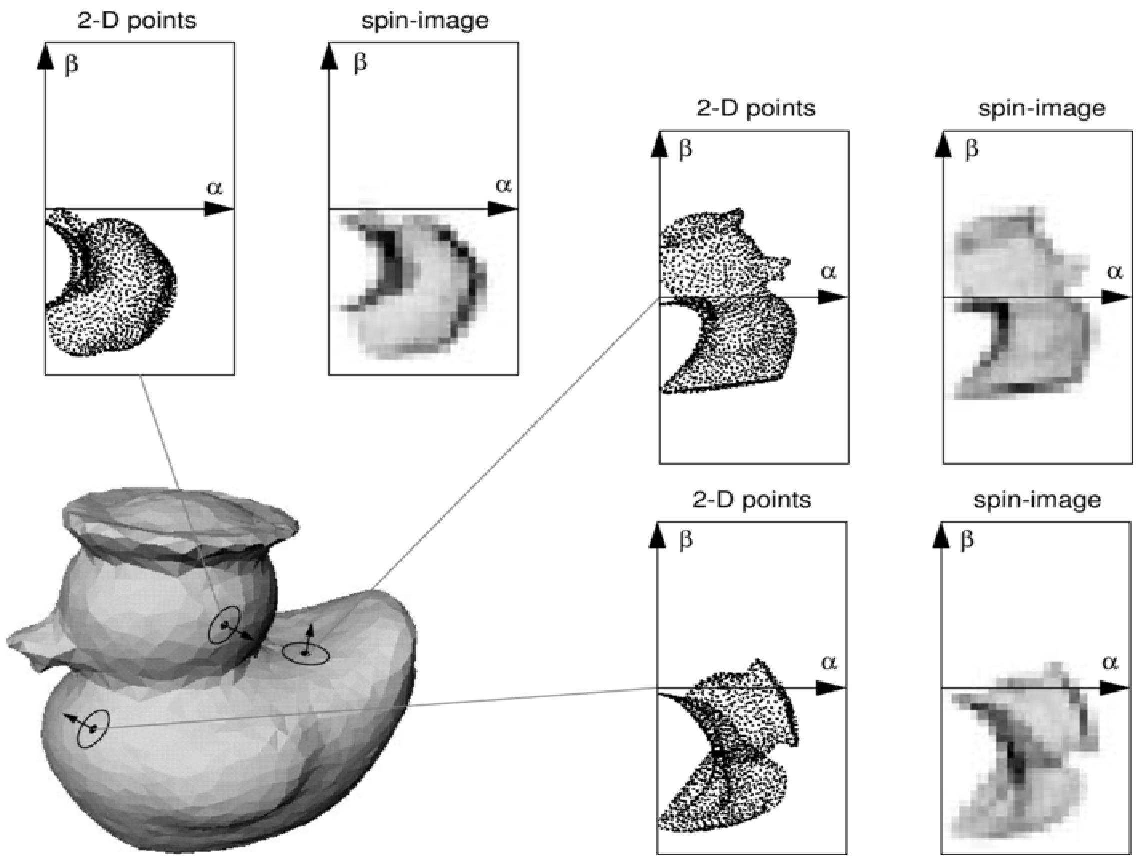


Fig. 2.4.1 Spin Image computed for 3-point model [6]

2.5 Viewpoint Feature Histogram (VFH)

Viewpoint Feature Histogram [7] has its root in Fast Point Feature Histogram (FPFH), however in this descriptor information of viewpoint is added too. Although FPFH was found to be very accurate and very reasonable as far as computational cost is concerned but FPFH cannot be used for object pose estimation. Therefore, viewpoint information was added so that it can be used for Pose estimation as well. It consists of mainly two components.

- Viewpoint Direction Component
- Extended Fast Point Feature Histogram

In order to compute Viewpoint direction component first of all centroid is computed. Centroid is computed by taking the average of X, Y and Z coordinates of all the points lying in the object. Once centroid is computed then vector between centroid and viewpoint is computed and finally for all the points in the cluster the angle between this vector and their normal is calculated and result is binned into a histogram.

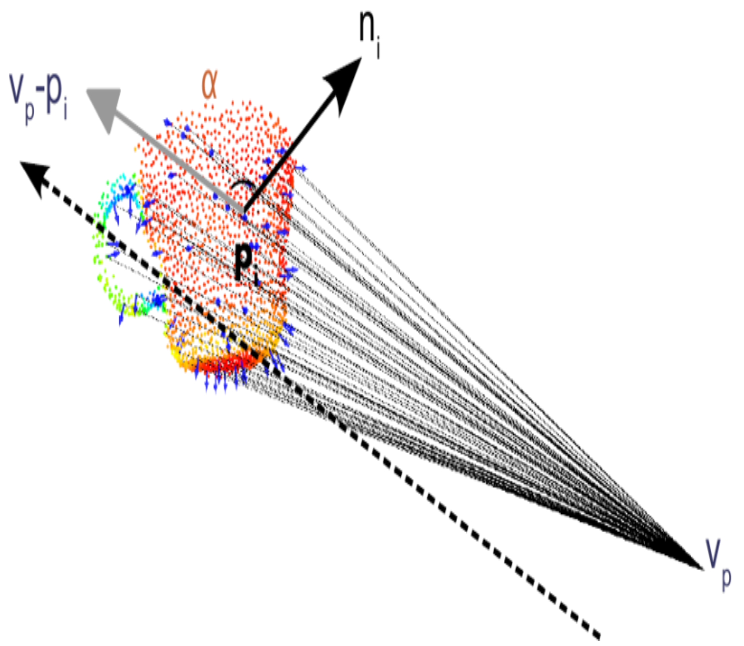


Fig. 2.5.1 Viewpoint Component of VFH [7]

In order to calculate the other component three variables are calculated using the same way as we do for FPFH but with a minor difference. It is computed for the centroid using computed viewpoint direction as its normal and setting all cluster's points as neighbor.

This only outputs one descriptor for the whole object making it a global descriptor. The resulting four histograms i.e. one for Viewpoint direction component and three for extended Fast Point Feature Histogram are concatenated to get the final Viewpoint Feature Histogram. This histogram was very useful in the sense because one can get information of object's pose as well which is very useful as far as grasping of an object is concerned.

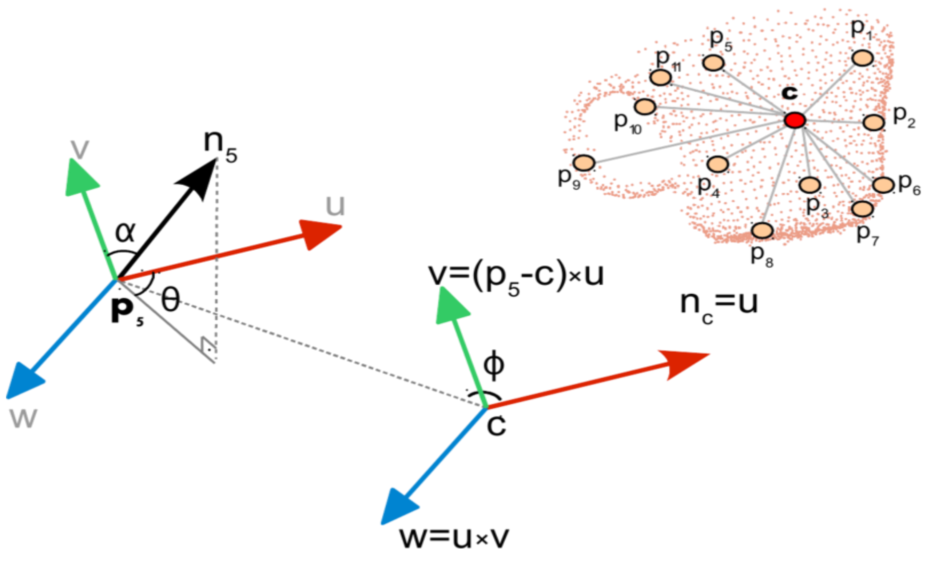


Fig. 2.5.2 Extended Fast Point Feature Histogram [7]

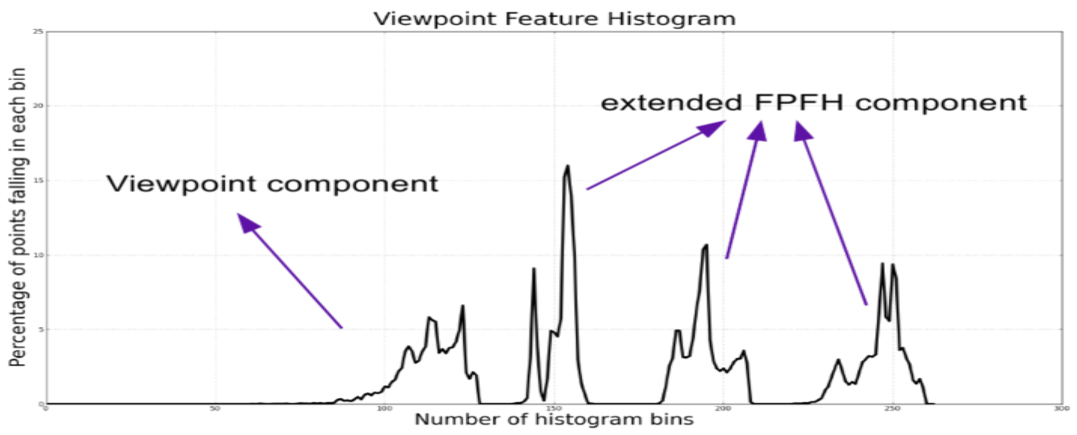


Fig. 2.5.3 Viewpoint Feature Histogram [7]

On X axis there are number of histogram bins and on Y axis there is percentage of points falling in each bin.

2.6 Clustered Viewpoint Feature Histogram (CVFH)

Viewpoint Feature Histogram is very useful for object recognition and pose estimation but it has main drawback that it cannot handle partial occlusion or noisy environment. If some points of an object's cluster are missing due to sensor errors, or due to occlusion then the computed centroid will be different than that of original so in this case no positive match will be available in the data base.
In order to handle these kind of errors a new descriptor was proposed named as Clustered Viewpoint Feature Histogram (CVFH) [8].

In this descriptor instead of computing Viewpoint Feature Histogram for the whole cluster, the object is first divided into stable smooth regions using Region growing segmentation technique.

In this technique the angle between normal of points and difference of curvatures are checked to see if the points belong to same smooth surface or not.

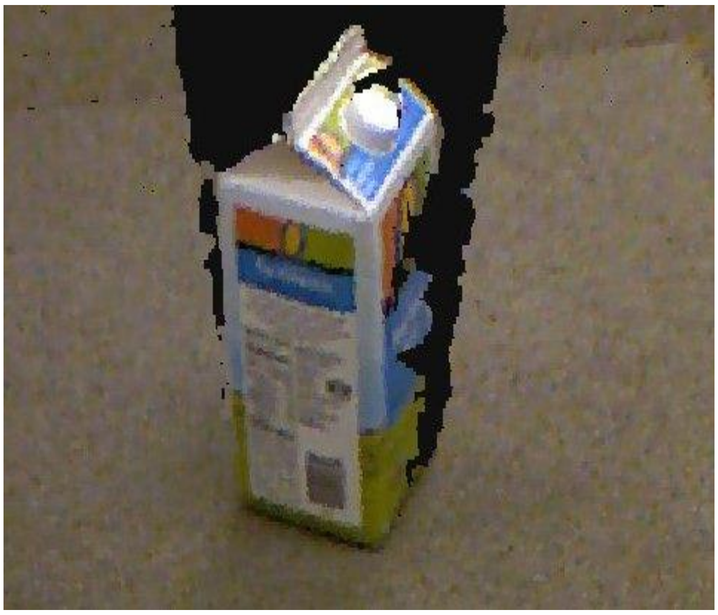


Fig. 2.6.1 Typical occlusion issues in a Point Cloud [8]

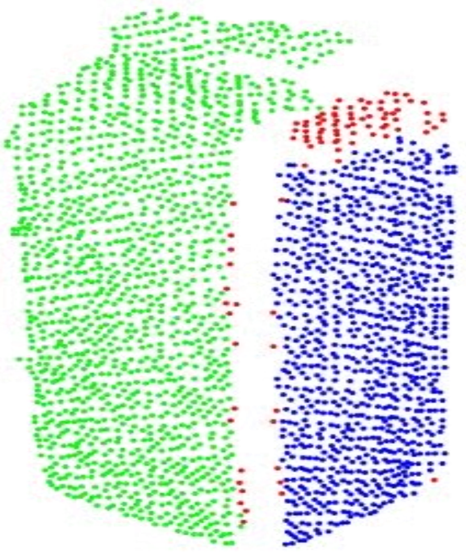


Fig. 2.6.2 Object regions computed for the CVFH [8]

Once stable regions are computed then for each stable region VFH is computed. This descriptor has an advantage over VFH that as long as a stable region of an object is fully visible then the object can be recognized.

Chapter 3: Methodology

3.1 Overview

In this chapter, algorithm and methodology of our system is discussed. The objective of our research was accomplished using two main steps.

- Build Model Database/Training
- Testing

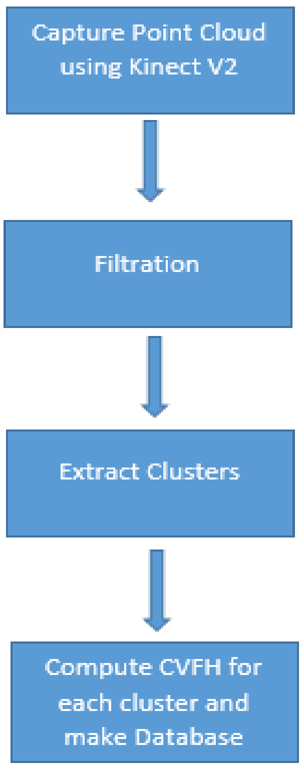

Capture Point Cloud using Kinect V2
Filtration
Extract Clusters
Compute CVFH for each cluster and make Database

Fig. 3.1.1 Steps performed to build Model Database

In Figure 3.1.1 all the steps involved to train our system are shown. First of all, information about environment is captured using Kinect v2 depth sensor. After getting the point cloud data undesired objects in the point cloud were removed using filters. Once undesired objects were removed then next task was to perform segmentation. So clusters were extracted using Euclidean Segmentation.

Once segmentation was done next step was to compute clustered viewpoint feature histogram for the whole cluster one by one and store the result in data base in the form of K-dimensional tree and hence database was created using the aforementioned steps for all the desired objects.

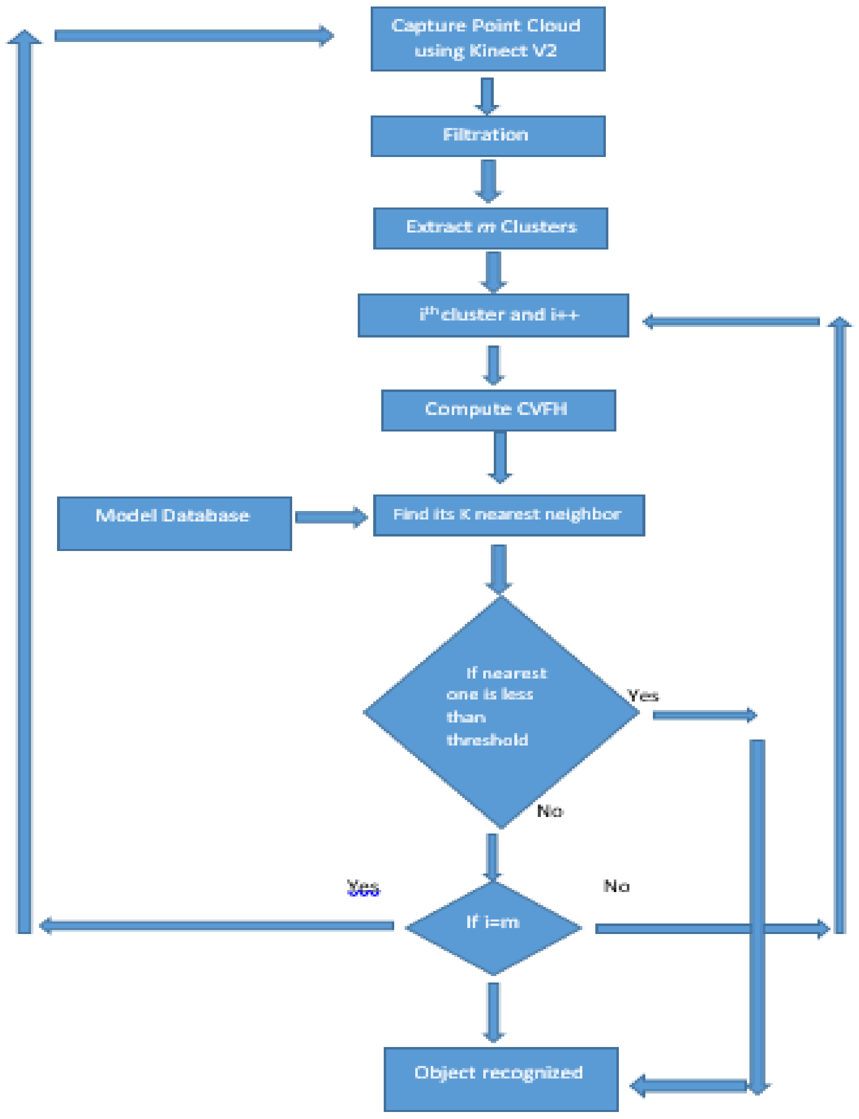


Fig. 3.1.2 Testing Phase

All the steps of the algorithm in order to achieve the objective are shown in the Fig 3.1.2 and all these steps will be described with detail in the subsequent chapters.

Chapter 4: Depth Sensors

In order to retrieve data, we need a depth sensor. Depth sensors give us accurate distance of points found in a scene relative to the sensor. There are different depth sensors available, some are fast and some are slow, some are very expensive and each one has different working principle. Few are highlighted below. There are normally three types of depth sensors available.

- Stereo Cameras
- Structured Light
- Time of Flight Cameras

4.1 Stereo Cameras:

In this type of camera two same cameras are joined together which grab information of slightly different scenes and after analyzing the difference between two scenes depth information about each point in the image is achievable.

This sensor is quite cheap but it is not very accurate. In order to minimize errors proper calibration of both sensors is very important. Moreover, its performance in bad light is very inaccurate.

Fig. 4.1.1 Stereo Camera [9]

4.2 Structured Light Cameras:

Structured light sensor like Kinect V1 capture scene information by projecting infrared light on the object. When we look the pattern from a perspective different than projector's then the pattern is observed as distorted. Using this distortion information depth information can be obtained. The resolution of this sensor is 640X480 with 30 frame per second. Its working range is about 3-6 meter. They cannot see very small objects as this is its main drawback however it has an advantage that its cost is quite low as compared to other time of flight based cameras.

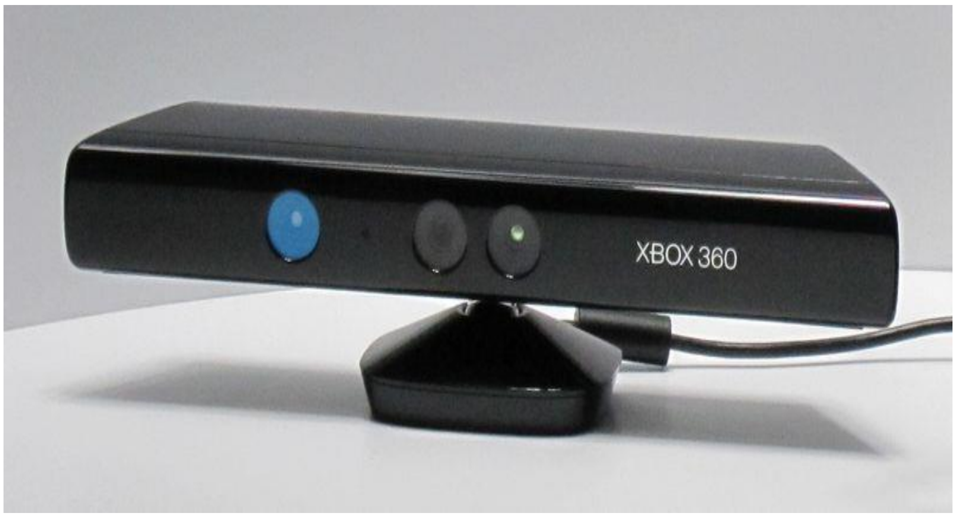


Fig. 4.2.1 Kinect Sensor V1 [10]

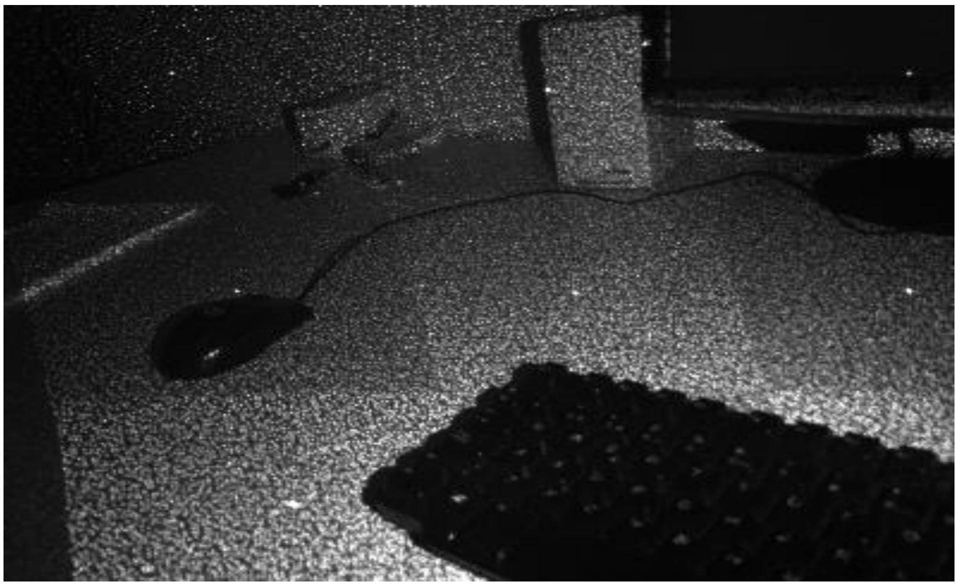


Fig. 4.2.2 Pattern of infrared light projected by Kinect [11]

4.3 Time of Flight (T.o.F) Cameras:

These sensors transmit a pulse of light which travel certain distance and after striking with desired object it returns back and then by measuring the time taken distance information can be obtained. Since speed of light is constant so to get the range of desired object a simple formula can be used. These sensors are very accurate and these are not affected by poor light conditions. But only drawback of these sensors is that these type of sensors are normally very expensive.

- **LIDAR (Light + Radar)**
- **KINECT V2**

Lidar is mounted on a platform that can rotate at a very fast rate. It scans seen point by point. These sensors are accurate but these are quite expensive. Moreover, the resolution of these sensors are too high so in order to process the data computation cost is very high for real time application.

m

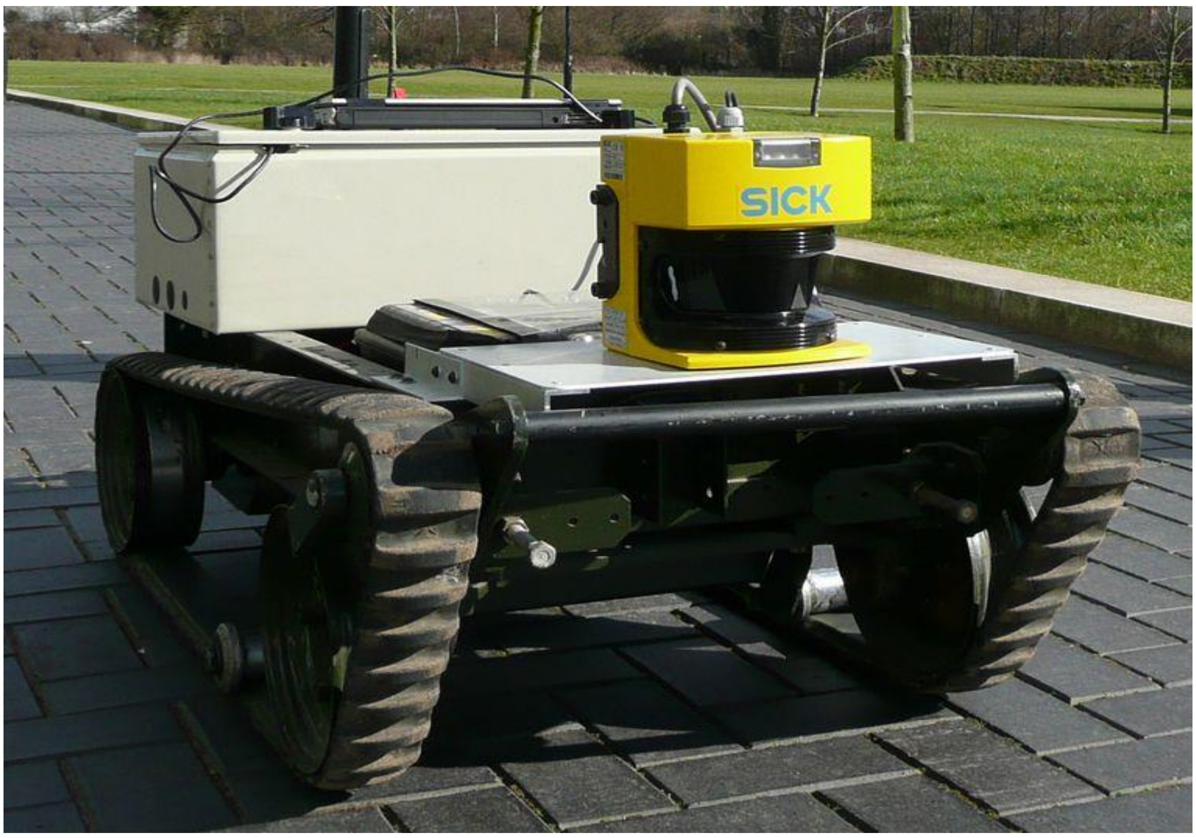


Fig. 4.3.1 Sick Laser Rangefinder mounted on a Mobile Robot [12]

Kinect v2 is the another time of flight sensor and its cost is very low. It doesn’t perform point scanning as Lidar does rather it transmit single pulse of light to retrieve information about the whole scene in a single frame.

A standalone version Kinect v2 was released by Microsoft in September 2014. Its performance is very good as compared to that of Kinect v1. It consists of a color camera full HD resolution, a depth sensor, IR sensor and an array of four microphones.
The RGB camera captures information at a resolution of 1920X1080 pixels. IR is used for real time acquisition of depth information with 512X424 pixel resolution. Field of view to sense depth horizontally is 70 degrees and 60 degrees vertically. Its range is from 0.5 m to 4.5 m.

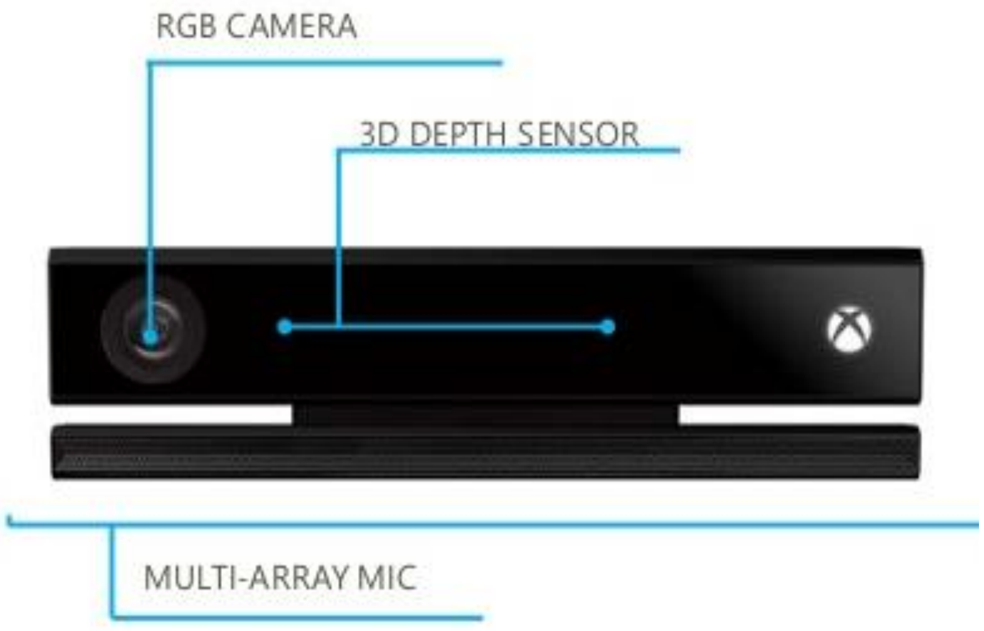


Fig. 4.3.2 Kinect v2 Sensor [13]

Chapter 5: Training

In order to build database models or train our system there are different steps involved as shown in Figure 3.1.1. All the steps are described below.

5.1 Retrieve Point Cloud using Depth Sensor:

First step in order to train our system is to capture point cloud data of scene using Kinect V2 Sensor. A captured point cloud is shown in Figure 5.1.1. Kinect v2 is 3D depth sensor. Being a cheaper depth sensor it is very extensively used in the field of machine vision and so it was also choosed for the research. In order to use the sensor to grab data open source driver *libfreenect2* [14] was used.

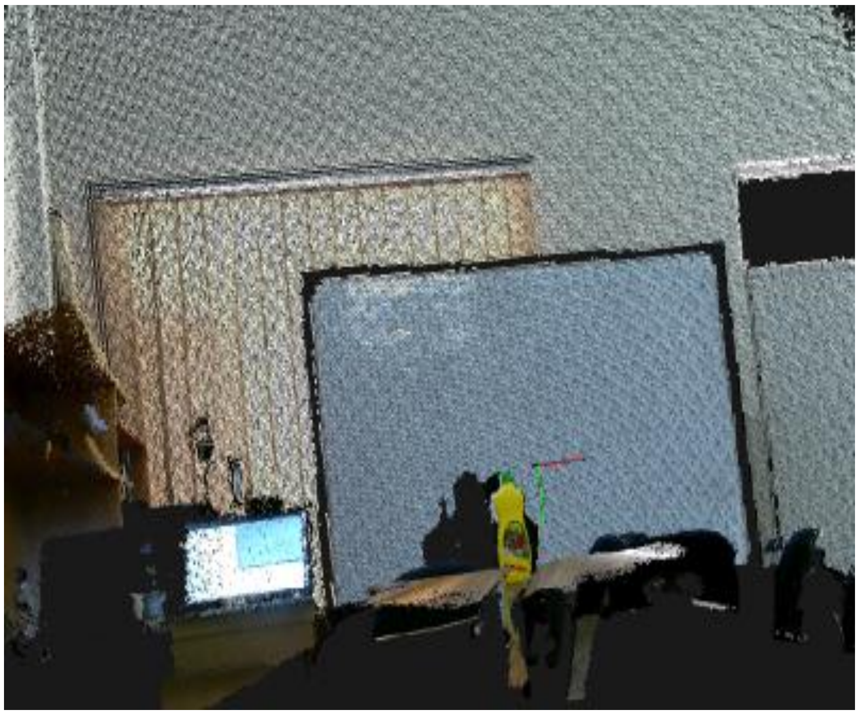


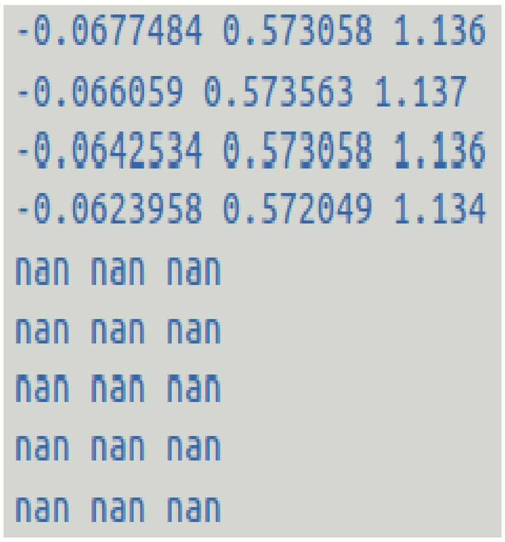
Fig. 5.1.1 Point Cloud captured using Kinect V2

5.2 Filtration:

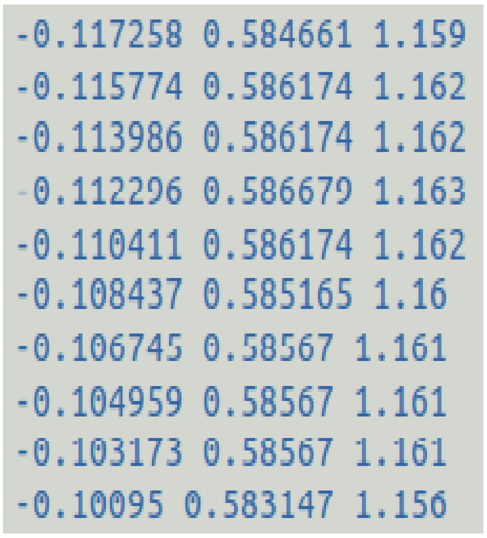
Once the point cloud data is captured then the next step is to remove undesired points in the data.

5.2.1 Not a Number (NaN) Filter

There were some NaN points in the data due to noise so these were removed using NaN filter available in Point Cloud Library as shown in Figure 5.2.1.1.



```
-0.0677484 0.573058 1.136
-0.066059 0.573563 1.137
-0.0642534 0.573058 1.136
-0.0623958 0.572049 1.134
nan nan nan
nan nan nan
nan nan nan
nan nan nan
nan nan nan
```



```
-0.117258 0.584661 1.159
-0.115774 0.586174 1.162
-0.113986 0.586174 1.162
-0.112296 0.586679 1.163
-0.110411 0.586174 1.162
-0.108437 0.585165 1.16
-0.106745 0.58567 1.161
-0.104959 0.58567 1.161
-0.103173 0.58567 1.161
-0.10095 0.583147 1.156
```

Fig. 5.2.1.1 Data before and after application of NaN Filter

After the removal of NaN points still there were points in the environment other than the desired object as you can see in the Figure 5.1.1 i.e. wall etc. These undesired points slow down the computation. In order to remove these points Passthrough Filter was used.

5.2.2 Passthrough Filter

In this filter user sets desired range along desired axis. So all those points who don’t fall in the given range by user along any axis are removed. This filter is quite useful in order to remove any undesired object in the scene.

After the application of Passthrough filter undesired objects in the scene were removed and we were left with only the desired objects as shown in Figure 5.2.2.1.

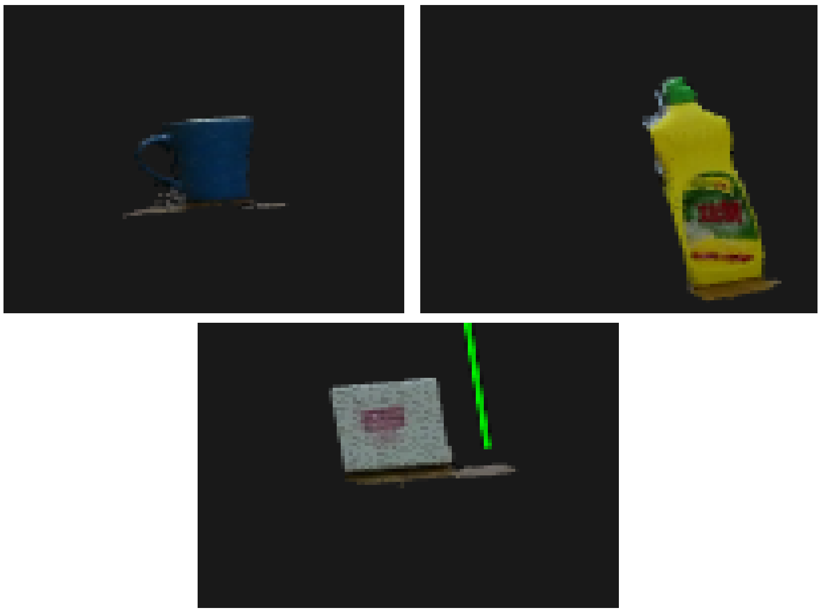


Fig. 5.2.2.1 Results after application of Passthrough Filter

5.2.3 Voxel Grid Filter

After application of Passthrough filter, still for the desired objects the depth data can be further reduced to improve the computation by down sampling the cloud. Down sampling will return us the same cloud equivalent to original cloud but it will have less points and hence it will reduce the complexity. Down sampling is accomplished using Voxel Grid

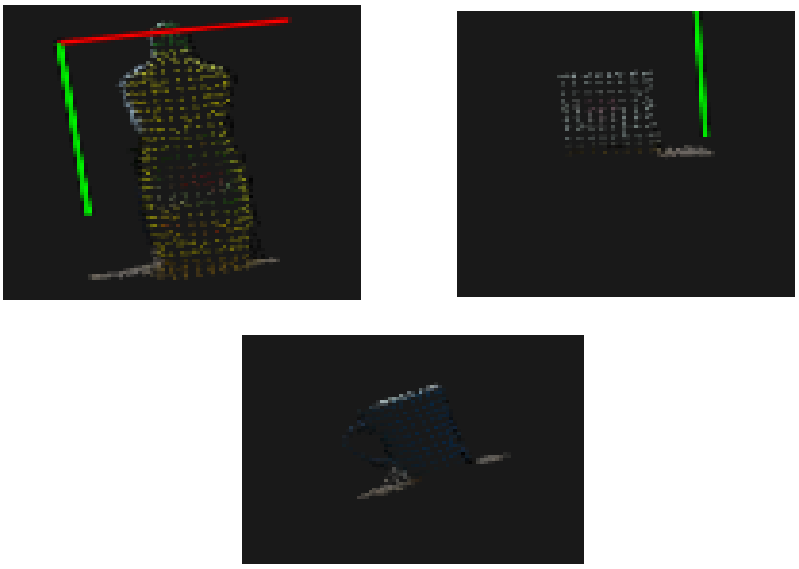


Fig. 5.2.3.1 Results after application of Voxel Grid Filter

. In this approach the whole cloud is splited into cube shaped parts and then centroid is computed using the mean values of the all the points in the cube shaped region.

After the application of Voxel Grid filters unneeded points in the cloud were decreased to a substantial extent as shown in Figure 5.2.3.1.

5.3 Clustering:

Once filtration of undesired objects is done then the next step is to extract clusters. Clustering is a process of breaking a cloud into different sections. It was accomplished using Euclidean segmentation as shown in Figure 5.3.1. In this method distance between two points is checked and if it lies in a certain range less than specified threshold then these both points are considered to be part of same cluster. In point cloud library there is a class named as *pcl:EuclideanClusterExtraction*. It uses the same method in order to extract clusters. If $a_{i,}$ b_i $\mathcal{E}$ P and P is a set of point cloud then the Euclidean distance dL can be computed using "(1)"

$$dL = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2} \qquad (1)$$

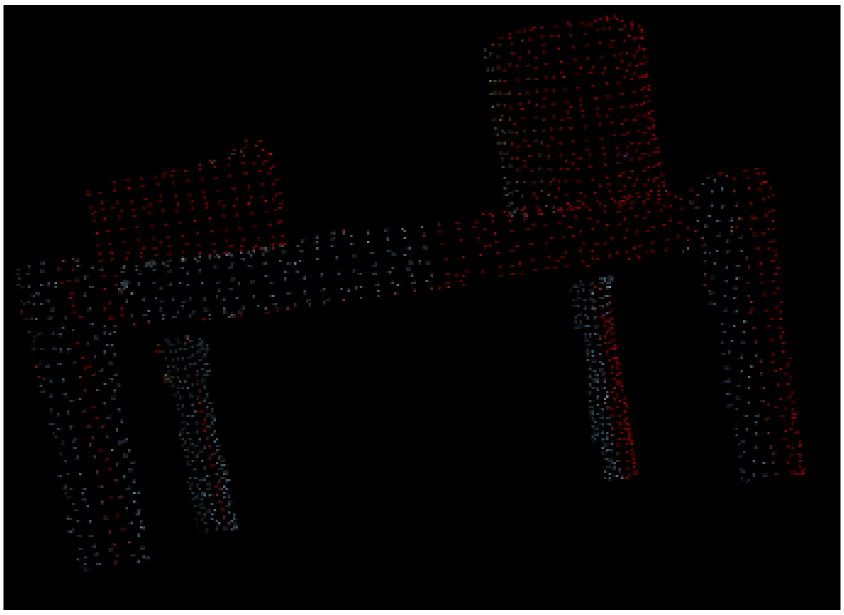


Fig. 5.3.1 Clustering

It works like seed fill algorithm. A point in a cloud is chosen as part of cluster and then it spreads to all near points and even then next to near points until no new point can be added so in this way a cluster is made.

In this algorithm Cluster minimum and maximum size matters a lot. If size of cluster is too small, then an object can be divided into different clusters and if it is made very large then more than one objects can be part of a single cluster and both cases are undesired so it is chosen very carefully.

5.4 Compute Clustered Viewpoint Feature Histogram (CVFH) Features:

Although Viewpoint Feature Histogram (VFH) is very effective as far as computational complexity is concerned as compared to others local features yet there is room for an improvement as it can’t perform well if some points of cluster are missing due to error in measurement etc. Because in this case computed centroid will be not accurate and it will not find any match in the database. So in order to remove the drawback of VFH an improvement is suggested. This improvement involves calculation of small stable regions using region growing clustering method for the object instead of calculating VFH for the whole cluster. So for all stable regions the descriptor is calculated. Since descriptor for each stable region is calculated during training and also during testing so in this way the recognition is possible as long as long one of the stable region is fully visible. This is named as Clustered Viewpoint Feature Histogram (CVFH).

We have tested this using Kinect V2 sensor. Limitation of this approach includes scale variance. It means that only those objects with particular dimensions can be recognized who were experienced by the system during training and it also don’t use the information of color or texture of the objects. Two objects can be recognized as same objects if they have same geometry no matter if there lies a difference between their color and textures.

The CVFH computed for three different objects i.e. Bottle, Cup and box using *pcl_viewer* is shown in the Figure 5.4.1, Figure 5.4.2 and Figure 5.4.3 respectively. On X-axis there lies no of histogram bins and on Y-axis there lies percentage of points falling in each bin.

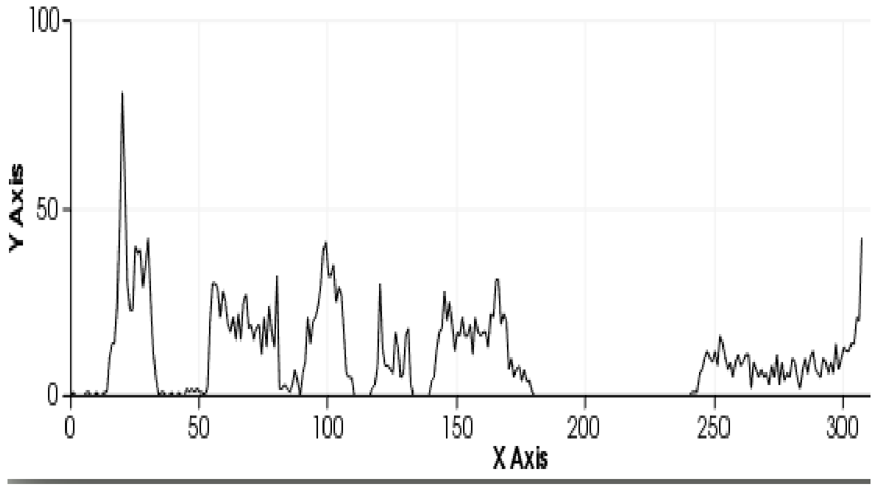


Fig. 5.4.1 CVFH Representation for Bottle

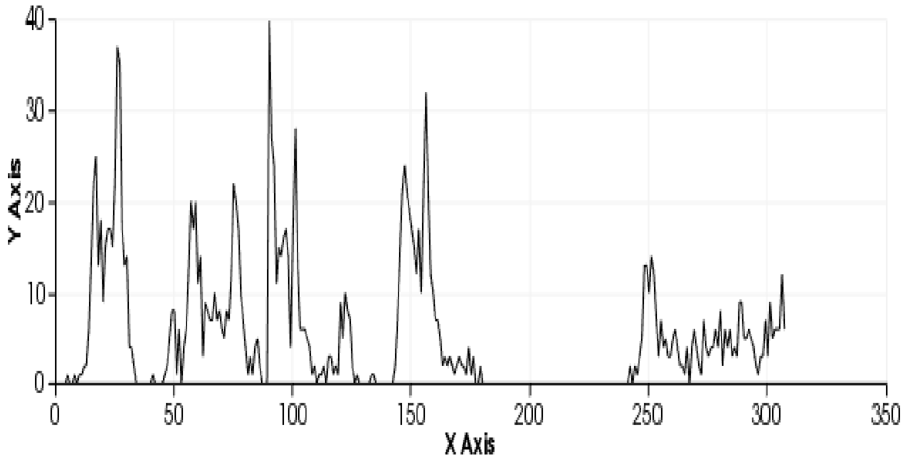


Fig. 5.4.2 CVFH Representation for Cup

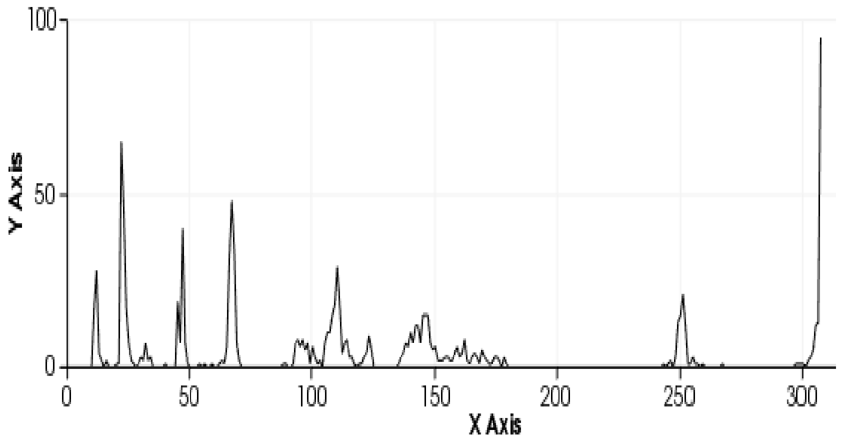


Fig. 5.4.3 CVFH Representation for Box

There is a class in Point cloud library named as *pcl::CVFHEstimation* which was used to compute the descriptor for each stable region and it is also derived from VFH. In order to compute CVFH we first of all have to compute normal of the filtered point cloud and there is also a class for this purpose named as *pcl::NormalEstimation* in Point cloud library which was also used.

5.5 K-dimensional (Kd) Tree Representation:

It is a data structure which is used to arrange points with *K* dimensions. This technique is very useful and extensively used in order to search nearest neighbors or one can use it to find all neighbors in a specified radius too. Since in point cloud there are normally three dimensions we are most concerned about so this is why our K-d tree will be three dimensional.

There is a class named as *pcl::Kdtree* in Point cloud library which was used and using point cloud library we can create K-d tree very simply and in a very fast way. After performing aforementioned steps, we were available with CVFH features for all the desired objects.

Chapter 6: Testing

6.1 Overview:

The algorithm choosed to achieve the object is shown in Figure 3.1.2. First of all, a point cloud is captured using the Kinect v2 sensor of the scene. Since the point cloud contains a lot of points data which make it very complex as far as computation is concerned so initially all the undesired points in the scene were removed. In order to do this first a Passthrough filter was used and then Voxel Grid filter was used.

There is a class in Point Cloud Library named as *pcl::PassThrough* which was used to remove all the undesired points lying along x, y and z axis other than a certain desired range along each axis. This filter significantly reduced the total number of points in the cloud and hence improved the processing speed to a substantial extent.

There is also a class named as *pcl::VoxelGrid* which is also used for the same purpose as mentioned above and it also reduced the total number of points in the point cloud retaining the shape of original objects.

One more important thing is that the parameters for the both filters were kept same as the parameters while training our system. If the parameters were made different then the recognition system will not be able to recognize objects due to difference in resolution.

After performing filtration, the next step is to extract clusters using the class *pcl:EuclideanClusterExtraction* using the same way as we did while building model database. Here threshold value for clustering was same however minimum and maximum size of clusters was variable. Since the object in the scene could be different so these factors were can't be made constant or same as that of while building model database.

After computation of clusters next task was to calculate CVFH feature for each cluster. For each cluster's CVFH feature the algorithm tried to find the nearest object match using the Euclidean distance method in the model database. If a match was found, then it checks weather the distance is less than certain threshold set by user (50 in Our case). If it is less than certain threshold, then object is said to be recognized.

However, if a match was not found then the algorithm will consider another cluster and will compute CVFH for this cluster and will find its match and the process will be repeated for all the clusters in the frame 1 until the object is recognized. Then after this again new scene is captured and for the scene all aforementioned steps are repeated

Chapter 7: Experimental Results

7.1 Experiment No.1

A bottle was placed before Kinect V2 Sensor at a fixed distance and its point cloud image was retrieved as shown in Figure 7.1.1. Its cluster consists of 562 data points and the algorithm returns two nearest matches found in the database as it can be seen in the Figure 7.1.2.

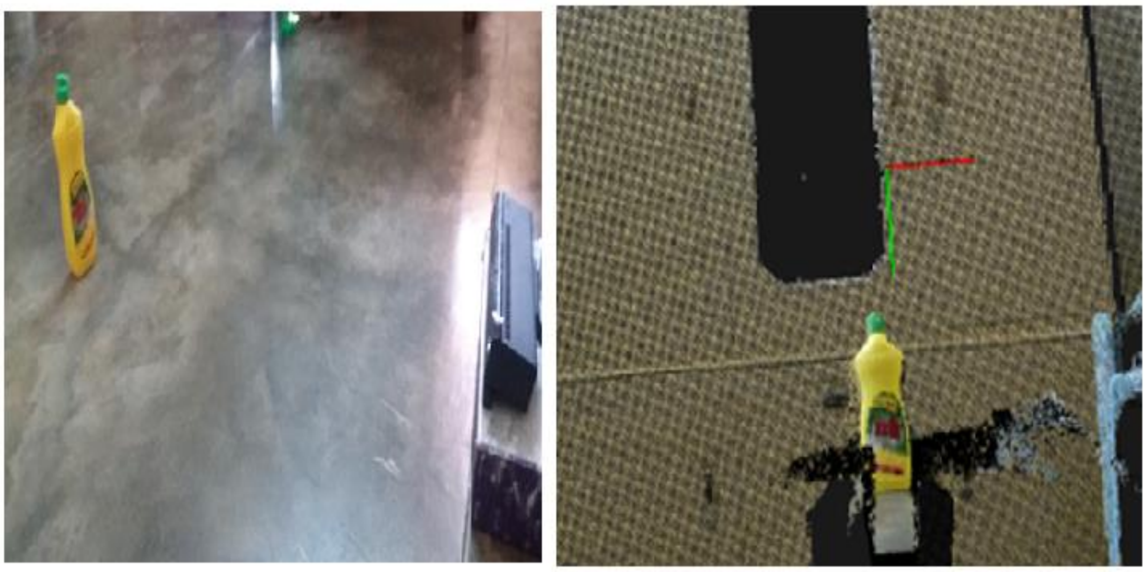
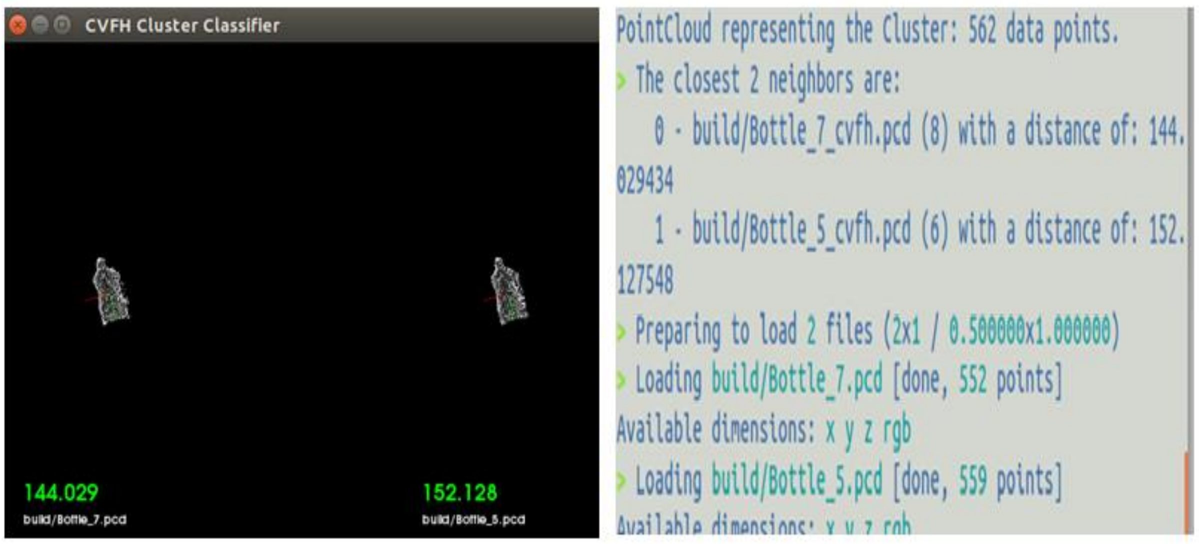


Fig. 7.1.1 Actual scene and its Point cloud (Bottle)



```
PointCloud representing the Cluster: 562 data points.
> The closest 2 neighbors are:
    0 - build/Bottle_7_cvfh.pcd (8) with a distance of: 144.
029434
    1 - build/Bottle_5_cvfh.pcd (6) with a distance of: 152.
127548
> Preparing to load 2 files (2x1 / 0.500000x1.000000)
> Loading build/Bottle_7.pcd [done, 552 points]
Available dimensions: x y z rgb
> Loading build/Bottle_5.pcd [done, 559 points]
Available dimensions: x y z rgb
```

Fig. 7.1.2 Output (Bottle)

7.2 Experiment No.2

A glass was placed before Kinect V2 Sensor at a fixed distance and its point cloud image was retrieved as shown in Figure 7.2.1. Its cluster consists of 289 data points and the algorithm returns two nearest matches found in the database as it can be seen in the Figure 7.2.2.

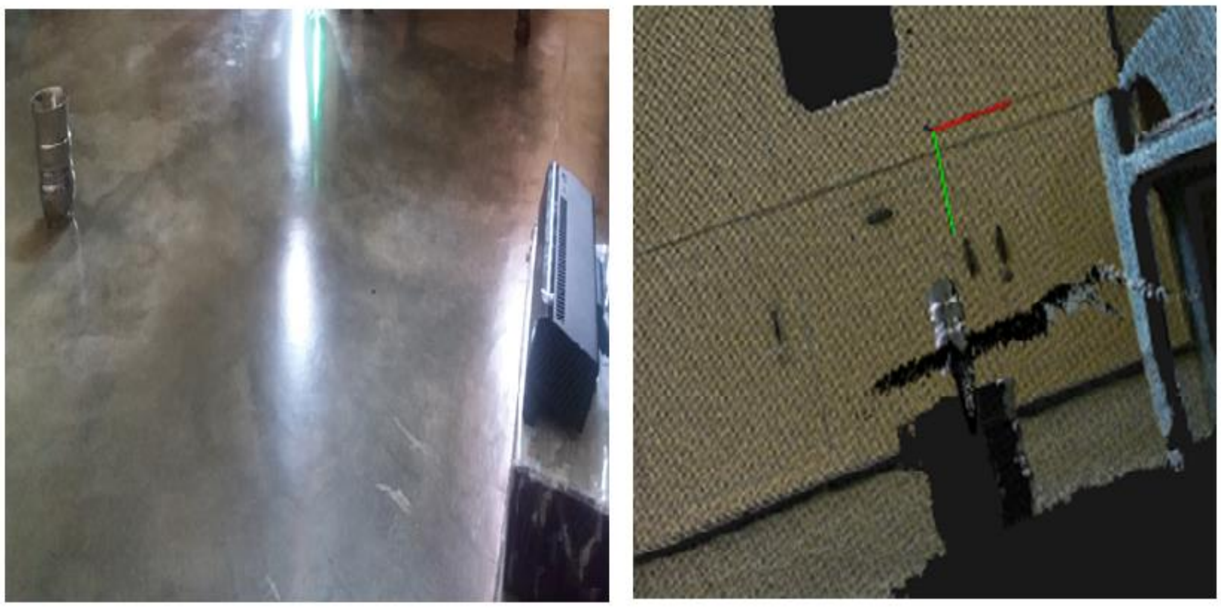
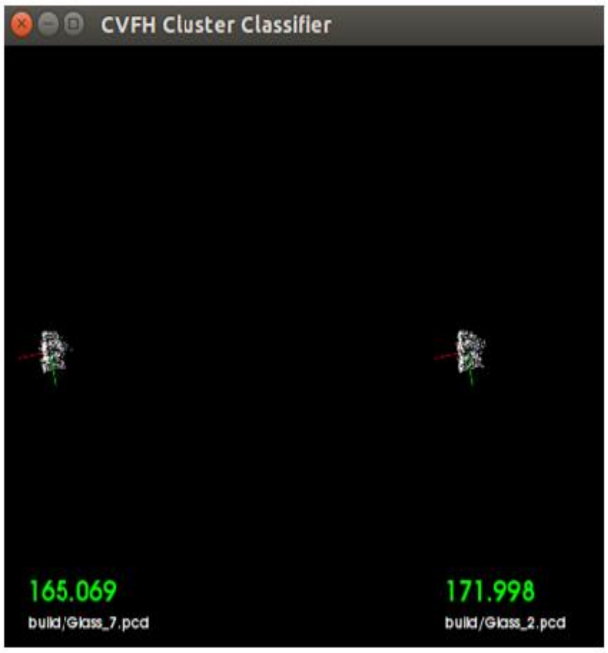
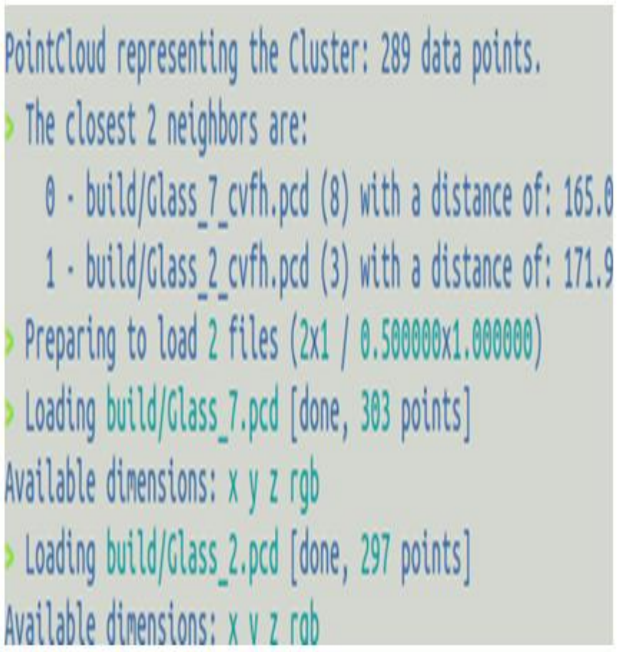


Fig. 7.2.1 Actual scene and its Point cloud (Glass)





```
PointCloud representing the Cluster: 289 data points.
> The closest 2 neighbors are:
    0 - build/Glass_7_cvfh.pcd (8) with a distance of: 165.0
    1 - build/Glass_2_cvfh.pcd (3) with a distance of: 171.9
> Preparing to load 2 files (2x1 / 0.500000x1.000000)
> Loading build/Glass_7.pcd [done, 303 points]
Available dimensions: x y z rgb
> Loading build/Glass_2.pcd [done, 297 points]
Available dimensions: x y z rgb
```

Fig. 7.2.2 Output (Glass)

7.3 Experiment No.3

A box was placed before Kinect V2 Sensor at a fixed distance and its point cloud image was retrieved as shown in Figure 7.3.1. Its cluster consists of 178 data points and the algorithm returns two nearest matches found in the database as it can be seen in the Figure 7.3.2.

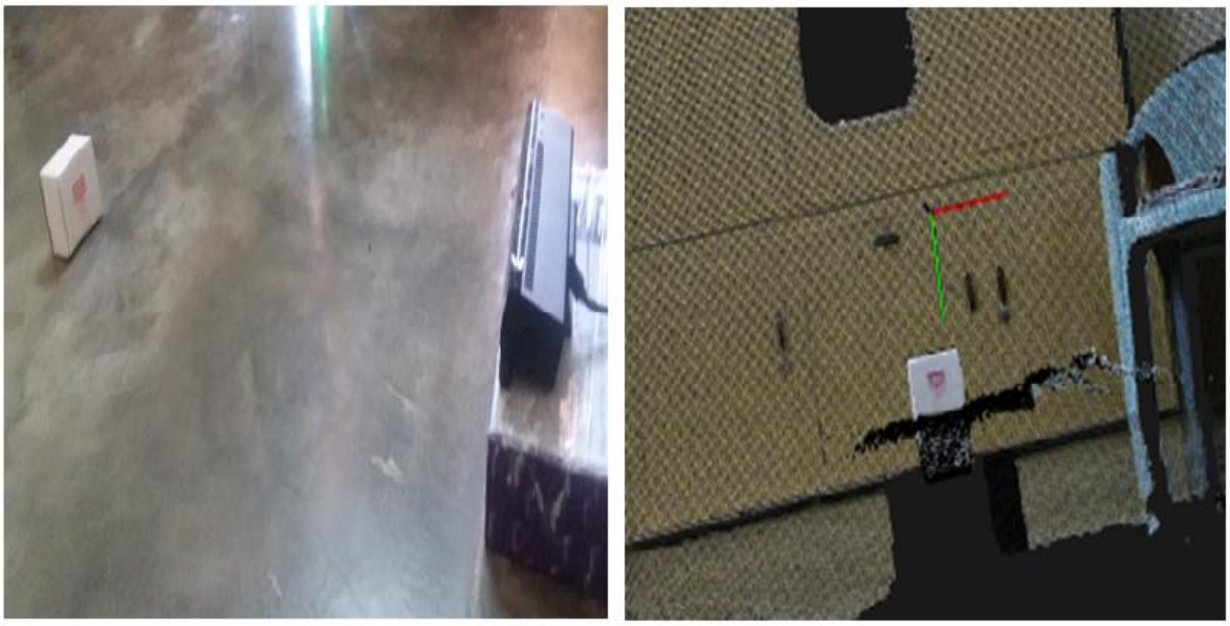
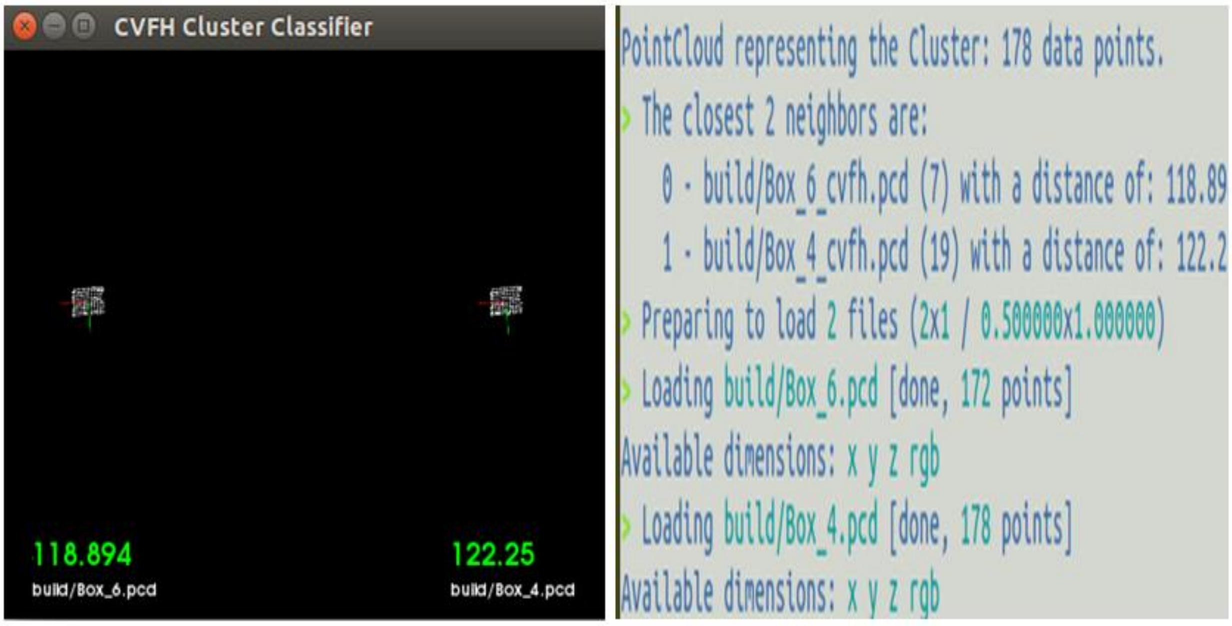


Fig. 7.3.1 Actual scene and its Point cloud (Box)



```
PointCloud representing the Cluster: 178 data points.
> The closest 2 neighbors are:
    0 - build/Box_6_cvfh.pcd (7) with a distance of: 118.89
    1 - build/Box_4_cvfh.pcd (19) with a distance of: 122.2
> Preparing to load 2 files (2x1 / 0.500000x1.000000)
> Loading build/Box_6.pcd [done, 172 points]
Available dimensions: x y z rgb
> Loading build/Box_4.pcd [done, 178 points]
Available dimensions: x y z rgb
```

Fig. 7.3.2 Output (Box)

7.4 Experiment No.4

A bottle was placed with some other objects to create partial occlusion, before Kinect V2 Sensor at a fixed distance and its point cloud image was retrieved as shown in Figure 7.4.1. Its cluster consists of 579 data points and the algorithm returns two nearest matches found in the database as it can be seen in the Figure 7.4.2. Although the object was recognized correctly however threshold distance was greater than the specified.

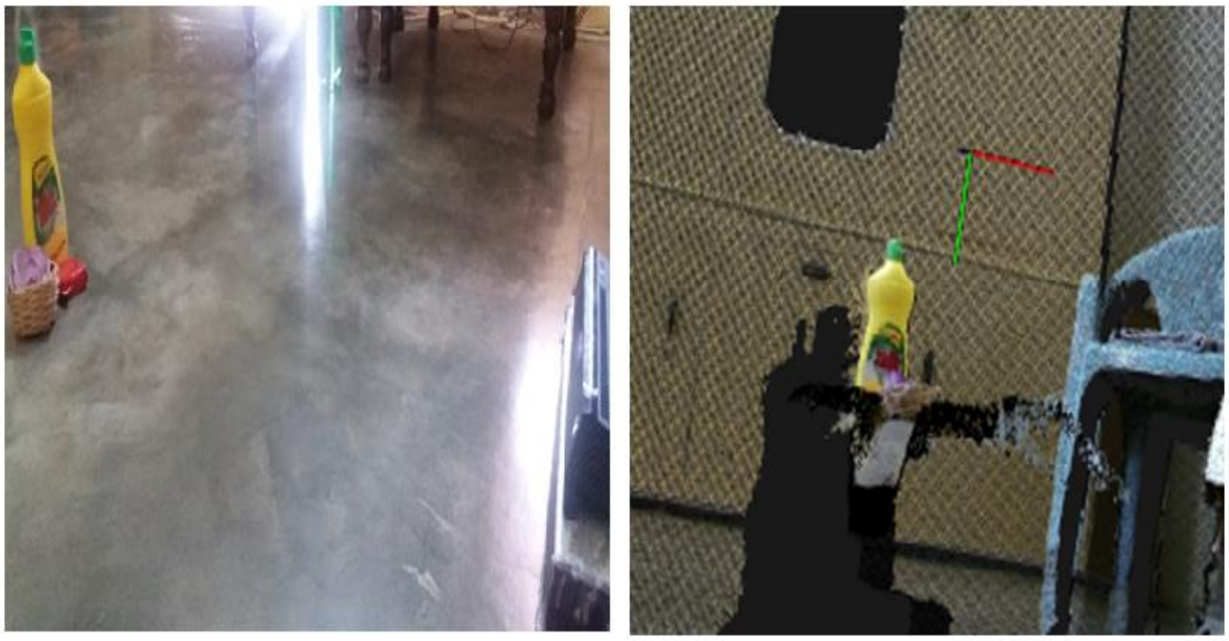
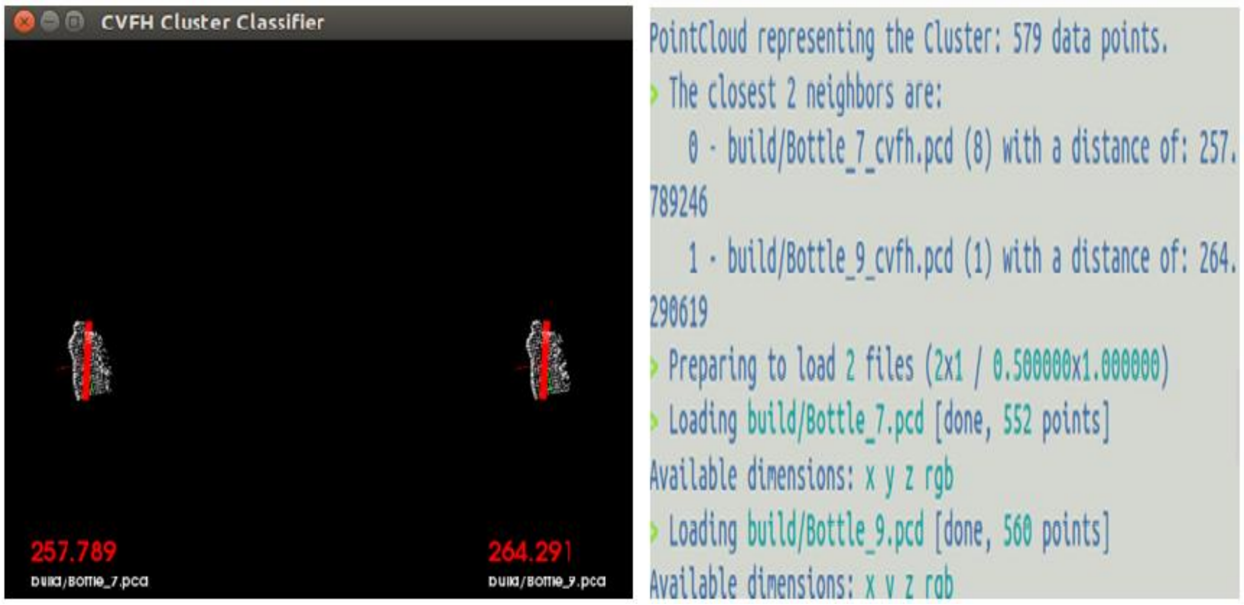


Fig. 7.4.1 Actual Occluded scene and its Point cloud (Bottle)



```
PointCloud representing the Cluster: 579 data points.
> The closest 2 neighbors are:
    0 - build/Bottle_7_cvfh.pcd (8) with a distance of: 257.
789246
    1 - build/Bottle_9_cvfh.pcd (1) with a distance of: 264.
290619
> Preparing to load 2 files (2x1 / 0.500000x1.000000)
> Loading build/Bottle_7.pcd [done, 552 points]
Available dimensions: x y z rgb
> Loading build/Bottle_9.pcd [done, 560 points]
Available dimensions: x y z rgb
```

Fig. 7.4.2 Output (Bottle)

7.5 Experiment No.5

A glass was placed with an another object to create partial occlusion, before Kinect V2 Sensor at a fixed distance and its point cloud image was retrieved as shown in Figure 7.5.1. Its cluster consists of 165 data points and the algorithm returns two nearest matches found in the database as it can be seen in the Figure 7.5.2. Although the object was recognized correctly however threshold distance was greater than the specified.

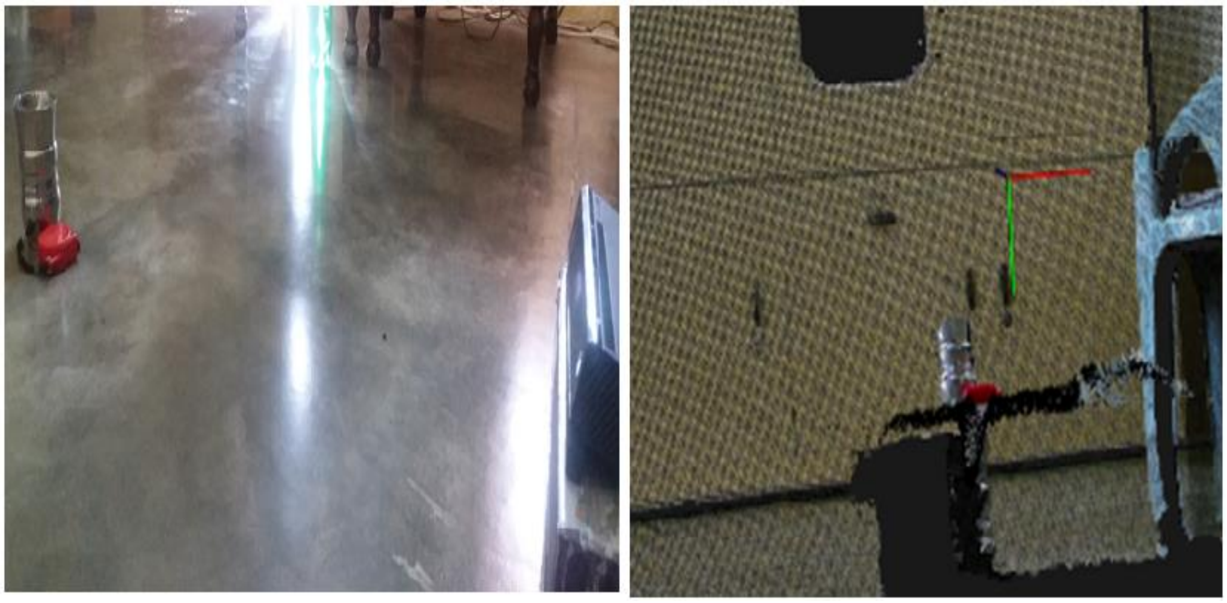
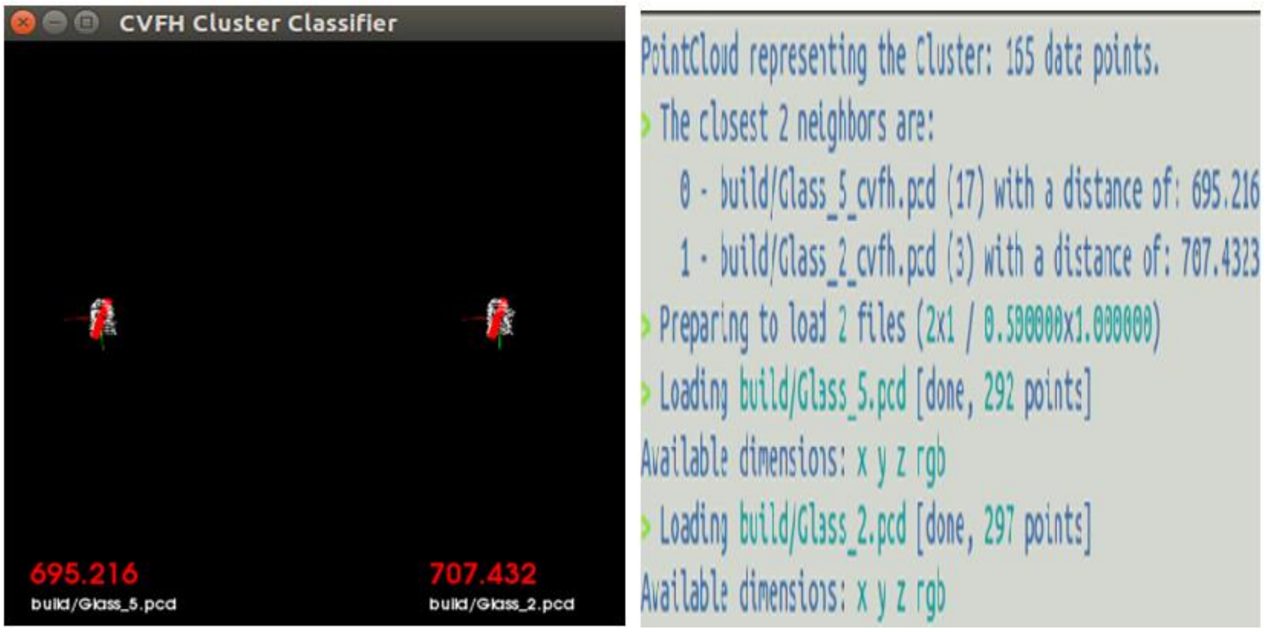


Fig. 7.5.1 Actual Occluded scene and its Point cloud (Glass)



```
PointCloud representing the Cluster: 165 data points.
> The closest 2 neighbors are:
    0 - build/Glass_5_cvfh.pcd (17) with a distance of: 695.216
    1 - build/Glass_2_cvfh.pcd (3) with a distance of: 707.4323
> Preparing to load 2 files (2x1 / 0.500000x1.000000)
> Loading build/Glass_5.pcd [done, 292 points]
Available dimensions: x y z rgb
> Loading build/Glass_2.pcd [done, 297 points]
Available dimensions: x y z rgb
```

Fig. 7.5.2 Output (Glass)

Chapter 8: Conclusion

8.1 Conclusion:

Using depth sensor Kinect V2, scene was captured successfully. There were many undesired points in the form of undesired objects. In order to remove that NaN filter, Passthrough Filter and Voxel Grid filter was used.

Once filtration was done then next step performed was to extract clusters using Euclidean segmentation. After segmentation of the filtered point cloud captured using Kinect V2, CVFH features were computed for each cluster of the desired objects and model database was built successfully.

Using K Nearest neighbor method 2 nearest neighbor were found for the desired objects during testing successfully.

In case of partially occluded objects the algorithm also performed well however distance was a bit more than the specified threshold.

References

[1] http://legendtechz.blogspot.com/2013/03/5-explain-process-of-image-acquisition.html

[2] http://pointclouds.org

[3] R. Rusu, N. Blodow, Z. Marton, and M. Beetz, “Aligning point cloud views using persistent feature histograms,” in International Conference on Intelligent Robots and Systems (IROS), Nice, France, September 22-26 2008.

[4] R. Rusu, N. Blodow, and M. Beetz, “Fast point feature histograms (fpfh) for 3d registration,” in International Conference on Robotics and Automation (ICRA). Kobe, Japan: IEEE, May 12-17 2009

[5] F. Tombari, S. Salti, and L. Di Stefano, “Unique signatures of histograms for local surface description,” in Proceedings of the 11th European conference on computer vision conference on Computer vision: Part III. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 356– 369.

[6] A. E. Johnson, “Spin-images: A representation for 3-d surface matching,” Robotics Inst., Carnegie Mellon Univ, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-97-47, Aug. 1997.

[7] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu. “Fast 3d recognition and pose using the viewpoint feature histogram,” In Proceedings of the 23rd IEEE IRSl International Conference on Intelligent Robots and Systems (IROS), Taipei, Taiwan, 10/2010 2010.

[8] A. Aldoma, N. Blodow, D. Gossow, S. Gedikli, R. Rusu, M. Vincze, and G. Bradski, “Cad-model recognition and 6 dof pose estimation,” in ICCV 2011, 3D Representation and Recognition (3dRR11) workshop, 2011.

[9] http://robotica.unileon.es/index.php/File:Stereo.png

[10] http://static.clickbd.com/global/classified/item_img/271191_0_original.jpg

[11] http://robotica.unileon.es/index.php/File:Structured_light.png

[12] http://robotica.unileon.es/index.php/File:LIDAR.jpg

[13] http://www.slideshare.net/tinux/develop-store-apps-with-kinect-for-windows-v2

[14] https://github.com/OpenKinect/libfreenect2

[15] R. B. Rusu and S. Cousins, “3D is here: Point Cloud Library (PCL),” in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China, May 9-13 2011.

[16] W. Wohlkinger and M. Vincze, “Ensemble of shape functions for 3D object classification,” in Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on, December 2011, pp. 2987 –2992.

[17] Marius Muja and David G. Lowe, “Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration,” in International Conference on Computer Vision Theory and Applications (VISAPP'09), 2009.

[18] Johnson, A. E., & Hebert, M. “Surface Matching for Object Recognition in Complex 3D Scenes,” Image and Vision Computing, vol. 16, pp. 635-651, 1998.

[19] D. Huber, A. Kapuria, R. R. Donamukkala, and M. Hebert, “Parts-based 3D object classification,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 04), June 2004.

[20] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz, “Learning Informative Point Classes for the Acquisition of Object Model Maps,” in In Proceedings of the 10th International Conference on Control, Automation, Robotics and Vision (ICARCV), 2008.

[21] R B. Rusu, “Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments,” Ph.D. dissertation, Computer Science department, Technische Universitaet Muenchen, Germany, October, 2009.