Development and Evaluation of a Kinect based Bin-Picking System

DVA 407

Masters Thesis in Intelligent Embedded Systems

School of Innovation, Design and Engineering
Mälardalen University, Västerås

By
Chintan Mishra
Zeeshan Ahmed Khan

Supervisor:
Giacomo Spampinato

Examiner:
Mikael Ekström

Apr 2015

ACKNOWLEDGEMENT

First and foremost, we would like to thank and are whole heartedly grateful to our supervisor Giacomo Spampinato at MDH and Johan Ernlund at Robotdalen for their time, support, faith and patience. It was due to them that we were able to learn and grow not only academically but also as individuals in this project.

We would like to extend a special word of thanks to Fredrik and Batu for their valuable inputs and the experience on the subject. We are humbled by our friends and family for their support. A special thanks to Mikael Ekström for reviewing the work and suggesting important changes.Thesis work is tedious job and requires a lot of time. We are thankful to everyone who was in support and constantly motivating us towards success.

ABSTRACT

In this paper, an attempt has been made to explore the possibility of using Kinect sensor instead of the conventional laser depth sensing camera for object recognition in a bin-picking system. The hardware capabilities and limitations of the Kinect have been analyzed. Several state-of-the-art algorithms were studied, analyzed, tested under different conditions and their results were compared based on their accuracy and reproducibility. An insight has been provided on the future work and potential improvements.

Level:	Advanced Level Master Thesis
Author (s):	Chintan Mishra, Zeeshan Ahmed Khan
Supervisor:	Giacomo Spampinato
Title:	Development and Evaluation of Kinect Based Bin- Picking System
Purpose:	The purpose of this thesis is to investigate, and evaluate the Point Cloud from Kinect camera and develop a Bin-Picking System using Kinect camera
Method:	Evaluation has been done using comparative study of algorithms and testing them practically under different conditions
Value:	This thesis aims to streamline the various algorithms that are currently being used in IT industry, specifically in computer vision. Aims to provide an extremely cheap alternative to bin picking
Keywords:	Bin-Picking, Kinect , PCL, Spin-Image, SHOT, SIFT, RANSAC, Keypoints

ABBREVIATIONS

SHOT:	Signature of Histograms for Local Surface Description
SIFT:	Scale Invariant Feature Transformation
MRPT:	Mobile Robot Programing Toolkit
CAD:	Computer Aided Design
PCL:	Point Cloud Library
RANSAC:	Random Sample Consensus

Table of Contents

1 INTRODUCTION

Bin-Picking is the process of picking up a specific object from a random pile of objects. These objects can be differentiated from each other based on their physical properties. Since the early days of industrialization, sorting objects and placing them in supply lines has been a major part of production. For instance, vibratory separators were used to separate solid objects from scrap. Inertial vibrations made the lighter objects fall farther away from the tray and the heavier objects much closer to the tray which separated the pile of heavier objects from the lighter ones. In some versions, apertures were strategically placed so that the smaller objects could be filtered through. These were simple in concept and design.

However, there was a major disadvantage. The equipment was noisy, prone to mechanical wear and tear and had comparatively high maintenance costs. Besides, they were not intelligent and required more man power, meaning greater cost to company. This system called for an alternative and that was where the idea of replacing it with a vision system was born. Vision systems would provide the platform on which the machines could be made intelligent and processes such as "Pick and Place". It was less noisy and periodic maintenance costs could be reduced. Today, vision based Bin-Picking is used in many industries and especially in the automotive industry.

In its nascent days vision based systems were limited to 2D object recognition. These were simpler to implement but could manipulate the data only at a 2D level. Greater details of the object could not be captured. There were many adjustment requirements before capturing the environment for further image processing. The cameras were highly sensitive to subtle changes in the surroundings

The present day technology uses laser based cameras for object identification. It eliminates the high degree of sensitivity and adds a major feature, "depth data capture". This means that an object and/or the surroundings will be reliable and closer to the actual object properties. The object or the surroundings can now be "described" for different orientations in the 3D viewing platform.

The aim of this work is to try and reach closer to the results of Bin-Picking (by laser) by using the Microsoft Kinect Camera. The single major motivation behind the use of Kinect sensor was the high design and installation costs of laser cameras. Kinect is available at an affordable price of USD 150 which is very cheap when compared to its big brother the laser. Another advantage is its portability and inbuilt hardware such as tilt motors that may have future implementation. On the other hand, Kinect has a lower accuracy and lesser immunity to lighting conditions.

The purpose of this thesis work is to explore the use of known vision algorithms on Kinect. In addition to providing results of these combinations, analysis of different combinations of algorithms will lead to determining their accuracy in identifying the object of interest. Identification is not only linked to software but also aspects such as optimal placement of Kinect and adjusting physical environment in order to minimize disturbances. Algorithms will be explained in detail so that the reader can understand them in simple home environments and potentially continue working on them to take it to the next logical step of bin-picking and path –planning.

1.1 State of the art

Bin picking systems that make use of stereo systems have been widely researched. A study by Rahardja calculated the position and normal vector of randomly stacked parts with the use of a stereo camera which required unique landmark features, which are composed of seed and supporting features to identify the target object and to estimate the pose of the object [48][49].

A study by Wang, Kak, et al. used a 3D range finder directly to compute depth and shape of articulated objects. However, it was subject to occlusion due to low resolution of the range finder [49] [50]. Edge feature extraction techniques have been used for object recognition and pose estimation. It was effective against objects with smooth edges [51]. There are some solutions available in the market such as “Reliabot 3D vision package”, “PalletPicker-3D” and “SCAPE Bin-picker” for simple cylindrical and rotational objects [46].

It is the aim in bin-picking applications to have a general solution to deal with objects of different sizes and shapes. In 2010, Buchholz, WinkelBach and Wahl introduced a variation of RANSAC algorithm called RANSAM. The objects are localized by matching CAD data with a laser scan and have been highlighted in their paper for good performance despite clutter [52].

Since the introduction of Kinect in 2010, it has been at the center of non-gaming applications. The focus of most of the researches till this date has been on human motion analysis [43] and recognition of hand gestures [44] [45].

2 PURPOSE

This thesis is going to investigate the possibility of using a Kinect camera as a “cost effective” replacement to laser camera in bin-Picking. Many state-of-art algorithms are good candidates for laser cameras and have provided very accurate results in the recent past; however their implementation in Kinect is yet to be explored.

2.1 Thesis Structure

This thesis is divided into major sections covering the hardware analysis of the Kinect and the software aspect of the algorithms implemented. The hardware section describes the Kinect camera viewing limitations both with respect to itself and the environment, adjustments and its specification. Software part deals with exploring the appropriate algorithm on the basis of available data from Kinect. Different image processing algorithms and approaches will be combined and their results will be verified.

2.2 Scope of Thesis

This thesis includes evaluation and selection of algorithms for the possible replacement of laser camera by Kinect camera in bin-picking systems. An attempt has been made to study the performance of algorithms that generally have a good performance in laser cameras when used on a Kinect camera.

The set up for using the Kinect camera with respect to factors such as viewing area, lighting conditions and optimal distance from the camera has been discussed. Data capture, feature computation, keypoint extraction, segmentation, clutter and occlusion analysis are in scope of this document. Gripping patterns as well as position estimation are out of the scope of this thesis.

This will contribute in the work towards exploring a low cost alternative to the existing laser camera solution. In addition, an analysis and understanding of the algorithms discussed can be applied to indoor small scale object matching using PCL so that its industrial applications can be explored in future.

3 HARDWARE AND SOFTWARE

3.1 Kinect Camera

Kinect Camera is a motion sensor developed by Microsoft, based on RGB and depth camera. It works on depth sensing based on projection of IR (infrared) pattern on the environment and receiving the reflections from objects in the surroundings. It was developed as a Microsoft XBOX gaming console so that people can play games with the involvement of their full body rather than holding just a controller in hand. Kinect Sensor was launched in November 2010 and soon after its launch, programmers (Computer Vision) started hacking the sensor. Kinect is an extremely cheap product considering the availability of a depth sensor that changes the perspective of vision based recognition. Microsoft has released the Kinect SDK (Software Development Kit). Kinect camera now-a-days is in focus of development in motion sensing and object recognition fields. Some hardware details have been provided below.

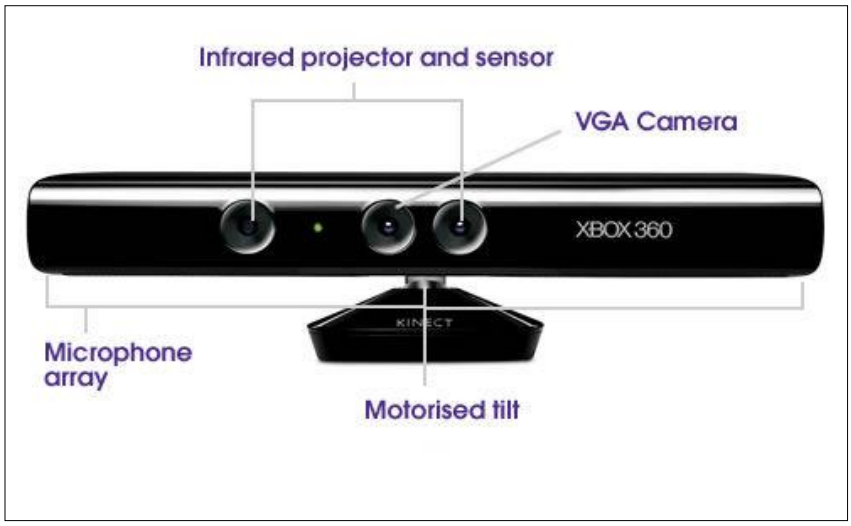

Figure 1: Kinect Camera

- 64 MB DDR2 SDRAM
- Two infrared cameras (CMOS & Color CMOS) and an IR projector.
- Four microphones
- A Prime Sense PS1080-A2 Chip [19]
- A 12-watt fan
- Pan and Tilt motor

A precaution here to observe is that the tilt head of the Kinect should not be tilted by hand. It may cause damage to the sensor [9].

3.2 Kinect Specification

The specifications of Kinect have been presented in many documents online [8] [9]

Property	**Value**
Field of View	57◦ H, 43◦ V
Frame-rate	30 Hz
Spatial Range (VGA)	640 x 480
Spatial Resolution @ 2m distance	3mm
Depth Range	0.8m - 4m
Depth Resolution @ 2m distance	1 cm

Table 1: Kinect Specifications

3.3 Point Cloud Library (PCL)

PCL is a stand-alone library [11], with numerous state-of-art algorithms for object recognition and tracking. The library is divided in several modules such as filters, features, geometry, keypoints etc. Each module is comprised of several algorithms. Pointcloud is a collection of points which refers to group of three dimensional points that is representing the properties of an object in x, y and z axis. There is an option also to represent the object in 3D.

PCL has been released under BSD License; so it is open source and everyone is eligible to participate in its development which is one of the reason that in regard of Robotics and Recognition programmers have put up such a collection of material in it that is useful for them and for others as well. The repository is being updated every day and periodic releases are done.

3.4 OpenNI Driver

PCL (Point Cloud Library) has been used for the development and PCL provide OpenNI Driver for driving the Kinect Camera. OpenNI is an acronym of Open Natural Interface and a very profound name in open-source industry. There are many Application Programing Interface (API) and frameworks in the market from OpenNI [23] mainly for vision and audio sensors. Software programmers and software companies usually prefer API's from OpenNI because of robust rendering, reliability and stability.

Programmers make modules of different functionalities in common use and organize collection of such modules usually known as a library. There are many libraries available such as OpenKinect, depthJS, SensorKinect, NITE etc. Two of them have been considered which are Point Cloud Library (PCL) and Libfreenect due to their application in wide number of projects and good open source support. Motivation behind the selection of OpenNI driver has been provided in section 5.1.

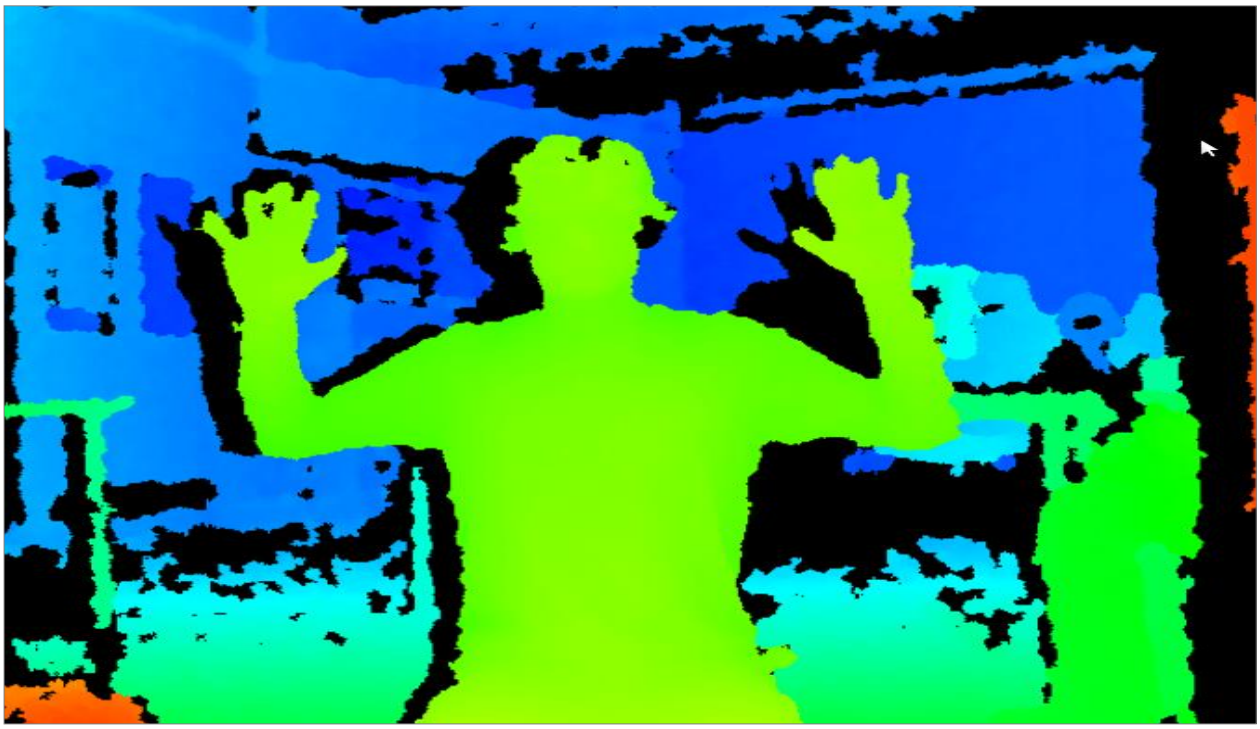

Figure 2: Image with Libfreenect driver

3.5 Software development model

Defining a solution verbally is considerably easy but while developing the solution one usually deals with number of problems. A good software development approach is to divide the problem in smaller chunks such that each chunk can be developed individually and later merged into a bigger solution. A "waterfall" model type approach has been adopted. This approach ensures that the development process is integrated in a sequential manner and provides a checkpoint after every functionality chunk to fall back on.

In this project, major functionalities have been considered as chunks. For instance, filtration has been considered as a major functionality which has been developed independently and later been integrated with the main program.

4 PRINCIPLE OF KINECT CAMERA

4.1 Depth Sensing

The principle behind depth sensing for the kinect is that an IR projector on the camera projects a speckled pattern of IR laser which is captured by an in-built infrared camera sensor. The image processor of Kinect uses the relative positions of the dots in the pattern to calculate the depth displacement at each pixel position in the image [8] [10]. This principle is known as the principle of ***structured light***. The images captured by the camera are compared to a known pattern. Any deviation from it can then determine whether the target is nearer or farther away. Figure 3 gives the actual picture of depth measurement in Kinect. Since the Kinect uses an IR sensor, it will not be working in sunlight

[10]. Besides, reflective and metallic surfaces are responsible for a lot of occlusion and ambiguous information.

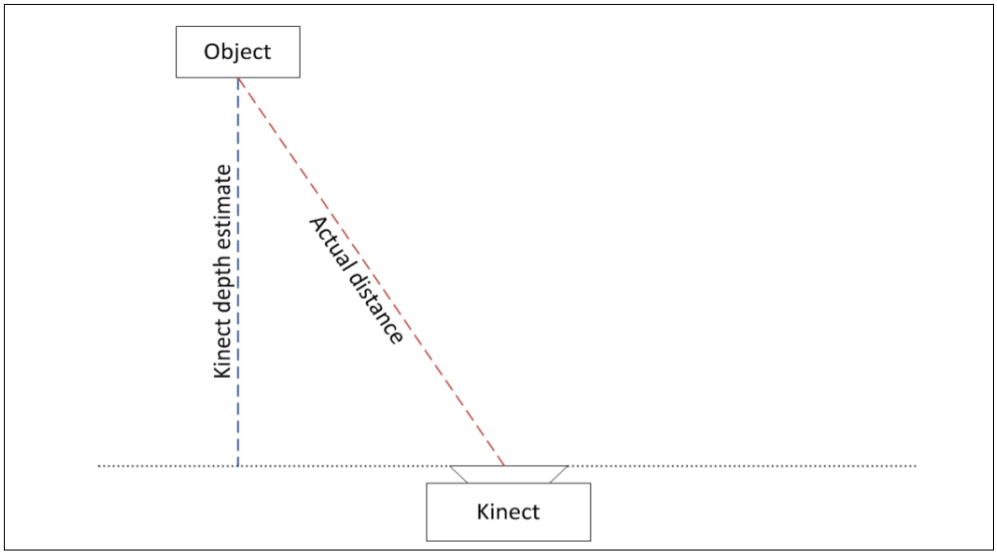

Figure 3: Depth Camera [8]

In figure 3, it can be seen how depth estimate is calculated by the Kinect. The line of actual distance is the reflected IR beam. Based on this value, the depth value is calculated by Kinect. The nominal range accuracy of the laser scanner is 0.7 mm for a highly reflective target at a distance of 10 m to the scanner. The average point spacing of the point cloud on a surface perpendicular to the range direction (and also the optical axis of the infrared camera of Kinect) is 5 mm [31]. This means that for a good recognition, the object should not be placed too far from the camera sensor.

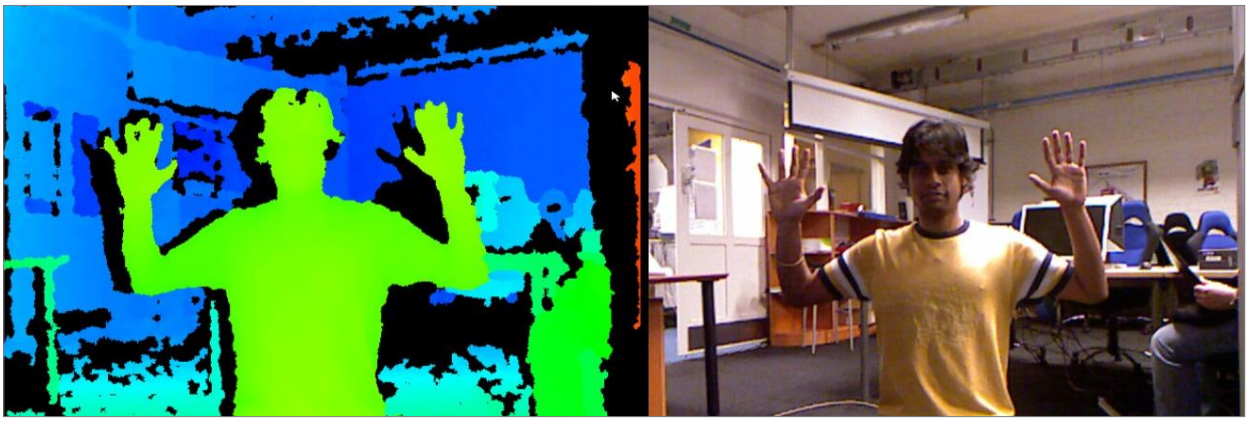

Figure 4: Depth Image

In figure 4, a depth image on the left and corresponding RGB feed to the right is shown. Varying distances are shown in different colors in the depth camera feed (left). It can be seen that the person in the image is closer to the Kinect and is shown in light green color. Similarly, the corresponding depth image background is shown in blue color. There were some ‘imperfections’ observed in the depth image. These imperfections are called clutter and occlusions. Clutter is the presence of noise where there is no real data and occlusion is the absence of data when the data is actually present. For instance,

the index finger, middle finger and ring finger of the left hand are seen cluttered whereas some portions behind the person appear dark due to occlusions. It is important to determine the optimum location of the Kinect for object detection in the bin-picking setup, so that clutter and occlusion is minimized.

4.2 Point Cloud

Kinect consists of a laser source that emits a single beam which is split into multiple beams by a diffraction grating to create a constant pattern of speckles projected onto the scene. This pattern is captured by the infrared camera and is correlated against a reference pattern [10]. The emitter is very powerful and the concept behind splitting it into multiple beams is to facilitate the use of a high power projector beam. This speckled pattern is interpreted by appropriate libraries (Point Cloud Library here) as a set of points known as 'Point Cloud'. Each point on the cloud primarily has three co-ordinates X, Y and Z which are usually called vertices.

Point cloud can be explained with the analogy of a thin sheet of cloth. When objects are covered with this thin sheet, only an outline of their physical attributes such as shape and size can be seen. Sharp lines of distinction between covered objects and the background are not as good as without a sheet. If the figure 5, below is to be taken as an example, its clearly visible that IR Image of Kinect shows a 'thin cloth sheet' analogy. The only difference is that instead of a cloth there are group of points that define an object.

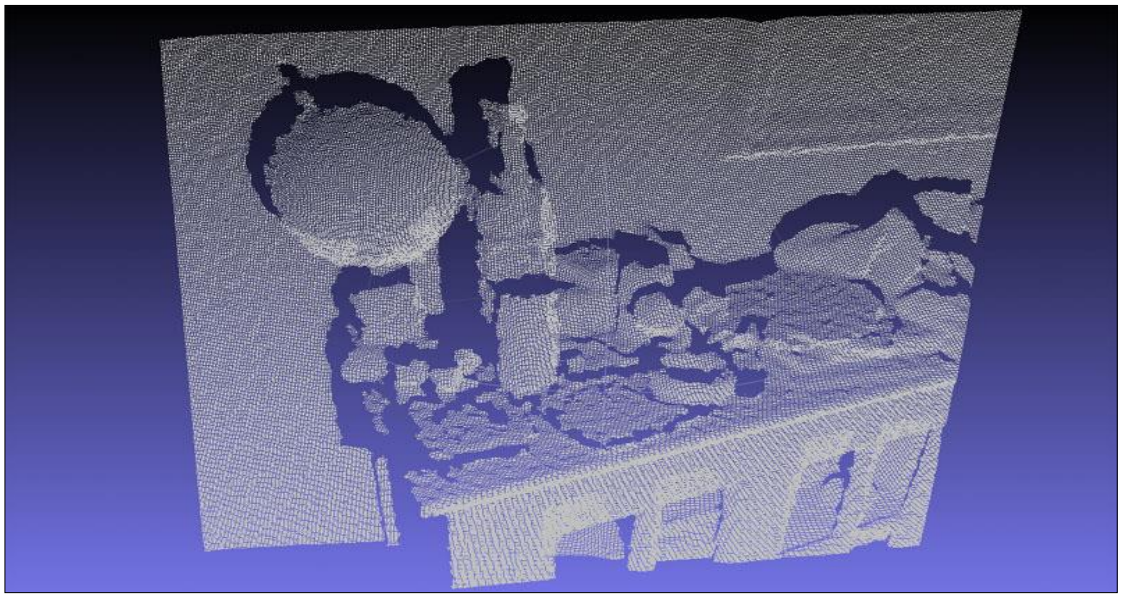

Figure 5: Point Cloud [21]

A usual pointcloud frame, from the Kinect has about 300000 points. Each point gives the orientation of the object/scene in space with respect to a specific viewpoint. It contains the orientation information as Cartesian co-ordinates. The default viewpoint of the camera is positive x-axis being the vector to the right, positive y-axis pointing downwards and positive z-axis being a vector pointing into the plane. Figure 3 shows how Kinect estimates the depth co-ordinate. Kinect works on scattering a stream of points in a range and then its IR camera captures the point cloud by Triangulation method.

A pointcloud can either be *organized* or *unorganized*. An organized pointcloud has an 'image like' organized structure. It comprises of a *width* and a *height* in the pointcloud which specify the number of

rows and the number of columns respectively [32]. In other words, it is arranged like a 2D array in the memory. It is easier to co-relate the points of an organized point cloud to its respective special co-ordinates. On the other hand, an unorganized pointcloud has all points stored as a 1D array.

5 RECOGNITION APPROACH

The open source libraries have predefined classes of many states of art algorithms with several parameters open for users to tame it as per their requirements. PCL has been preferred because of its advanced functionalities. It can handle complex geometric surface structures to be created thanks to a big number of points. The approach to object recognition can be summarized as follows:

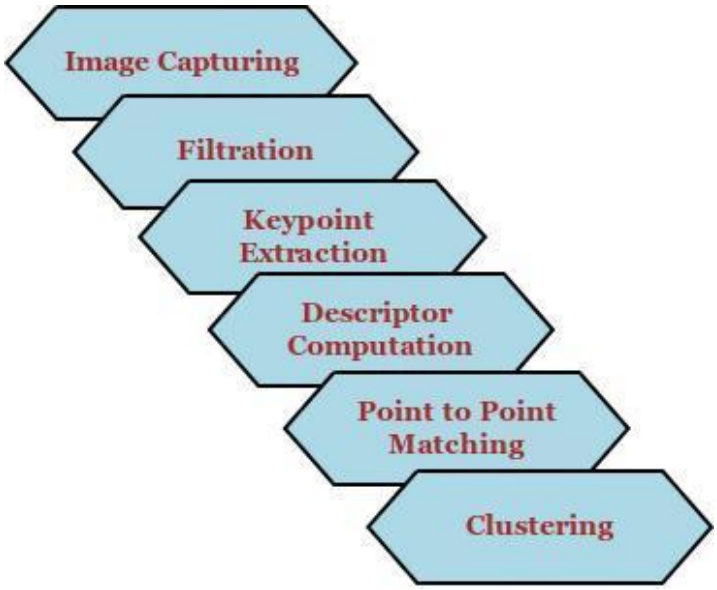

Figure 6: Development approach

The methodology and motivation behind each stage have been highlighted in later sections. A conceptual understanding at each stage has also been discussed so that the reader understands the basic concepts of these widely used algorithms that have been modified for recognition in 3D.

Image Capturing:

The image is captured and stored in a specific format. It has to contain data such as the 3D coordinates with respect to a defined reference axis, color information and/or gradient depending on the algorithm being used.

Filtration:

Filtration of the object under test is done to remove other unwanted objects in background to reduce clutter and occlusion in the captured image. It can be described as the process of "enhancing" an image by changing specific parameters. Unwanted information is neither time nor computation optimal. Besides, it also increases the chances of erroneous comparisons. The focus is to extract specific areas in

the object that would be unique to the object under test.

Keypoint Extraction:

Keypoints can be defined as distinctive features in an object or environment which can be used to perform object matching irrespective of its orientation in a scene. Keypoints reduce the number of points by "downsampling" the points. Simply put, downsampling can be defined as creating a new version of an image with a reduced size (width and height), which in turn takes lesser time to compute descriptors for all downsampled points.

Descriptor Computation:

A descriptor is computed that "describes" the object. For instance, say the given object is a pen which is 10mm in height. Say the pen is the only object among all other objects that has this unique 10mm height. If there existed a descriptor (some imaginary one) that calculates the distance between the top and the bottom vertex of an object, then this height would be how the object would be defined in the language of the descriptor. Hence, when this descriptor computes a value of 10mm, it would mean that the pen has been detected. The descriptors used in this thesis are 3D descriptors.

Point to Point matching:

After computation, the descriptors have been compared and good matches established based on a threshold value. These filtered good matches can then be clustered and classified to be belonging to a certain object instance. Explanation to matching can be found in section 5.5.

The whole process was applied to single isolated objects without clutter and occlusion like objects on a tabletop platform. This approach was then experimented on cluttered environments with similar objects. Different combinations of these algorithms have been tried in order to get the optimal results. Keypoint extractions have been done using Uniform Sampling [1] and SIFT (Scale Invariant Feature Transform) [2]. To describe the object Spin-Image, SHOT [4] and PFH (Point Feature Histogram) [33] have been used.

It is important to understand the concept of normal estimation. Surface normals, as the name suggests are normal vectors perpendicular to a given surface at that point. A bigger surface is generally subdivided into smaller surfaces. Plane tangents to these surfaces are drawn and the normals to these surfaces are computed. The angle between the normals of all these surfaces is a major contributor to determining the properties of the geometric structure and subsequent evaluation of descriptors. Discussing it from PCL perspective, an important input provided by the user to calculate normals is the feature radius. The radius should be selected so that adjacent surfaces are not taken into consideration during normal estimation [34].

5.1 Image capturing

It is important to store the image for the model and the scene in a specific format. Images are stored in PCD (Point Cloud Data) format. PCD files consist of a header that identify and declare the properties of that format. A header contains entries such as version, fields, size, type, count, width, height, viewpoint, points and data. Image is captured in 3D containing information along X, Y and Z axes according to world co-ordinate system or camera co-ordinate system.

Additional information such as RGB information can also be captured. In this project, XYZRGB format has been used to capture the information of the scene. Users of PCL are given the choice of storing images in their own custom format. Count specifies the number of elements in each dimension. This feature is to allocate the 'count' number of contagious memory locations. A field called 'Data' specifies whether the type of data is ASCII or binary.

The obtained pointcloud can be stored as an organized or an unorganized pointcloud. An unorganized pointcloud means that all points on the pointcloud are stored in a 1D array i.e. the number of rows is one and the number of columns equal the total number of points in the input cloud. In an organized pointcloud the number of rows (height) and columns (width) are generally configured as 640 and 480 respectively. An advantage of organized pointcloud is that a mapping between depth pixel data and the corresponding RGB image structure (640 x 480) can be established after calibration of RGB and depth cameras. If D_{m*n} is a 2D depth image array and C_{m*n} is a 2D RGB image array, then an element D_{ij} on the depth image matrix would have the corresponding RGB information in C_{ij} matrix that is the RGB image matrix.

5.1.1 Creating a reference model

An object/s has to be defined in terms of different parameters such as color, orientation in space and their shape. These definitions of the given object are stored in the computer which is then used as a reference against other objects. The closer an object is to the properties of the stored object, the higher chance that both objects are the same. An object to be identified has to be compared against a reference model/s of object/s stored in the memory of the computer. The reference object can be matched to several objects in a scene or a single entity in the scene can be matched to a database of reference models. A reference model is supposed to be detailed and contain all the information regarding the object to be detected in a scene. The goal is to retrieve the smallest available information about the object in the scene and search for a match in the reference model or object database. Since the reference model is detailed and contains information in all conceivable possibilities, there would be a higher probability of a good match and detection.

To create a model, a software tool called “Wings 3D” was selected. In the beginning, a simple cuboid model was constructed. The cuboid had to be converted into a pointcloud so that it could be stored and compared with a similar cuboid in the scene. This required the use of another useful tool called “Meshlab”. There exists an option to convert *.obj* files to pointcloud in Meshlab.

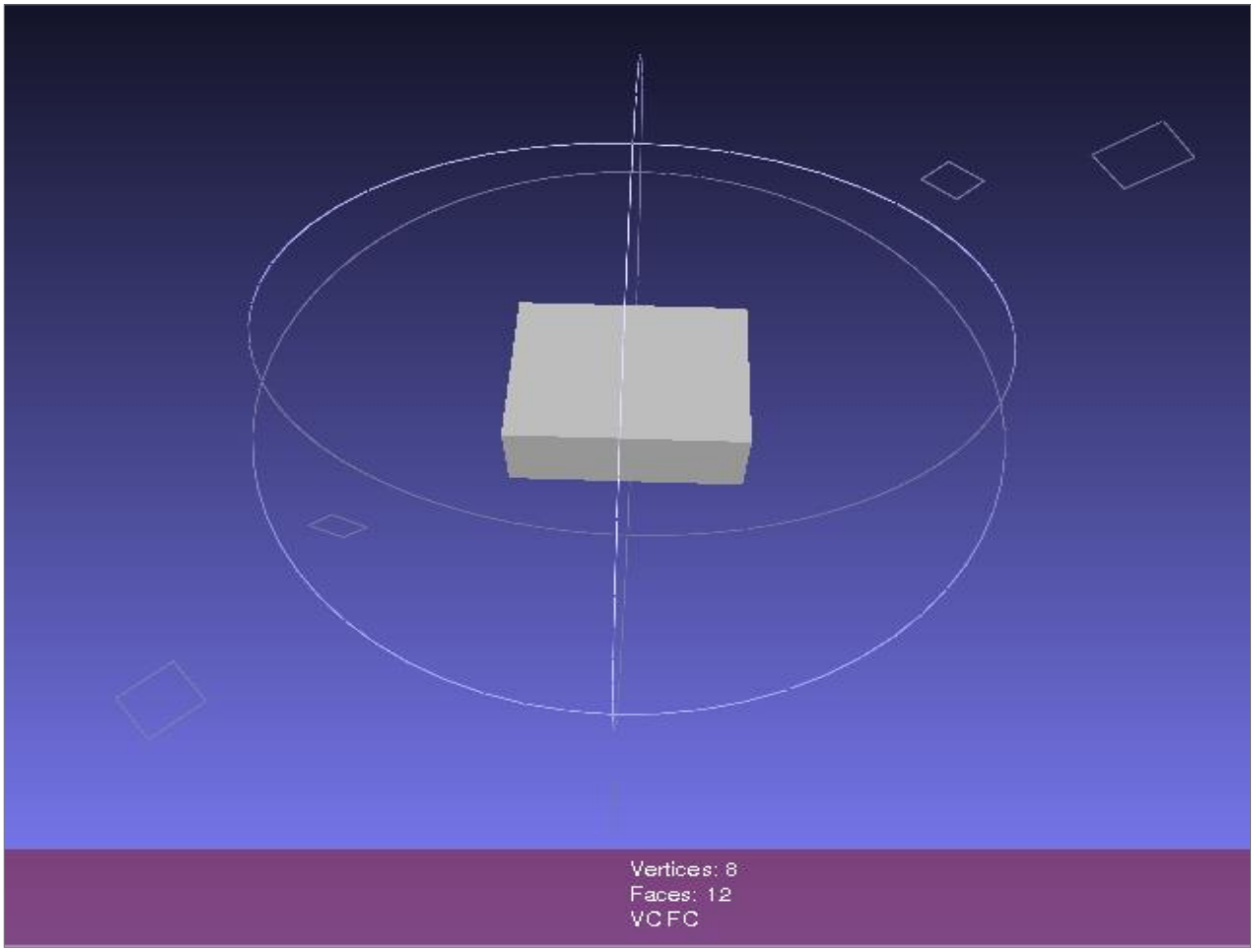

Figure 7: Cuboid model in Meshlab

An option to add surfaces to the *.obj* file was selected to obtain a complete object. The model cuboid was then converted to its corresponding pointcloud format. A screenshot of the cuboid has been shown in figure 7. Total number of vertices and faces has also been provided by Meshlab. The object could be rotated in 3D and information about the frames per second and angle of view were also provided as default.

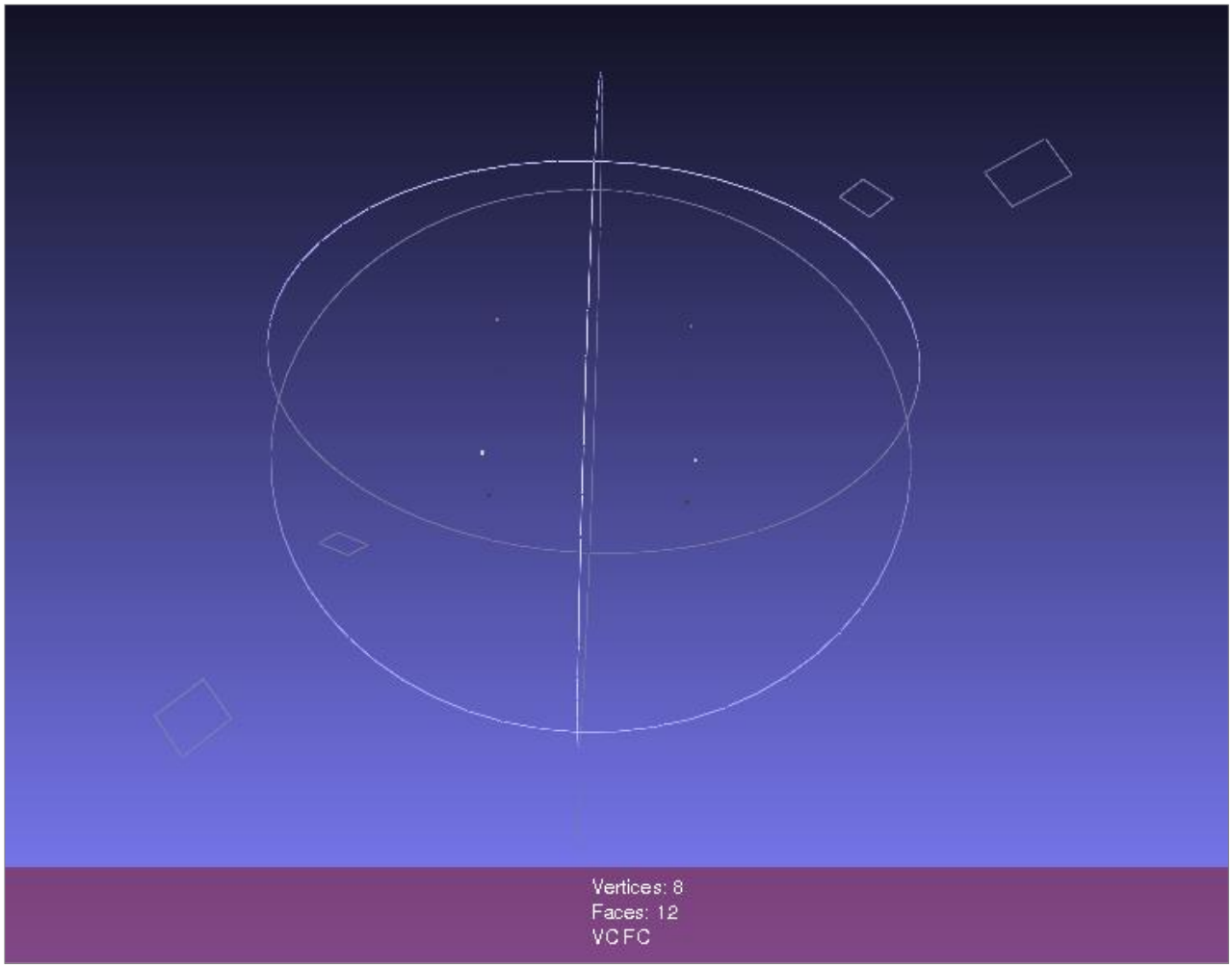

Figure 8: Pointcloud of cuboid model

The screenshot of the pointcloud model of a cuboid has been shown in figure 8. The pointcloud shown has only 8 points. It was seen that only the vertices of a given model were converted to the pointcloud and not the entire surface. It was difficult to obtain a good model with a few numbers of points. Hence, this approach did not give us the expected results.

The next approach was to obtain a CAD model from Solidworks and check if it could be converted to a pointcloud. A *.stl* file made in Solidworks was obtained for the test object. An object file to pcd converter program (obj2pcd) was provided in PCL. Hence this *.stl* file was converted to an *.obj* using Meshlab which later was given as input to the PCL program. Pointcloud files have to be converted into ASCII format in order to be visualized in Meshlab.

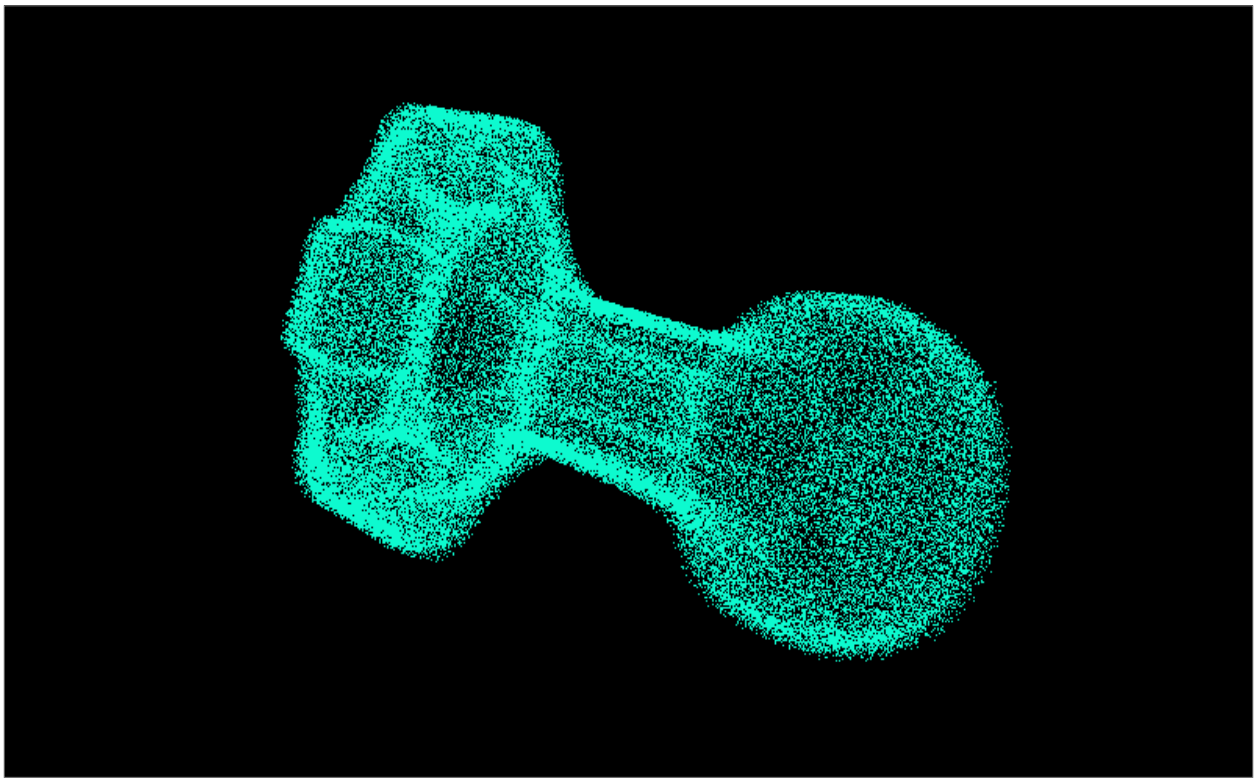

Figure 9: Pointcloud of the CAD model

A pointcloud of the test object was taken as the reference model to be compared with a scene pointcloud captured from the Kinect. The comparison techniques have been discussed in detail later in this report. The reference model did not show any match to the scene from Kinect. A little investigation led to the following conclusions:

I) After comparing coordinates of CAD model and Kinect scene model, it was found that the scale of the CAD model was different from Kinect scale.

II) The resolution of the pointcloud was much higher than the one obtained from Kinect. For comparison, it is desirable to have resolutions that are same or close.

The CAD model was scaled down in the X and Y coordinate by dividing each point by a factor of 1000. This scaled the model down to the Kinect scale of meters (The actual factor was 0.000966 but keypoints could be used to adjust these minor changes). The Z coordinate should not be changed; otherwise the shape of the model will be lost.

There are three parameters in the software tool to alter the resolution of the pointcloud: leaf size, resolution and level. Different combinations were tried but there was no appreciable match between the CAD model and the Kinect scene model.

In order to create a reference model, a pcd image of the object under test was captured with Kinect. Some reconstruction was applied to boost up the number of points and reduce the noise.

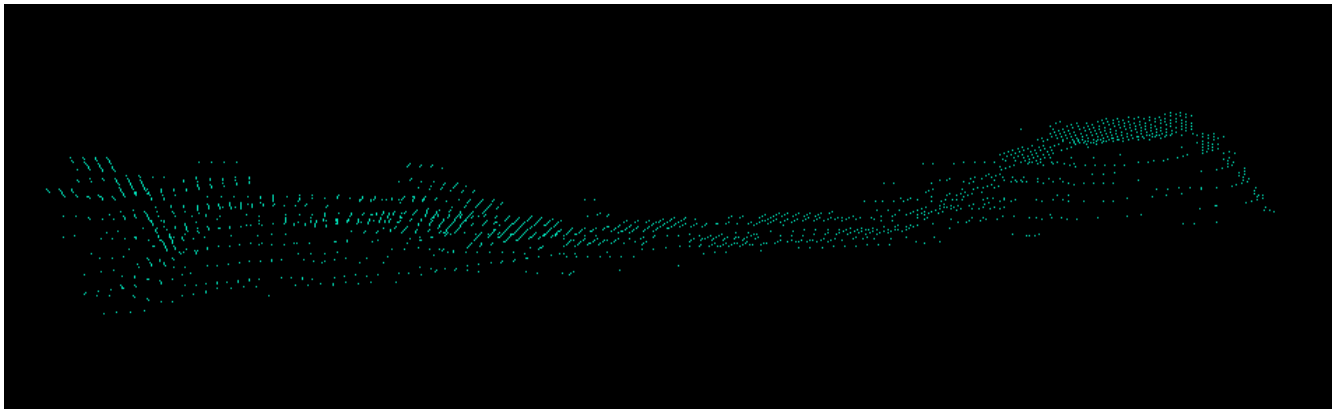

Figure 10: Pointcloud of the Kinect model before MLS

PCL library provides the implementation of an algorithm called Moving least squares (MLS) [29]. The output of the model after implementing MLS is shown in figure 11. MLS is a method for smoothing and interpolating scattered data [39], here it has been used to adjust the points in the cloud such that they can best fit the model. It is a modification of weighted least square (WLS) method. In order to understand WLS, least squares (LS) approximation technique has to be understood. The LS method minimizes the sum of squared residuals. A residual is similar to an error, only difference being that an error is difference between the observed value at a point and average value of all the points, whereas a residual is the error obtained by difference between the observed value and the average value of a "subset of random points" in the cloud. The optimum value will be when the total sum of the square of residuals is a minimum. In MLS, weighted least square is formulated for an arbitrary point for the points around it in a certain radius and the best fit is determined "locally" within the radius rather than globally as in WLS. The approximation is done by dividing the entire surface into smaller polynomials than solving for a large single system. The idea is to start with a weighted least squares formulation for an arbitrary fixed point in the pointcloud and then move this point over the entire parameter domain, in which a WLS fit is computed and evaluated for each point individually[40][41][42].

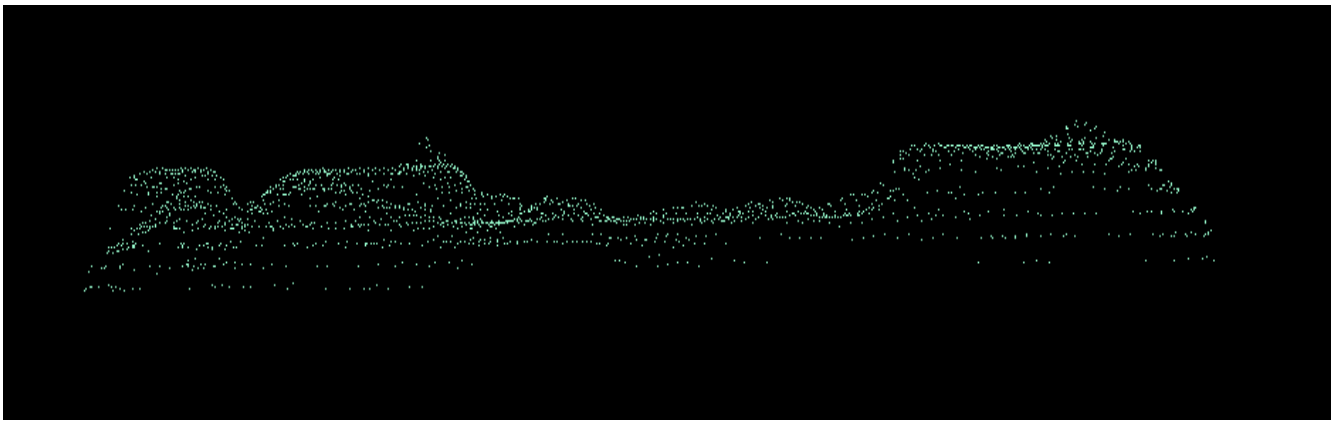

Figure 11: Pointcloud of the Kinect model after MLS

Though there was a marked improvement in the quality of the model, the model could not be matched with a different instance of the same object in a different orientation captured with Kinect.

Another idea was to incrementally register a pair of pointclouds to obtain a better model. In other

words, the model was captured in different orientations from a fixed viewpoint and resulting pointcloud images were stored. These models were aligned to the first cloud's frame and the regions that were occluded in the first cloud's frame could be seen in other clouds.

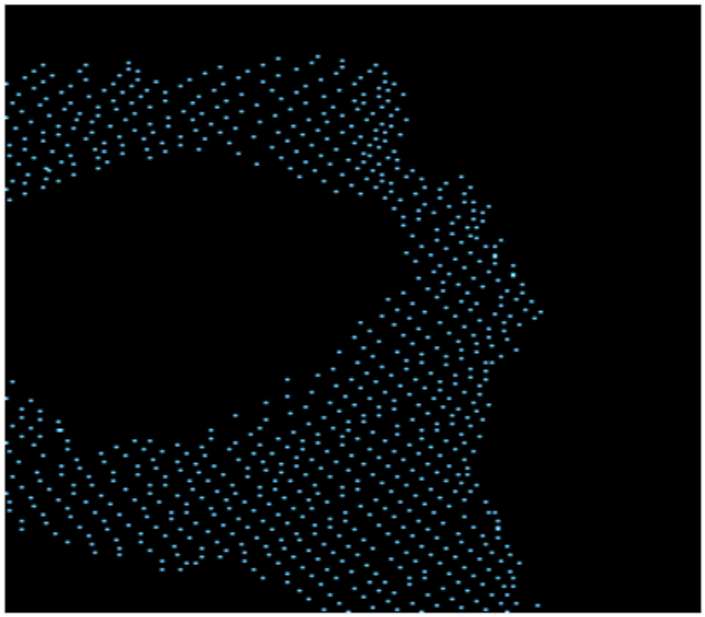

Figure 12: Pointcloud of the model before ICP

The algorithm used for this purpose was **Iterative Closest Point** (ICP). The given reference model (captured from Kinect and filtered) and the sensed shape representing a major chunk of the data shape can be aligned by translating the scene cloud to the reference cloud. A fitness score is given based on the local minimum of the mean square distances between translations a relatively small set of rotations between the reference model cloud and scene cloud [30]. If the score was close to zero, it meant that the object cloud was a good fit to the reference and was included in the registration. This procedure was repeated for a single object in the scene placed in different orientations.

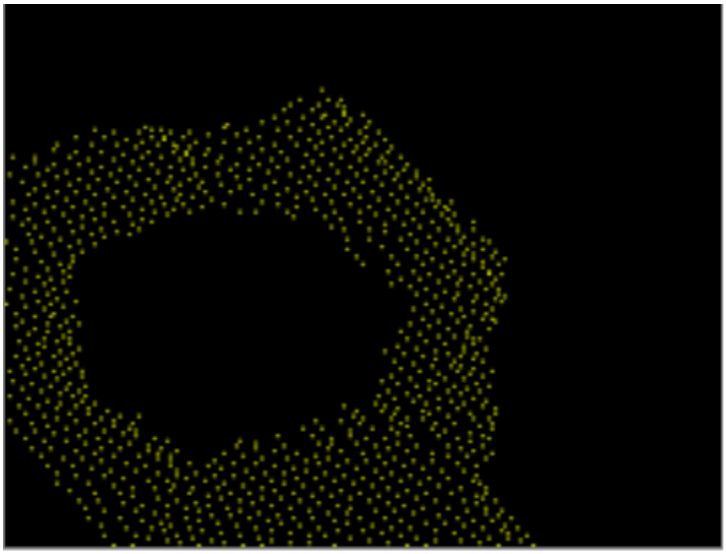

Figure 13: Pointcloud of the model after ICP

Comparing figures 12 and 13, it can be seen that the occlusion to the right has been compensated after ICP. In previous attempts, the model could not be compared to the scene at all. The ICP model served as a reasonable model because firstly, the model could be compared with the scene and secondly, it showed some good correspondences in a scene with similar objects. The descriptor parameters for comparison had to be trimmed for better results but this served as a good starting point.

5.1.2 Optimal placement of Kinect

It was visible earlier (figure 2) that the camera feed was not perfect. A limitation of Kinect with respect to distance from the Kinect has been illustrated. Some drawbacks with respect to the kind of objects that can be detected have also been shown. In figure 2, the left hand of the person in the picture is not clear and the fingers appear scarily webbed because of some noise which had corrupted the actual data. There are some regions in the picture that were observed to be blacked out. Some of these regions were either metallic or reflective. In some cases say in regions below the left arm, there was information but was not shown in the feed. This served as a strong motivation to check the depth information sensitivity of the Kinect. The images were subject to clutter and occlusions.

In simple terms, clutter can be defined as the presence of noisy data where there is no data present and an occlusion is when there is data actually present but is not detected by the camera sensor. The observations with clutter and occlusion set us on the path to minimize the clutter and occlusions. Hence, the first thing was to eliminate unwanted objects in the environment that would contribute to the noise. Putting the Kinect face down eliminated most of the objects in the surroundings; when the Kinect is facing the floor, only the floor and the objects placed on it were possible to be given as camera feed which minimized the noise. The viewing window of the Kinect would be uniform, hence leading to simpler segmentation and filtration algorithms.

The next step was determining the optimal distance to place the object for best possible detection. Setting the position of the Kinect was tricky because on one hand it should not be too near to the bin since it would hinder the movement of the pick and place robot and on the other hand, placing it too far away would clutter away the details of the object. It was observed that the information carried by the object reduces as a function if distance from the sensor. If the object is farther away, the neighboring IR projector particles would be farther away and could corrupt the actual information of the object with noise. Hence, closer the object, better the resolution. The object in consideration which had to be identified also had a hole of around 0.04 m in diameter.

The depth information sensitivity of the Kinect was analyzed from Openni (PCL) and libfreenect Kinect drivers. Libfreenect driver was the starting point to observe the depth sensitivity of the Kinect. There was significant color variations in the live depth image feed for **depths > 1.5 cm** in our observations.

The first step in determining the sensitivity of Kinect sensor was to check the farthest possible distance it could detect an orifice. Generally, there is an increase in clutter with an increase in distance for an object sensed by the Kinect (away from the Kinect). The test object had detailed features and a hole. An orifice if detected will continue to lose information as it is moved farther away from the Kinect. There will be clutter and occlusions added to it.

A monkey wrench was used for the purpose. Monkey wrenches have adjustable jaws. Measurements were made for a maximum aperture size (distance between the adjustable jaws) of 10-11mm. The measurements were started with 1mm as minimum aperture size The results shown below are the mean value of a number of measurements for a given aperture size. The following results were obtained for Libfreenect drivers:

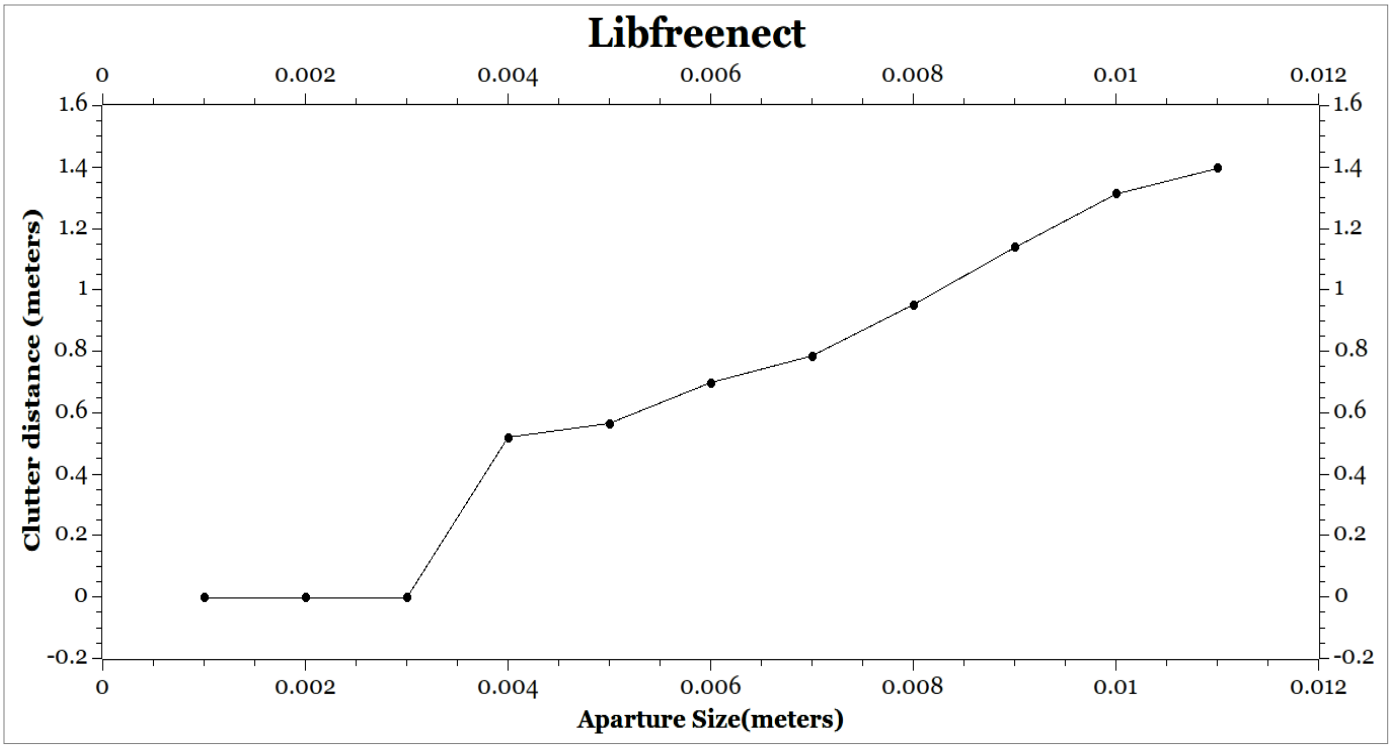

Figure 14: Clutter graph for Libfreenect driver

The aperture size was kept at 1 mm and it was observed that the Kinect did not detect it and cluttered the 1mm gap with noise. Next the aperture size was increased to 2mm and still there was no detection and was shown cluttered in the camera feed. As the aperture size was increased, it was seen that at 4mm there was detection by Kinect at a distance of 60cm. However, if the aperture was kept at 4mm and moved further away, there was clutter observed again which confirmed that object details are corrupted with increase in distance from the Kinect. The maximum distance at which a certain aperture could be detected was plotted on the graphs. Similarly, other points too have been plotted in figures 14 and 9. For a given aperture size, ten such readings were taken and the mean value of the readings was used for the plot. The aperture size was increased by 1mm and a mean of ten measurements for that corresponding aperture size was repeated. The detection was appreciable after 4mm and not so good for lesser values.

The experiment was repeated for Openni drivers. It was observed that the Kinect starts detecting the details well if the aperture size greater than 6mm. The detection was appreciable after 6mm and not so good for lesser values.

It was also seen that for higher values (not in figure 14) for Libfreenect drivers that the slope of the graph tends to dip. To be sure that it was not the driver responsible for the graphical pattern, the same test was carried out on Openni driver and the following results were observed:

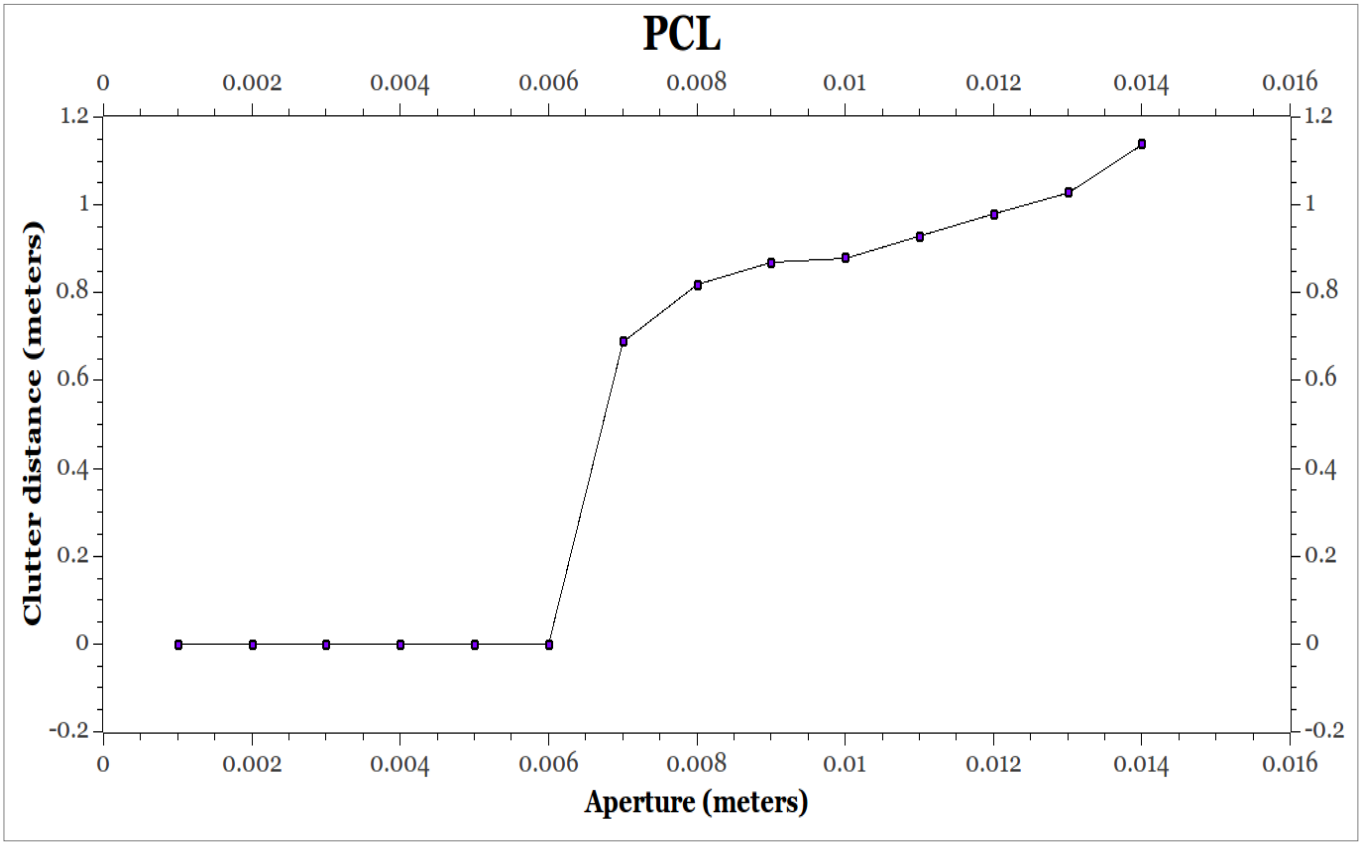

Figure 15: Clutter graph for Openni driver

A software limitation of the Openni driver is obvious from the observations in the graphs above. It could be argued that Openni does not detect the aperture size as sensitively as Libfreenect since it starts detecting only if the aperture size is greater than 6mm and a distance of around 70cm and should not be considered. However, there is a major advantage that is mentioned later in this section.

The quality of the image decays as an object is moved farther away from the camera. It can be seen that the graph obtained from Openni and libfreenect drivers seem to "resemble" a logarithmic behavior. It could not be said for sure at this point but could be confirmed with the following tests.

In order to determine if there was any logarithmic relation between clutter distance and aperture, certain points in the graph were put in known mathematical equations to see if they behaved according to the equation. Two points were selected greater than aperture size of 8mm because that is where the graph starts resembling a logarithmic graph. It is known that slope of any logarithmic graph is less than 1 in the first quadrant of a 2D Cartesian co-ordinate system. Hence, the slope of the line between x-co-ordinates 0.85 - 0.9 cm was calculated.

I) The minimum slope on the graph was:

$(y_2 - y_1)/(x_2 - x_1) = (0.88 - 0.87)/(0.01 - 0.009) = 10$

$\theta = atan(10) = 84.29$ degrees

The slope of the graph after 8 mm is not less than 45 degrees and hence cannot be a logarithmic graph. Besides, the slope was not decreasing for higher values of x as required logarithmic graph.

II) The second test was when a trial was done to fit the points in the graph to a logarithmic equation. The graph was shifted down in the negative Cartesian Y-axis by **0.69meters** because this point is where the graph changed slope and hence is possible get a clear picture if the co-ordinate system is shifted such that these set of points could be fit into general logarithm function.

After the shift, the graph now cut the x-axis at **x = 0.007.** The modified logarithmic equation will be of the form:

$y = \log_n(x+c)$; *where* n *is unknown*

A logarithmic graph cuts the x-axis at x=1. Hence, the value of c would be

c = 1-0.007 = 0.993

⇨ **$y = \log_n(x+0.993)$;**

Trial and error method was used and corresponding x, y (transformed y-coordinate) values in the graph were substituted. The results were checked for several values of **n which was found to be 1.015** i.e. **almost 1**. It is known that logarithms with base unity have indeterminate solution ruling out the possibility of the graph obtained being a logarithmic graph based on our observations.

The above tests proved that the function could not be approximated with a logarithmic function. Nevertheless, our results and the graphs would be a major contributor in determining optimal position to place an object where the detection would show best results. For example, an object with aperture size of 10mm will have best detection approximately between 60-80mm (Figure 15) for Openni and between 60-110mm (Figure 14) for libfreenect respectively.

Libfreenect driver shows a linear behavior between 80-160 cm from the Kinect and smaller aperture sizes are detected better. On the downside, the construction of range-images from raw input point clouds can involve some loss of information as a result of multiple 3D points projecting to the same 2D range image pixel does not have a consistent graph for greater distances[26]. However, raw point clouds are a natural way to represent 3D sensor output point clouds are a natural way to represent 3D sensor output and there is no assumption of available connectivity information or underlying topology [26]. A pointcloud provides a better description of an object and this was the motivation to select Openni drivers for detection. These measurements highlighted the sensor limitations for a given model at a specific distance.

After trial and error it was decided that a distance **between 1-1.2m** would be good enough for experimental purpose.

5.2 Filtration

Pointclouds are detailed and contain a lot of information. Hence, it becomes increasingly more important to manipulate the image and exclude details that are not relevant. In the setup Kinect is facing the floor. This automatically reduces the chances of including objects in the scene that are not

important. Some objects are placed randomly so as to create a cluttered scene.

The starting point was to place a single test object and observe the scene. Noise could creep in despite the precautions and hence passthrough filter was included. A passthrough filter was implemented for the X axis and the Y axis setting the maximum limit on the viewing window of the Kinect. This eliminated many surrounding points and reduced the noise and was implemented before applying plane segmentation.

There were some points corresponding to the object while the others belonged to the floor (in other words a "plane"). The first step was to isolate the points in the plane from the object which was done using RANSAC (Random Sample consensus) technique. This process is known as segmentation.

Segmentation, as the name suggests is the process of dividing an image into regions that can be later split up for further analysis [27]. This technique was used to isolate the objects under test from the floor. In plane segmentation using RANSAC model, three random points are selected and a plane equation is generated. All other points in the scene are fitted to it and a check is done on whether they satisfy the equation. If they do, they are considered to be "inliers" in that equation.

This process is iterated for many points and the equation that has the maximum number of inliers is the "winner" and is the final confirmed plane equation [28]. All the points lying in the plane were isolated and subtracted away from the scene. The filtration technique was applied with increased number of objects and showed good results.

Segmentation with Kinect is highly sensitive to lighting conditions and the nature of the surface. It is advised here that the background object/s should not be highly reflective. After the procedures followed above if there was any noise then a SOR (Statistical outlier removal) filter was applied. SOR filter operates with the assumption that the distribution of points is Gaussian [11] [36]. It requires setting a threshold value for two parameters namely, the number of "K- neighbors" and standard deviation of the mean distance to the query point. A point on the cloud is selected and its proximity to the other points is calculated. If it lies within a certain proximity tolerance limit decided by the user, then the given point is said to be an "inlier" else it is said to be an "outlier".

The proximity between two points can be calculated by Euclidean distance between them. Since the distribution of points has been assumed Gaussian, two parameters can be considered, namely the average or the mean distance between the points and the standard deviation. The distance threshold will be equal to:

Distance threshold = Mean + stddev_mult * stddev. [35][36]

Standard deviation multiplier has been set to 1 for simplicity. This value can be changed by trial and error based on the environment. Points will be classified as inlier or outlier if their average neighbor distance is below or above this threshold respectively. Filtration process can be summarized in the following flow:

Raw pointcloud data → Passthrough XY filter → Plane segmentation → SOR

5.3 Keypoint Extraction

Keypoints are specific regions or points that are unique to an object. These specific regions are invariant to the position and also to illumination to a certain degree which make them robust and reliable. These features have a high rate of good matches. The computation time of descriptors is also reduced because keypoints "downsample" the points thereby reducing the number of points meaning lesser descriptor computation time. Downsampling involves computing a weighted average of the

original pixels that overlap each new pixel [37].

The pointcloud data that has been obtained contain a lot of points and hence is heavy on the processor. These have to be downsampled to make them faster for processing. Though it is not a requirement in our project to achieve real-time solution, it helps save time. Points on a cloud which are close to each other usually have same characteristics. So, instead of selecting all of these points a few points termed as keypoints are extracted from them in a user defined region. In our case, the user defined region is the one obtained after segmentation and removal of outliers. There was another advantage of extracting keypoints. Tools are available that can be used for converting CAD models to pointcloud format. However, the pointcloud obtained from the object was very dense and had a different resolution from that of the Kinect setting resolution. The algorithms could not compare the object model and scene and downsampling of the model was necessary so that they could be compared with the actual scene. The reference model and the scene, both have to be of the same resolution for best results while downsampling/extracting keypoints. This was because; earlier while creating a reference model section (section 5.1.1), downsampling resulted in data loss of details in an already sparse pointcloud

The effects of keypoint radius on recognition are presented in results section. Resolution of the pointcloud was **0.002m** and hence radius of the keypoint was kept between 0.002m and 0.003m. This constraint on the keypoint would mean that all the surrounding points within this radius can be downsampled.

5.3.1 Uniform Sampling

In uniform sampling, a set of points that lie within an imaginary 3D cube of certain volume or voxel can be approximated with a single point representing the centroid of this cube or voxel. Uniform sampling not only down samples the pointcloud but also provides some filtration. Since downsampling involves the weighted average of new pixels, it reduces the number of invalid points. Uniform sampling frames the pointcloud in 3D Grid as shown in figure 16.

The size of each voxel is user defined based on the level of downsampling aimed to achieve and the amount of detail desired to retain in the object [1]. For simpler objects, larger size for each voxel can be selected whereas for smaller objects and objects with a complex geometry, the voxel needs to be smaller. An array of voxels in the cloud is known as voxel grid. Figure 16 shows single voxel and then the 3D grouping of voxels and voxel grid.

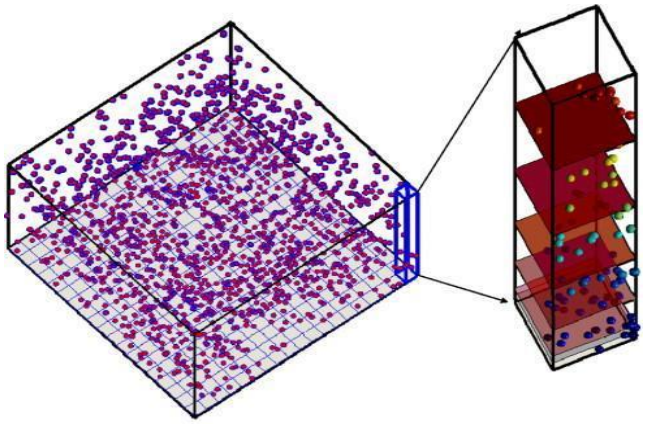

Figure 16: Voxel-grid representation of a Pointcloud [1]

5.3.2 SIFT

SIFT is an acronym of Scale Invariant Feature Transformation. As the name suggests, these keypoints are invariant to change in scale and rotation. Kinect camera is very sensitive to minor changes in illumination which can affect the segmentation of the background from the actual scene before computing the keypoints. SIFT keypoints have partial invariance to change in illumination [2]. SIFT requires the point cloud to have either the RGB data or intensity data or both. The general idea of SIFT is explained below. SIFT computation can be divided into three stages:

a) A Scale space

In this step Gaussian blur function is applied incrementally to an image creating a "scale space". The Gaussian smoothing operator is a 2-D convolution operator that is used to `blur' images and remove detail and noise [53]. Let us assume an image I(x, y, and z).

$$\mathbf{L}(x, y, z, \sigma) = \mathbf{G}(x, y, z, \sigma) * \mathbf{I}(x, y, z)$$

The above is a convolution between Gaussian function and the image, where x, y, z are the coordinates, σ is a scaling parameter and L is the scale space [2]. The above equation has been explained for 2D [54] but the same logic is applicable for 3D in PCL. After the image has been progressively blurred, the resulting set of images is known as an octave. Hence, an octave contains a set of progressively blurred images of a given size. The size of the image can be changed multiplying or "scaling" it by a factor of 2. Several octaves can be constructed for each instance of a resized master image.

b) Laplacian of Gaussian

Differentiating the image to obtain edges and corners will be a time consuming process. Hence, the difference between two consecutive scales in an octave is

$$\mathbf{D}(x, y, z, \sigma) = (\mathbf{G}(x, y, z, k\sigma) - \mathbf{G}(x, y, z, \sigma)) * \mathbf{I}(x, y, z) = \mathbf{L}(x, y, z, k\sigma) - \mathbf{L}(x, y, z, \sigma) \text{ [2]}$$

In the above equation, D is the "difference of Gaussian". The variable '*k*' here is the factor by which the amount of blur differs in successive images for a given octave. Difference of Gaussian calculation is done for all the octaves present in the scale space.

c) Keypoint extraction

This is the step where the modification has been done for 3D. The original algorithm for 2D computes maxima and minima after obtaining the difference in Gaussian. Figure 17 shows the difference of Gaussian images in a given octave scale. Consider the pixel marked 'X' in figure 17. It is compared with its neighbors in the current scale and immediate neighbor scales. Hence the number of neighbors will be nine in the top scale, nine in the bottom and six pixels surrounding the pixel. If the difference of Gaussian value is greater than the neighbors' values then it is classified as local maxima and if it has the least value among all neighbors, it is classified as local minima.

In PCL, the pixel is a 3D point in the pointcloud. The neighbors or neighboring pixels here are the points lying within a search radius (a user input). A pixel can be classified as a keypoint if:

I) The difference of Gaussian value is an extrema (maxima or minima) and if it is an extrema for its own scale.

II) The point is also an extrema for the immediate smaller and immediate larger scales in the scale space.

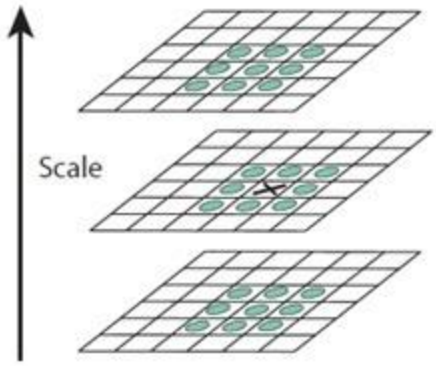

Figure 17: Computing maxima and minima in an octave [2]

5.4 Descriptors

It is not enough if only the keypoints in a pointcloud are extracted. The image has to be made rotation invariant so that matching can be done regardless of the orientation of the object. This can be achieved by calculating descriptors. Object Recognition revolves around the concepts of Surface Matching and Feature Extraction. An object model has to be "described" or depicted based on characteristics of the surface of the object so that it can be matched based on these characteristics with other objects.

There are many profound state-of-the-art descriptor options today, but two of them namely Spin-Image [3] and SHOT [4] have been focused upon. Spin-Images are currently being used in the industry due to their high reliability, accuracy and robustness. SHOT is a recent development and has been shown to have a better performance than spin images [4].

5.4.1 Spin Image

Surface matching is the process of determining the similarity between shapes of two compared objects. In bin-picking scenario, things are complicated because of the amount of clutter and occlusion involved which meant that the descriptor should measure local surface properties. Besides, the objects in a bin are randomly oriented. Local surface properties aid in breaking down the scene into smaller chunks that can be compared with the features of the model for matching [6].

Spin-Image is an algorithm that provides the position of a point in 3D as a 2D co-ordinate representation. They are rotation, scale and pose invariant [3]. The object surfaces are assumed to be regular polygonal surface meshes of a certain resolution. This is known as the mesh resolution. In other words mesh resolution can be defined as the median of all edge lengths in a mesh. Edge length means the distance between two adjacent points in a cloud. The mesh here would be the surface that would be obtained if all points in the pointcloud would be triangulated. The resolution of the Kinect pointcloud has been calculated in the program. A prerequisite to calculation of spin-image is to calculate the normals in the object model and in the scene. A normal to an oriented surface point in the cloud is a vector pointing perpendicular to the plane. Figure 18 gives an idea about the normal to an image.

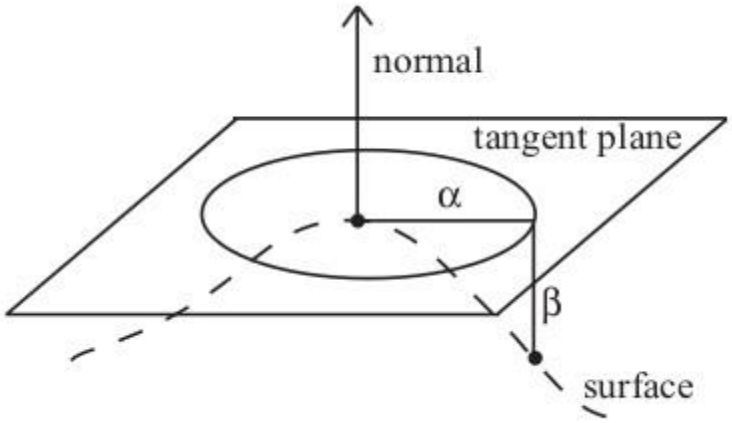

Figure 18: Accumulation of bins in a Spin-Image

Each spin-image pixel is a bin that stores the number of surface points with the given radius (α) and depth (β) [7] [3]. Consider the plane in figure 19 that is rotating about the normal of a point. When it rotates, it sweeps neighboring points in the model. One such point has been illustrated in figure 18 where a neighboring point has been labeled "surface". The radius and the depth give the local position of point on surface with respect to the test point at which normal has been drawn.

There are three parameters that govern the calculation of spin images namely **image width, bin size** and **support angle**. The surface points (α, β) (figure 18) are smoothened out by bilinear interpolation. These points have to be collected in a 2D matrix. This matrix can have any number of rows and columns but are collected in square matrices for simplicity. Image width decides the number of rows and columns in this square matrix [38]. The image width has been set to 8 in our implementation.

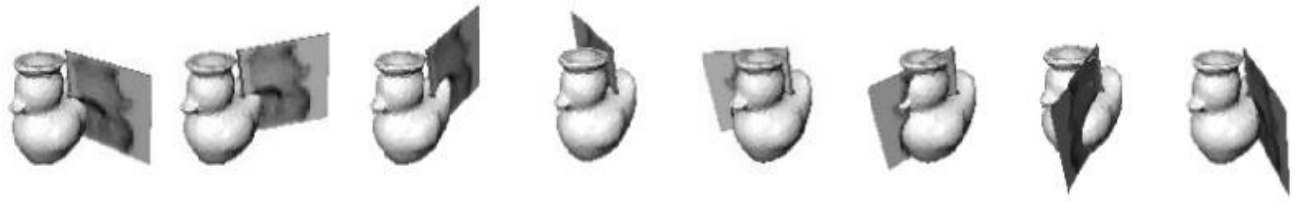

Figure 19: Sweep angle in a Spin-Image [3]

Bin size determines the maximum number of neighboring points that can be grouped together for a given point. The representations of all these points are in 2D. In PCL, bin size is calculated as follows:

Bin size = Search radius / Image Width

The search radius is a user input that considers all the neighbors in a specific periphery around a vertex. Bin size should be set equal to the spin-image resolution value for best results. A larger bin size means that more number of vertices will be included and larger the bin size, the lesser sensitivity to minute details [7].

The support angle is stored as the cosine of the angle of orientation. Spin-Image generation at an orientation point can be considered to be a plane sweeping a cylinder with its axis of rotation at the normal of the orientation point. The angle by which the plane rotates is known as support angle. In the picture above (Figure 19), as the plane swept the duck, it collected the neighboring points.

The sweep is determined by the third parameter which is the support angle (cosine of support angle). The support angle is selected to minimize clutter. If the value is 1 (cos 360) then it makes one complete rotation. The support angle has to be selected carefully. In a cluttered scene, more often than not, a complete rotation sweeps points that may belong to another object and wrongly be calculated as a feature of the same object. A sweep angle between 45 degrees and 90 degrees can impart the selective localization of the object and reduce the chances of including a feature of the other object.

The spin image descriptor calculated was stored in the form of a histogram. Bilinear interpolation gives a sort of "weighted average" from surrounding points that are collected by the rotating plane. It is these weights that are stored in the histogram. The histogram is a vector of 153 fields (17x9), but they can be adjusted to collect 64 fields (8x8) because image width has been set as 8.

After calculation of the spin images, they were compared by Euclidean distance measurement between the scene and the model. A perfect match would mean that the Euclidean distance is zero; hence, theoretically smaller the distance, better the match.

5.4.2 SHOT

SHOT is an acronym of Signature of Histogram of Orientation and was developed by Federico Tombari, Samuele Salti, and Luigi Di Stefano. This algorithm incorporates the best features of both signature and histogram way of describing a particular object [4]. It is a recent advancement in the field of image recognition and has proved to be a very accurate and powerful tool. SHOT is efficient, descriptive and robust to noise [4].

In SHOT, an invariant reference axis is defined against which the descriptor is calculated describing the local structure around a point of the cloud under consideration. Descriptor of local structure around a given point is calculated based on a radius value input by the user for only the points within the support of the radius. The descriptor here is a histogram of 352 fields (descriptor length) which should be interpreted as a 2D 32x11 matrix. The maximum width of this histogram here is 11 (10+1) where 10 is the user input value. This input value "n" can be changed by the user to any value less than 10 if desired and would be taken up by PCL as (n+1).

Reference axis is based upon a spherical co-ordinate system [4]. Points around a feature point are collected within the defined radius input by user according to a **"function of angle θ"** and not θ alone, where θ is the angle between normal at each point within the support radius and the normal at the feature point [4]. The support angle range for a normal is [-π, π]; hence, if the user input is 10 and if the grouping were done based on θ alone, then each interval or "bin" of the histogram is as follows:

Bin 1 = [-π,-9 π/10), Bin 2 = [-9 π/10,-8 π/10), Bin 3 = [-8 π/10,-7 π/10) …

Say, if the inclination is between the normal at a feature point and another point is *-19π /20*, it should lay in the bin 2. But since the binning is done based on ***cos θ*** and not θ, the binning is done on a range of [cos (-π), cos (π)] i.e. [-1, 1]. Hence, the binning is also done accordingly.

Earlier in this section, it was observed that the binning was taken up as n+1 by PCL. Since the upper limit of the bin interval is always excluded, there might be an extra bin added to include the last value of the last bin.

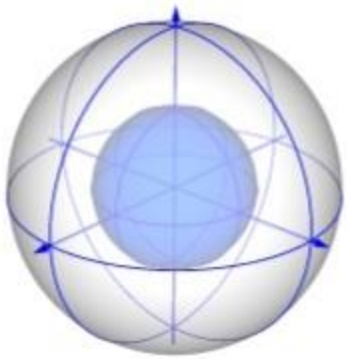

Figure 20: SHOT structure [4]

5.5 Point Matching and Clustering

After the descriptors have been calculated, the model and scene have to be matched. The descriptors are matched based on Euclidean distances. The smaller the distance between the descriptor points, the better the match. In ideal conditions, the distance is supposed to be zero. In this project, the threshold has been kept at little above zero because it is impossible to have a perfect zero score.

After point matching stage, the points have to be grouped together so that they can be identified to belong to a particular instance of an object. Say, a point on the model matches with two points on the scene, the user should be able to determine that both points on the scene show the same feature but belong to different objects. This can achieved with the help of clustering which is out of scope in our project.

Clustering has not been explored deeply in this project and can be placed under future work. To begin with, Euclidean clustering can be looked into for identification of simple objects in the bin. This can be modified for complicated objects if desired by users.

6 RESULTS

The results have been presented based on the different scenarios. Several combinations of algorithms were tried for the best possible matches who have been presented in the subsections below. For each combination, threshold parameters were changed and the results were observed. However, it should be noted that threshold also depended on other parameters such as descriptor radius, keypoint radius, bin-size (for spin images) have been adjusted for best possible results for the test environment conditions in our set up. The same also applies to filtration and segmentation parameters that have been adjusted for the best conditions and discussed in the preceding sections.

The results discussed in this section have been analyzed on reference models and scene both of which have undergone filtration and segmentation. Results have been shown for scene conditions involving one object and for a scene involving many objects. Effect of descriptor radius and keypoint radius on segmented scenes has also been analyzed to aid future work and exploit the algorithms better.

PFH (Point feature Histogram) was the first algorithm under test. Correspondences between the reference object and the object in scene could not be established at regions where the pointcloud was sparse. PFH attempts to capture as best as possible the sampled surface variations by taking into

account all the interactions between the directions of the estimated normals [33]. A sparse pointcould will not provide enough neighbors for a point to compute the normals correctly. This will not be a correct representation of the surface, which in turn would affect PFH parameters. In this algorithm, PFH algorithm took a very long time of almost over 30 minutes to calculate correspondences without keypoints; the resultant correspondences were not good. However, in regions of the object with a dense pointcloud, a few good correspondences could be seen.

It is to be noted that PFH results were observed when the scene was filtered and without the presence of other objects. The long time it took despite these modifications along with which it also gave a lot of wrong correspondences made us search for better algorithms.

6.1 Spin Image

Spin Images were the automatic starting point for comparing images. It could be recalled that in earlier sections while making a model, they were given as input to the spin image algorithm. They were the first choice since they have been industrially implemented successfully. The results of spin comparison were not satisfactory though many good correspondences were obtained. Due to the limitations in the complexity of the object, there were a lot of wrong correspondences observed. Spin-Image gave good results on simple objects such as cubes and cylinders.

Two test environments were used. In the first was with only one object in the scene which was matched with the model stored in the computer. In the second scenario, there were many test objects placed in the scene in order to make it partially cluttered. By partially cluttered, it is meant that some of the objects were placed distinctly on the floor while a few others were placed one on top of another in a random manner.

To begin with, a scene with one object was selected and Spin image algorithm was applied to it. Here, changed threshold values were changed and the results were compared. Threshold value refers to the squared distance between two descriptors being cmpared. If the value between reference descriptor and scene descriptor is less than threshold value then the point is said to be a valid match.

Threshold values are decided by trial and error. In this experiment, a stringent **threshold value of 0.1** was selected and results calculated. Next, the threshold was relaxed to a **value of 0.25** and the results were noted down. These results were then compared against each other.

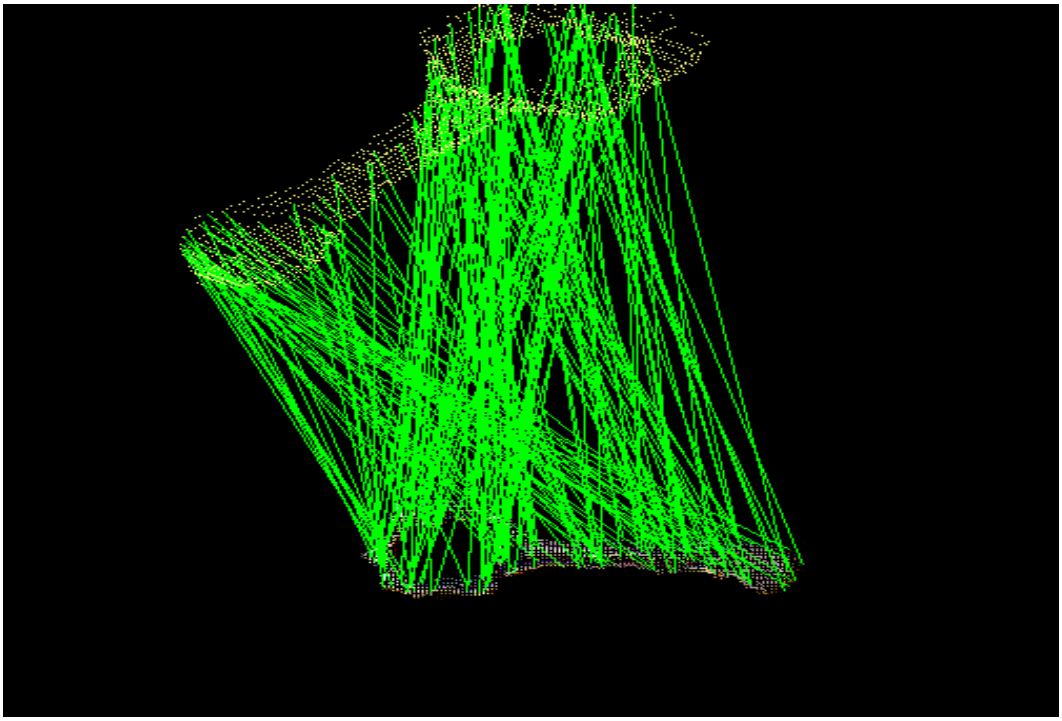

Figure 21: Correspondences with SPIN with threshold limit 0.25

LEGEND:
ms - Milliseconds
NA – Not Applicable

Threshold limit = 0.1

Keypoints Calculation time	**NA**
Descriptor Calculation time	**126 ms**
Total Recognition Time	**5719 ms**
Correspondences	**1899**
Results	**Partially good**

Table 2: Statistics for single object matching for threshold=0.1

Threshold limit = 0.25

Keypoints Calculation time	**NA**
Descriptor Calculation time	**127 ms**
Total Recognition Time	**5874 ms**
Correspondences	**1899**
Results	**Partially good**

Table 3: Statistics for single object matching for threshold=0.25

The recognition time was very low but there were several wrong correspondences and can be seen from figure 21 above. Table 2 and table 3 do not have a major difference and changing the threshold value did not have a major impact on the results.

Threshold limit = 0.25

Keypoints Calculation time	**NA**
Descriptor Calculation time	**NA**
Total Recognition Time	**NA**
Correspondences	**NA**
Results	**Error**

Table 4: Statistics for cluttered scene for threshold=0.25

Table 4 shows the results for a cluttered scene for a threshold value of 0.25.

6.2 SHOT

The results here were better than the results obtained only with spin images. The number of wrong correspondences reduced and in the scenario with only one object in the scene, the number of correspondences was very good. However, some wrong correspondences managed to creep in which were sorted out by lowering the threshold value in the scene. Similar to the tests in spin image, the threshold values of SHOT were changed and the results observed. To highlight the effect of threshold values, a threshold limit of 0.1 showcases stringent conditions. An increase in the value means that threshold has been relaxed.

Threshold values that are too stringent may lead to no matching at all as was seen with threshold value 0.1.After trial and error experiments it was found that 0.25 was the optimal value for threshold since higher values resulted in noise and incorrect correspondences. It should be kept in mind that it is descriptors in our project that are responsible for matching and keypoints are just for downsampling and scale invariance.

Threshold limit = 0.1

Keypoints Calculation time	**NA**
Descriptor Calculation time	**10381 ms**
Total Recognition Time	**NA**
Correspondences	**0**
Results	**No match**

Table 5: Statistics for single object matching for threshold=0.1

Threshold limit = 0.25

Keypoints Calculation time	**NA**
Descriptor Calculation time	**10439 ms**
Total Recognition Time	**11437 ms**
Correspondences	**77**
Results	**Mostly good**

Table 6: Statistics for single object matching for threshold=0.25

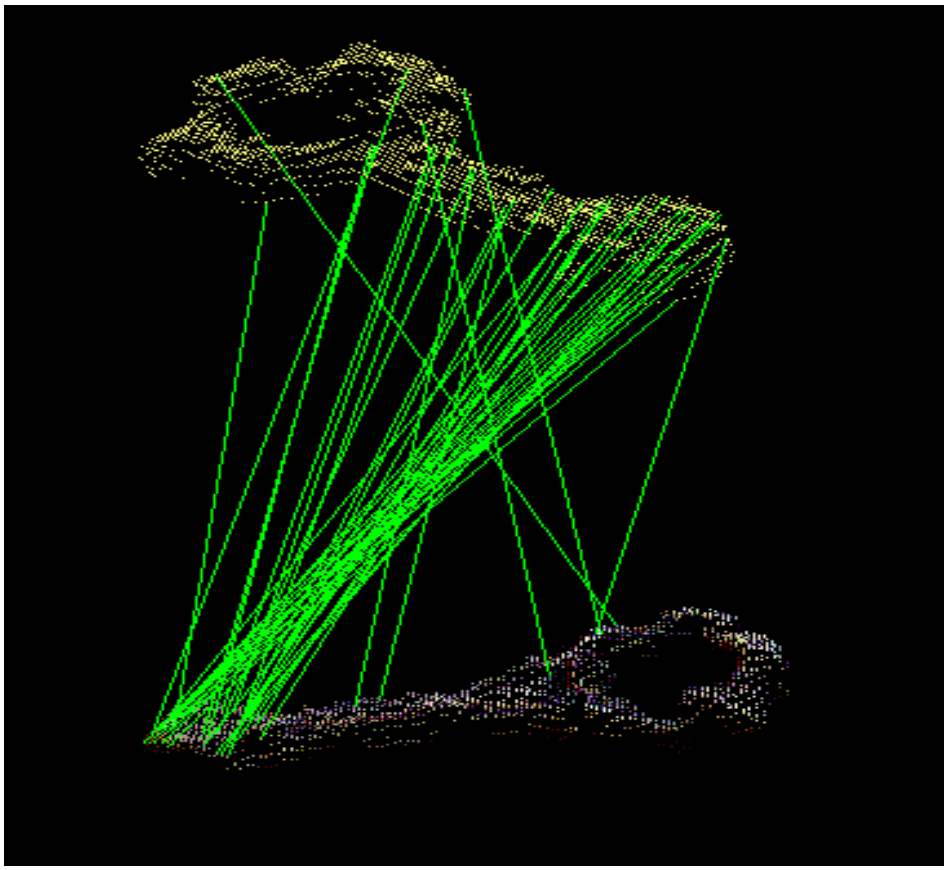

Figure 22: Correspondences with SHOT with threshold limit 0.25

A threshold of 0.1 gave no correspondences but a threshold value of 0.25 gave 77 correspondences. Most of them were very good. As seen in screenshot above (figure 22), it can be seen that the number of good correspondences are appreciably more than the bad correspondences. Though the recognition time was roughly twice that of spin images in section 6.1, they were more reliable

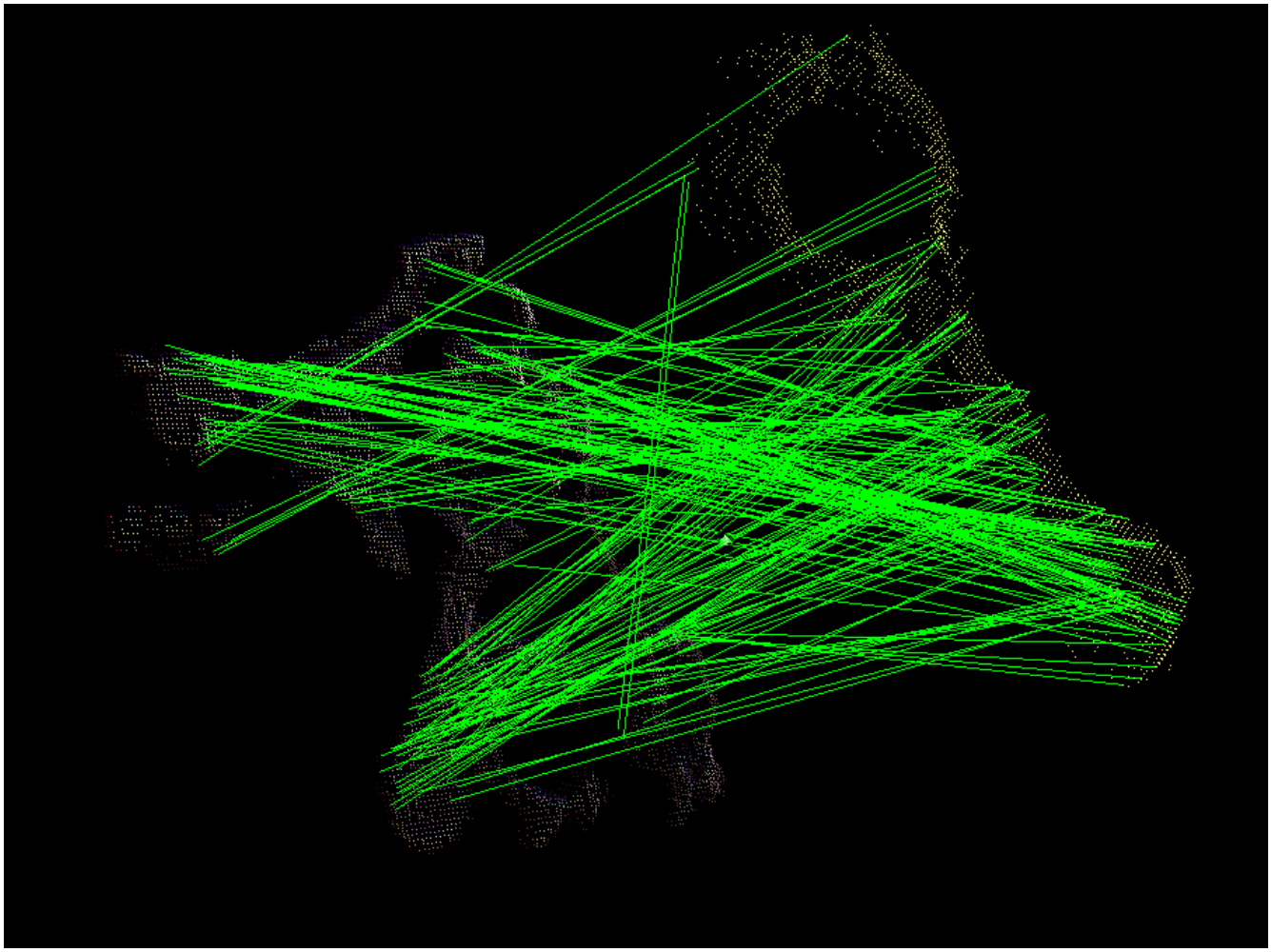

Figure 23: Correspondences with SHOT with threshold limit 0.25 for cluttered scene

Threshold limit = 0.25

Keypoints Calculation time	**NA**
Descriptor Calculation time	**39791 ms**
Total Recognition Time	**41691 ms**
Correspondences	**Not Calculated**
Results	**Ok**

Table 7: Statistics for single object match for threshold limit 0.25

The correspondences were better with SHOT compared to using spin image algorithm for cluttered scene in section 6.1. The better performance with single object environment was an early indication that SHOT may be a better option with Kinect than spin images. However, SHOT alone was not enough to give close to satisfactory results in a cluttered environment as evident from figure 23.

6.3 Uniform Sampling and SHOT

Uniform sampling was the first type of keypoint that was tried in our algorithms. In this section uniform sampling with SHOT has been used. Uniform sampling and SHOT showed good results for simple objects (cubes and cylinders). Our test object also showed good results for a single and isolated object environment. Isolated objects here mean objects in the environment were not overlapping each other. Some promising matches in a test cluttered scene were also observed.

Threshold limit = 0.1

Keypoints Calculation time	**7ms**
Descriptor Calculation time	**9124 ms**
Total Recognition Time	**10028 ms**
Results	**No match**

Table 8: Statistics for single object matching for threshold limit 0.1

Threshold limit = 0.25

Keypoints Calculation time	**26ms**
Descriptor Calculation time	**9193 ms**
Total Recognition Time	**10433 ms**
Correspondences	**77**
Results	**Mostly good**

Table 9: Statistics for single object matching for threshold limit 0.25

In table 9 it can be seen that there was no marked difference for single object identification with keypoints. The same number of correspondences was found when only SHOT was used. However it can be seen that the recognition time was reduced which highlights the use of keypoints.

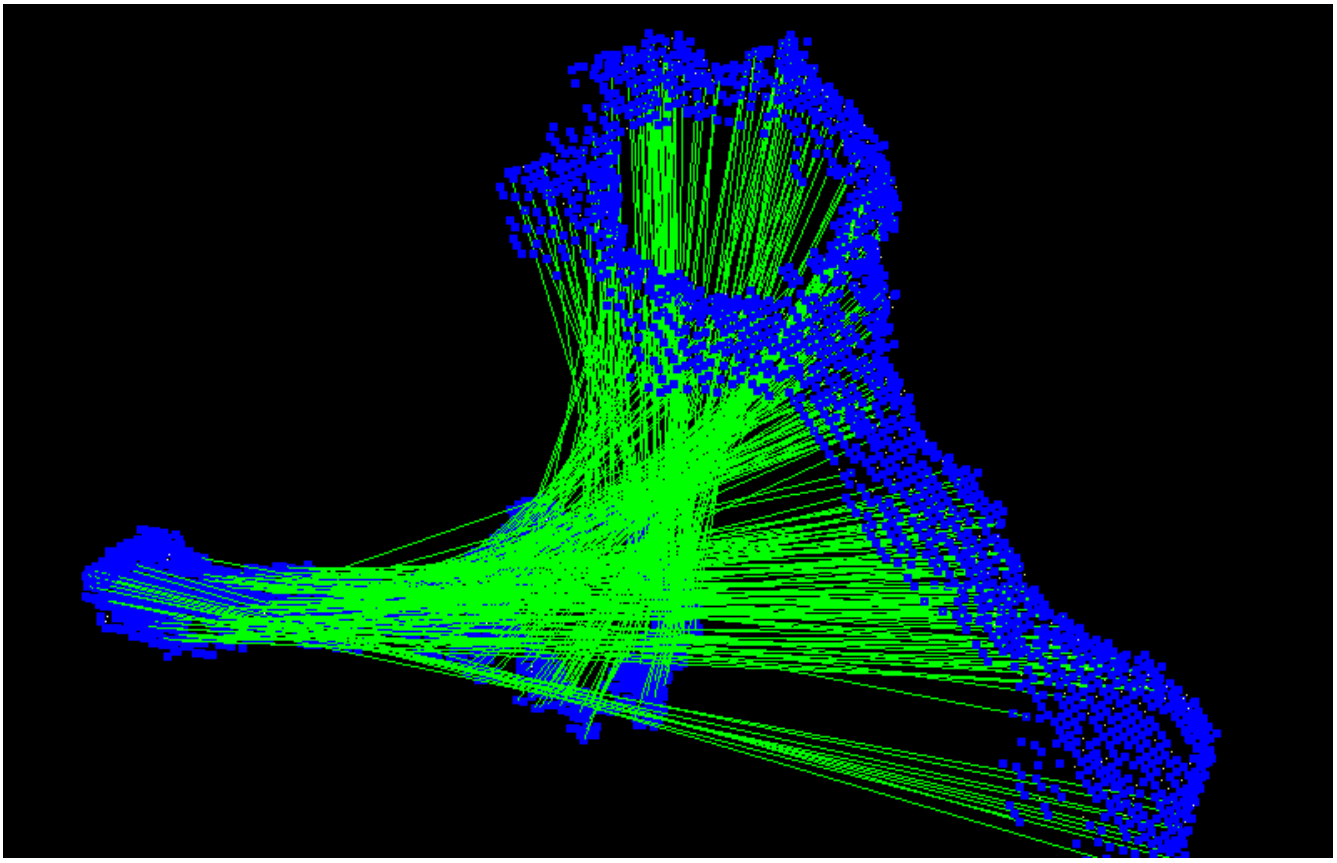

Figure 24: Correspondences with Uniform Sampling and SHOT with threshold limit 0.25

Table 8 shows the results with a threshold value of 0.1. This was not different from what was observed in table 5 also for threshold value of 0.1, where there were no correspondences. If the threshold value is relaxed to 0.25 as in table 9 and from figure 24, it can be seen that the correspondence results are very good.

Threshold limit = 0.25

Keypoints Calculation time	**19 ms**
Descriptor Calculation time	**33727 ms**
Total Recognition Time	**35438 ms**
Results	**Mostly good**

Table 10: Statistics for cluttered scene for threshold limit 0.25

Table 10 shows the data for uniform sampling with cluttered scene. The threshold limit was kept at 0.25 since threshold value of 0.1 gave no correspondences. It can be observed from figure 25 that a lot of wrong correspondences have crept in. They were as good as ones observed when only SHOT descriptor was applied. There was a reduction of 5000 *ms* if uniform sampling was applied compared to when only SHOT was used. Even though the correspondence results did not show a major change what did show was reduction in comparison time. This underlined the advantage of using keypoints and guided us to further explore other keypoints that could overcome the drawback of uniform sampling method of

using them in cluttered and noisy environment.

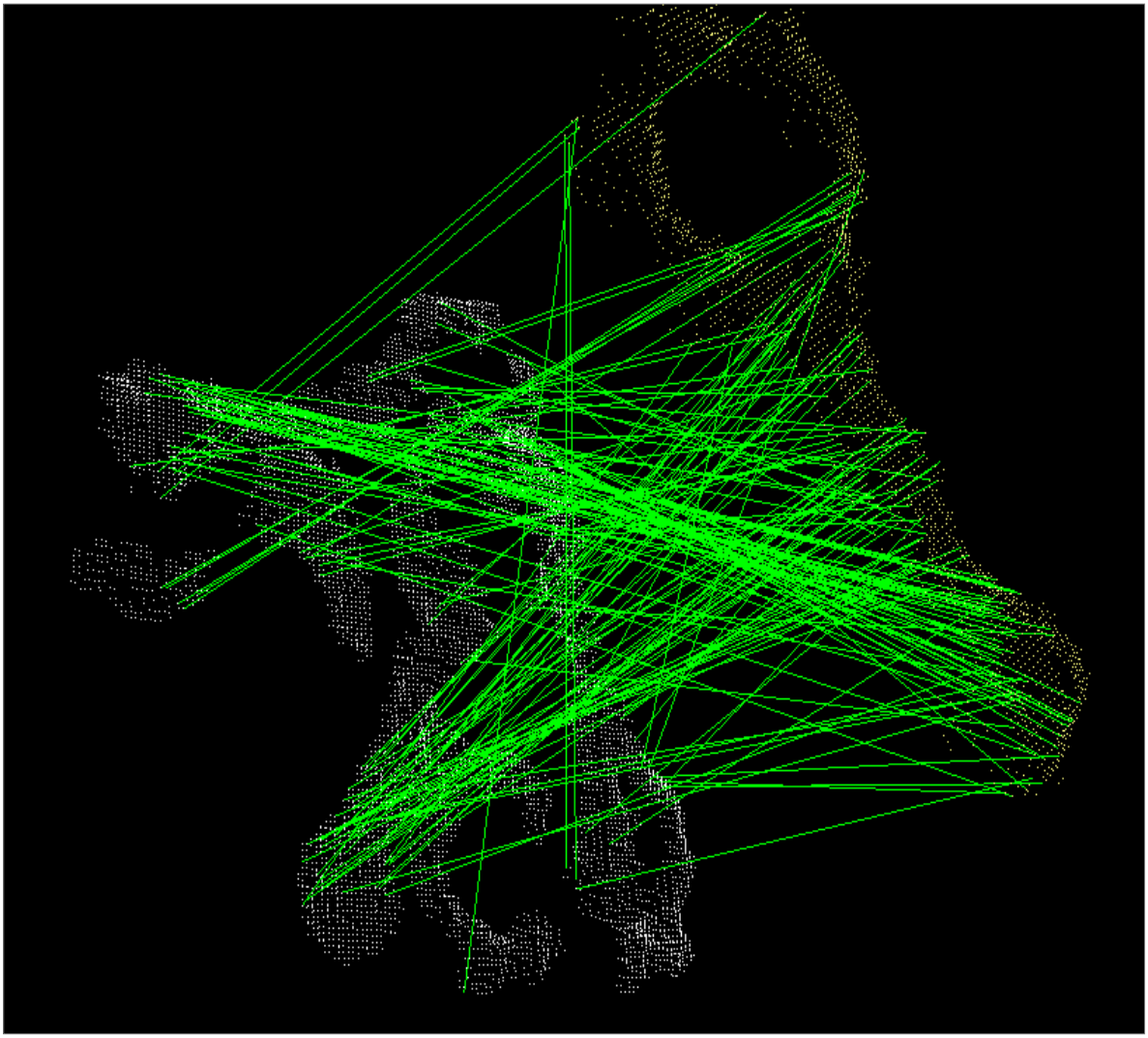

Figure 25: Uniform Sampling and SHOT for cluttered scene with threshold limit 0.25

6.4 SIFT and SHOT

This was the most promising combination of all the algorithms tested. The number of wrong correspondences was relatively lower than all the previous methods.

Threshold limit = 0.25

Keypoints Calculation time	**1871 ms**
Descriptor Calculation time	**8169 ms**
Total Recognition Time	**10842 ms**
Correspondences	**42**
Results	**Mostly good**

Table 11: Statistics for single object scene-for threshold limit 0.25

Threshold limit of 0.1 has not been considered because it did not give us good results. Hence, the analysis was done for higher values and as seen in cases before, it gave good results with a value of 0.25. Table 11 is the results for partially cluttered object scenes where very good. The number of correspondences however are only 42 compared to table 6, where there were 77 correspondences when only SHOT was used. This is because by downsampling, the number of points has been reduced which might look a drawback for single isolated objects but is not a drawback in cluttered scenes where because keypoints are of "better quality". Total recognition time was a shade higher than in section 6.3 table 9 (uniform sampling and SHOT).

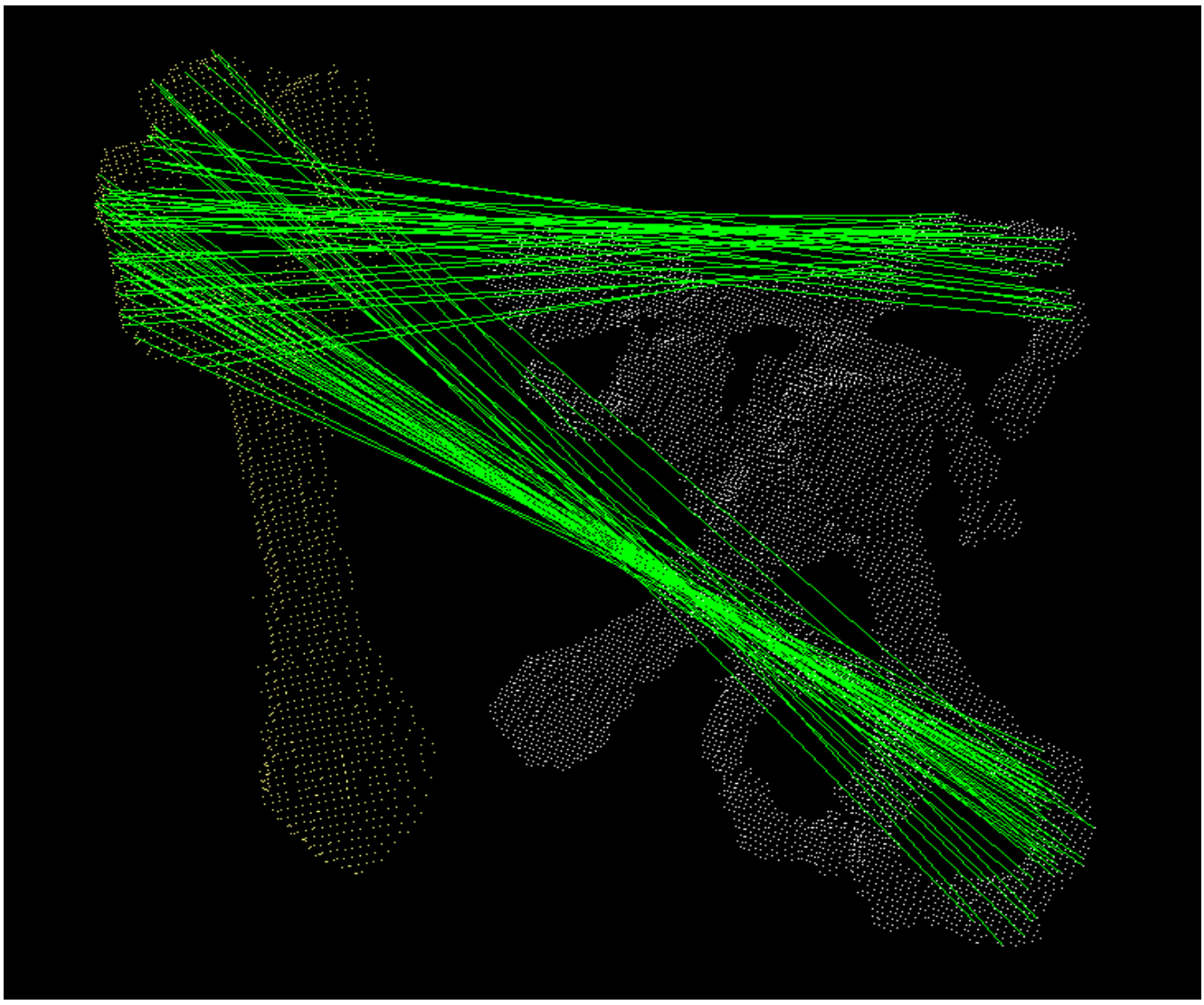

Figure 26: Single object scene with SIFT and SHOT with threshold limit 0.25

In the figure 26, correspondences with the lowest scores have been displayed. The computation time compared to the output of uniform sampling was higher but the number of wrong matches was lower when it came to a cluttered environment. In figure 27, points highlighted in blue are keypoints.

It can be seen that some other objects were also mixed along with our objects of interest and were not

identified which was a good step ahead. Figure 26 shows the correspondences for an isolated environment. Such good result was not observed with uniform sampling method and extra information provided by unwanted objects was weeded out better by this approach. This proves that SIFT is a better keypoint algorithm compared to its counterpart with a little trade-off in total recognition time which was not significant in this case.

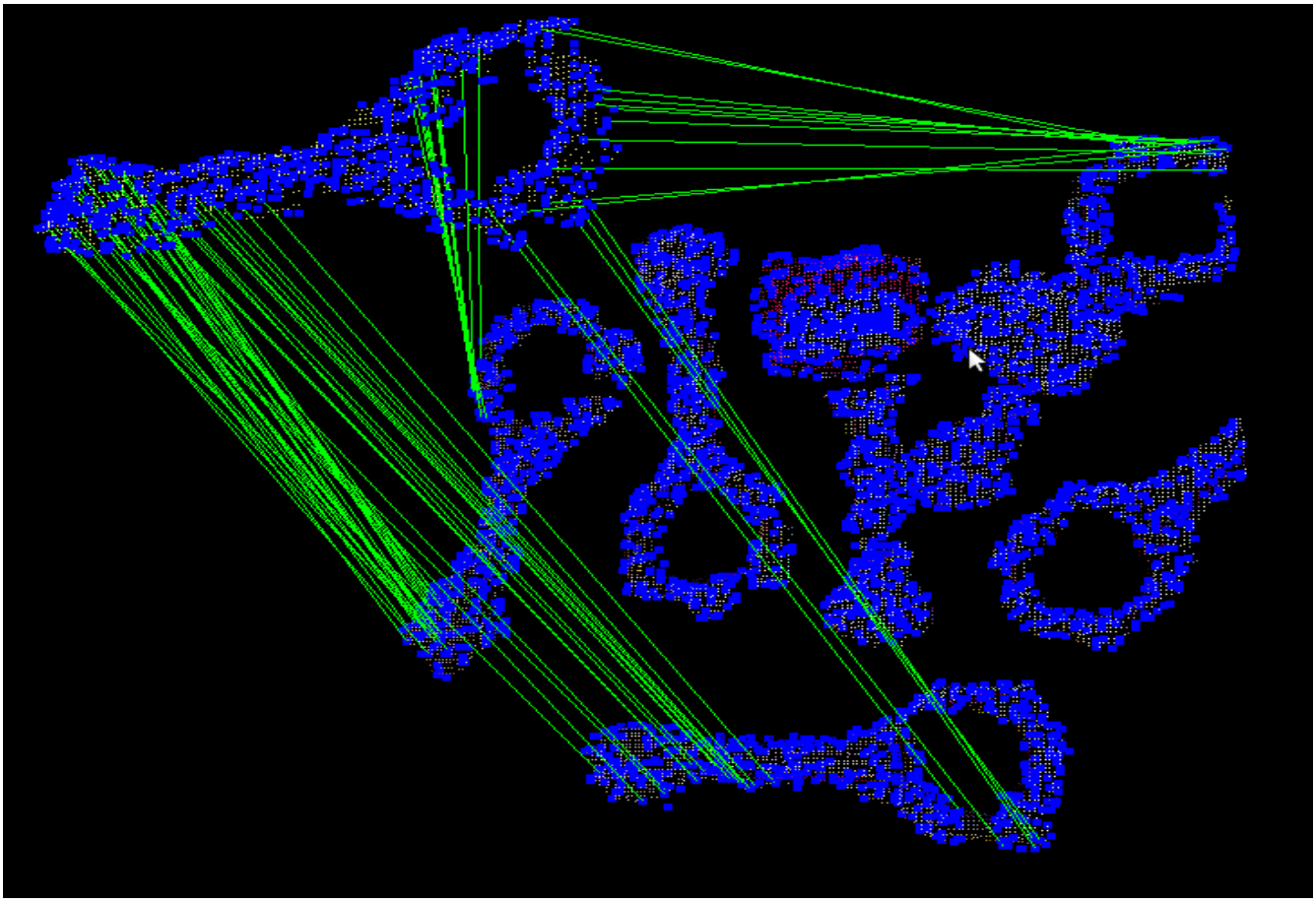

Figure 27: Partially cluttered scene with SIFT and SHOT with threshold limit 0.25

Threshold limit = 0.25

Keypoints Calculation time	**4549 ms**
Descriptor Calculation time	**31136 ms**
Total Recognition Time	**37828 ms**
Results	**Mostly good**

Table 12: Statistics for cluttered scene for threshold limit of 0.25

Table 12 shows that though the total recognition time was a little more compared to the use of uniform sampling keypoints, the quality was much better. In figure 27, better results are seen with overlapped objects compared to spin image algorithm.

6.5 SIFT and Spin Image

The correspondences with SIFT and Spin images were not as good as SIFT and SHOT combination. In addition, double correspondence instances were observed i.e. a single point on model having two correspondences in the scene. This problem could be eliminated by increasing the threshold parameters but then led to wrong correspondences. Hence, this technique took us back to the problem of wrong correspondence elimination which should have been minimized. The results have been presented below.

Threshold limit = 0.25

Keypoints Calculation time	**1908 ms**
Descriptor Calculation time	**5098 ms**
Total Recognition Time	**7296 ms**
Correspondences	**1749**
Results	**Mostly bad**

Table 13: Statistics for single object scene for threshold limit of 0.25

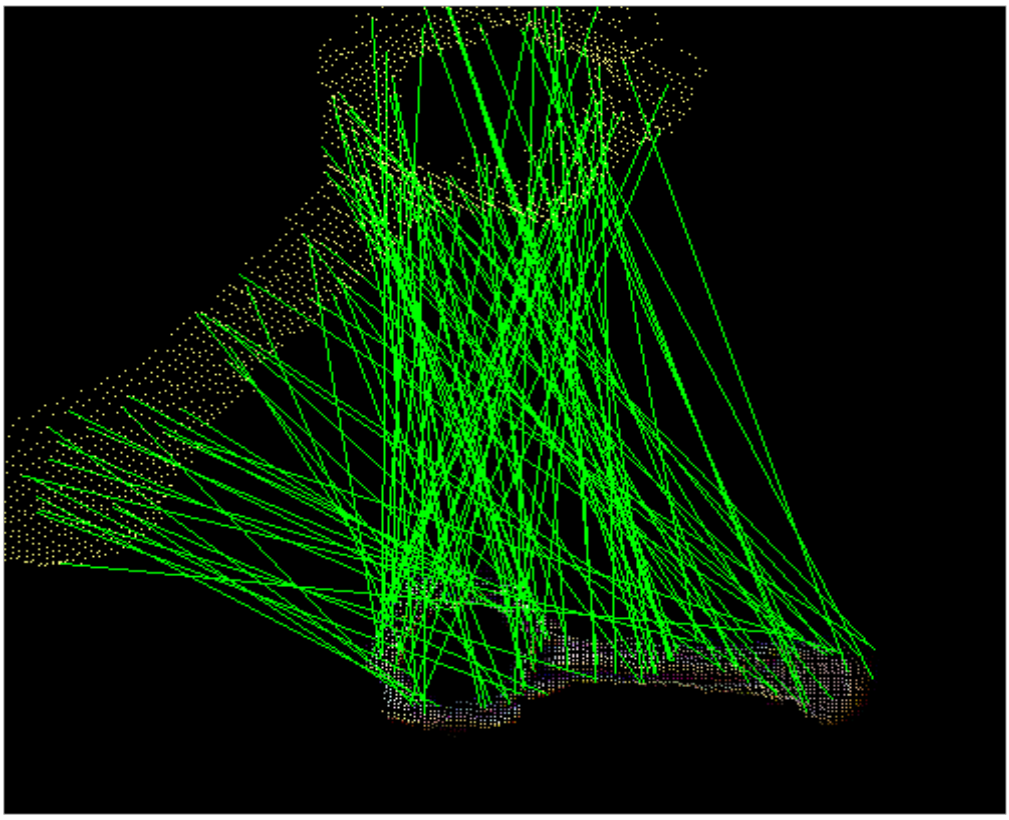

Figure 28: SIFT and Spin correspondences for threshold limit of 0.25

Threshold limit = 0.25

Keypoints Calculation time	**4549 ms**
Descriptor Calculation time	**31136 ms**
Total Recognition Time	**52797 ms**
Results	**Bad**

Table 14: Statistics for cluttered scene for threshold limit of 0.25

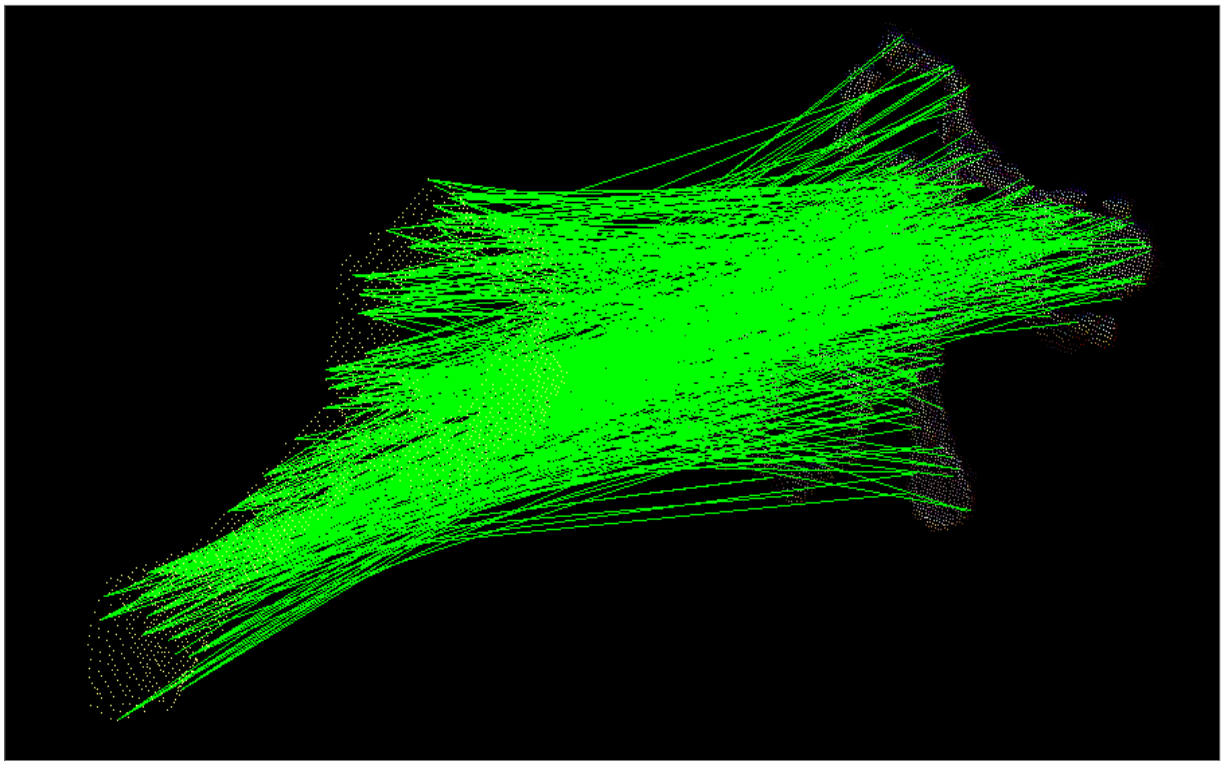

Figure 29: SIFT and Spin correspondences for a cluttered scene for threshold limit of 0.25

The number of correspondences in table 13 is very high but they are bad correspondences as is evident from figure 28. In figure 29, double correspondences can be clearly observed. Hence, this combination was not a wise choice for recognition with Kinect. The reliability of this approach is lower than SIFT and SHOT combination. More number of correspondences was observed but there were many points with double correspondence matches.

6.6 Analysis of Keypoint and Descriptor radii

Model radius	Model Keypoints	Correspondences	Results
1	2 out of 1981	5	Bad
0.1	6 out of 1981	6	Bad
0.05	15 out of 1981	6	Bad
0.01	199 out of 1981	69	Bad
0.009	241 out of 1981	91	Bad
0.005	620 out of 1981	207	Bad
0.004	878 out of 1981	249	Bad
0.003	1305 out of 1981	307	Ok
0.002	1790 out of 1981	340	Mostly good
0.001	1981 out of 1981	349	Mostly good

Table 15: Number of correspondences with change in model radius

In table 15, the scene keypoint calculation radius has been kept at 0.002m and the algorithm used is Uniform sampling and SHOT. 0.002m is the resolution of Kinect camera at a distance of 1m. As defined it is the distance between two points in a pointcloud.

It can be observed that the number of model keypoints increases with decrease in model radius. A marked increase in the number of keypoints can also be seen. As the radius approached the Kinect resolution of 2 mm, there is an almost linear increase in the number of correspondences. Good correspondences were noted after 0.003m which confirms the fact that closer the scene resolution was to the reference, more reliable were the correspondences.

Another interesting point was that if the keypoint size was reduced beyond the resolution (0.001m), there would still be good correspondences. However, the number of keypoints would be same as the original number of points. This would defeat the entire purpose of using keypoints to reduce the total number of points in the scene, thereby reducing comparison time. Therefore, selecting a resolution less than the Kinect resolution is not profitable.

The same applies to table 16 below where the model radius has been kept constant and changes in scene radius are observed. It was profitable to select a radius between 3mm and 2mm in our case.

Scene radius	Scene Keypoints	Correspondences	Results
1	4 out of 7375	5	Bad
0.1	17 out of 7375	6	Bad
0.05	46 out of 7375	6	Bad
0.01	664 out of 7375	69	Bad
0.009	792 out of 7375	91	Bad
0.005	2078 out of 7375	207	Bad
0.004	2897 out of 7375	249	Bad
0.003	4333 out of 7375	307	Ok
0.002	6997 out of 7375	340	Mostly good
0.001	7375 out of 7375	349	Mostly good

Table 16: Number of correspondences with change in cluttered scene radius

Descriptor radius	Correspondences	Results
0.9	5036	Bad
0.2	974	Bad
0.18	796	Bad
0.15	634	Bad
0.1	227	Mostly good
0.09	353	Very good
0.08	410	Good
0.07	204	Very Good
0.06	181	Good
0.05	121	Good
0.04	138	Good
0.03	293	Good
0.02	340	Mostly good
0.01	806	Ok
0.005	982	Mostly Bad
0.001	0	Segmentation fault

Table 17: Effect of descriptor radius on recognition

Table 17 lists the behavior in the number and nature of correspondences with change in descriptor radius, keeping all other parameters constant. Very high or very low descriptor values lead to bad correspondences. It is to be noted that with increase in descriptor radius, the number of correspondences may go high but almost all of them are erroneous. Unlike keypoint radii where a decrease in radii showed an increase in good correspondences and was almost linear behavior, descriptors had a different tale.

Though the number of correspondences started increasing with a decrease in descriptor radius after 0.05 meters, the quality of correspondences started decreasing and descriptor radius close to the image resolution or keypoint radius led to segmentation faults. A decrease in descriptor radius does not guarantee good correspondences. Correspondence scores are lowest and good results are obtained only for a certain range of values which in this case was 0.04 to 0.1

7 CONCLUSION

This thesis is aiming to provide a cheaper solution to the present day laser bin picking systems around the world by using pointcloud data obtained from a Kinect sensor. The problem statement along with the research motivation is clear and the approach towards the solution is justified and logical. A brief introduction to the PCL library and a methodical approach has helped in addressing the problems at several checkpoints such as during filtration and extraction of interest points. SHOT descriptor computed on SIFT keypoints of filtered pointcloud have shown the best results.

SIFT with SHOT gave very good correspondences even with a poor quality of object/scene cloud for simple objects. Regarding complicated images, one-on-one correspondences had very low wrong correspondences.

For partially aligned objects, they gave strong correspondences when the threshold value was lowered for the match criteria. It can prove to be a good combination and prove competitive to the existing stable spin image algorithm.

The roadblocks with respect to the Kinect hardware have been addressed and feasible precautions/solutions have been provided. The implemented algorithms have proven to work well under controlled conditions and better results can be achieved by fine tuning the parameters and external conditions such as illumination.

The replacement of laser cameras by Kinect is not efficient as of now from an industrial point of view but is within the realms of possibility with further development in hardware and software.

8 FUTURE WORK

The Kinect camera operates at 30Hz which causes fluctuation in edge and boundary data of a pointcloud. This can be overcome by applying the concept of temporal smoothing. The frame rate can be reduced so that the borders of a surface could be smoothened and the values more consistent.

Clustering algorithms can be used to group together point-to-point correspondences after comparing the descriptors in the scene and the model. Geometric consistency is a very effective algorithm in this

regard. Layered approach can be implemented while capturing images. In the first instance, the camera captures the bin image and the rest of the processing takes place. After the objects have been identified and picked, depth filter values can be incremented for the next iteration to reveal more objects in the scene. The value can be incremented provided all possible objects that could have been picked are identified and picked in the current scan. A passthrough filter on the z-axis could reduce the noise and configured to extract selective information.

The results of Euclidean distance matching algorithm have been used in our project which is a basic method. Advanced and more accurate methods such as Chi-square matching can be implemented for more accurate and reliable results. Chi square is a measure of goodness of fit. The goodness of fit helps to summarize the discrepancy between observed and expected values. In our project, the expected value will be the descriptor vector calculated in the reference object and observed value would be the descriptor calculated on the scene object.

Index	Observed value (Oi)	Expected value (Ei)	O-E	$(O-E)^2$	$(O-E)^2/E$
1(N_1)	42	50	-8	64	1.28
2(N_2)	55	50	5	25	0.50
3(N_3)	0	50	-50	2500	50
Total					51.78=χ^2

Table 18: An Example of a Chi square evaluation table

Say that the descriptor vector is of the format ($N_1a+N_2b+ N_3 c+\ldots$) where N_1, N_2, N3 are constants. In table 18, χ^2 value has been calculated. This value has then to be referred to a χ^2 table for N-1 (here, 3-1 = 2) degrees of freedom. In our project for example, if spin images are considered, there can be a maximum of 80 degrees of freedom since the maximum image width is 9 and the maximum possible value of image width is 9x9 = 81; hence, 81-1 = 80 can be the maximum number of "dimensions" for our vector descriptor. If the calculated χ^2 value is less than the value in the table for a "T percent" level of significance, then it is said to be a good fit or in other terms a good match. This method is more accurate than Euclidean distance and hence could be a better parameter in accepting or rejecting a hypothesis which in our case is comparison of two vectors.

There is scope for applying surface reconstruction algorithms such as MLS (Moving least squares). MLS method reconstructs the noisy surface of the pointcloud. However this may result in loss of some features in the original image that could be vital for comparison. Hence, the required parameters should be set carefully by trial and error.

9 APPENDIX

There is an offset between RGB color image and depth image. Kinect calibration is desirable for accurate results because it aligns the RGB camera and the depth camera and gives a transformation matrix. If it is desired to find the location of the object in world co-ordinate frame instead of camera co-

ordinate frame, calibration becomes all the more necessary. A pointcloud could be a set of 300000 points. To be more exact, if the pointcloud contains 640x480 points in an organized pointcloud. It could also be adjusted to 320x240, but the former settings make it easier for one to one mapping with its corresponding RGB image.

Two methods for calibrating the Kinect camera have been discussed. For more details on installation of Kinect drivers (other than Openni which is provided by default when PCL is installed) the book "Hacking the Kinect" Jeff Kramer, Nicolas Burrus, Florian Echtler, Daniel Herrera C., and Matt Parker could be useful. The procedure for calibrating Kinect is as follows:

a) RGBD

RGB Demo is a good tool to calibrate the Kinect Camera, it is easy to install and then series of steps to use RGBD for calibration are as follow [25].

Run rgbd-viewer from terminal.

Select Dual IR/RGB Mode from application window.

Start capturing the images of checkered board recommended is 30, while image capturing try to be as close to the camera as possible and the corners should be properly visible.

Checkered board should be visible and not be blacked-out, this for depth calibration.

After taking 30 images Cover the IR emitter and start taking the images for the calibration of Color Camera.

Again take 30 images considering that corners should be properly visible.

Exit rgbd-viewer.

Run calibrate_kinect_ir.

At the end check out the pixel projection error, it is supposed to be less than 1 for good results. If it is not less than 1 then again run the procedure with better angles and movement of check-board.

b) MRPT Camera Calibration

Mobile Robot Programing Toolkit (MRPT) was a handy toolkit. MRPT was a project of MAPIR labs of University of Malaga, Spain [22]. Its libraries and applications were very useful. In MRPT tool kit there were several applications for different functionalities; one which is used for Camera Calibration is MRPT Camera Calibration 2.0. It has very friendly environment for a list of available cameras. The steps to calibrating the camera are given below:

First, select the required camera to be calibrated from a dropdown menu. Then the check board Detection parameters need to be adjusted. Pressing the start button started the Pictures/Point Cloud capture from camera and when it finished the capturing just calibrate button was pressed.

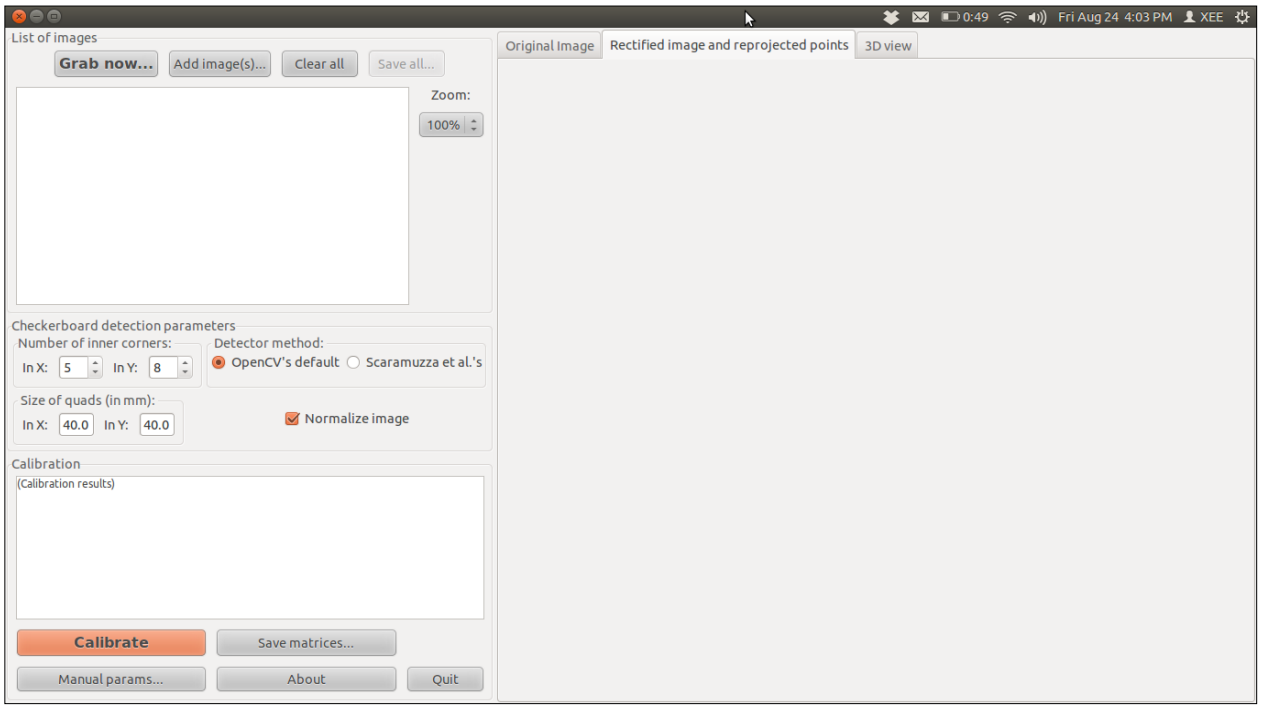

Figure 30: MRPT Camera Calibration_1

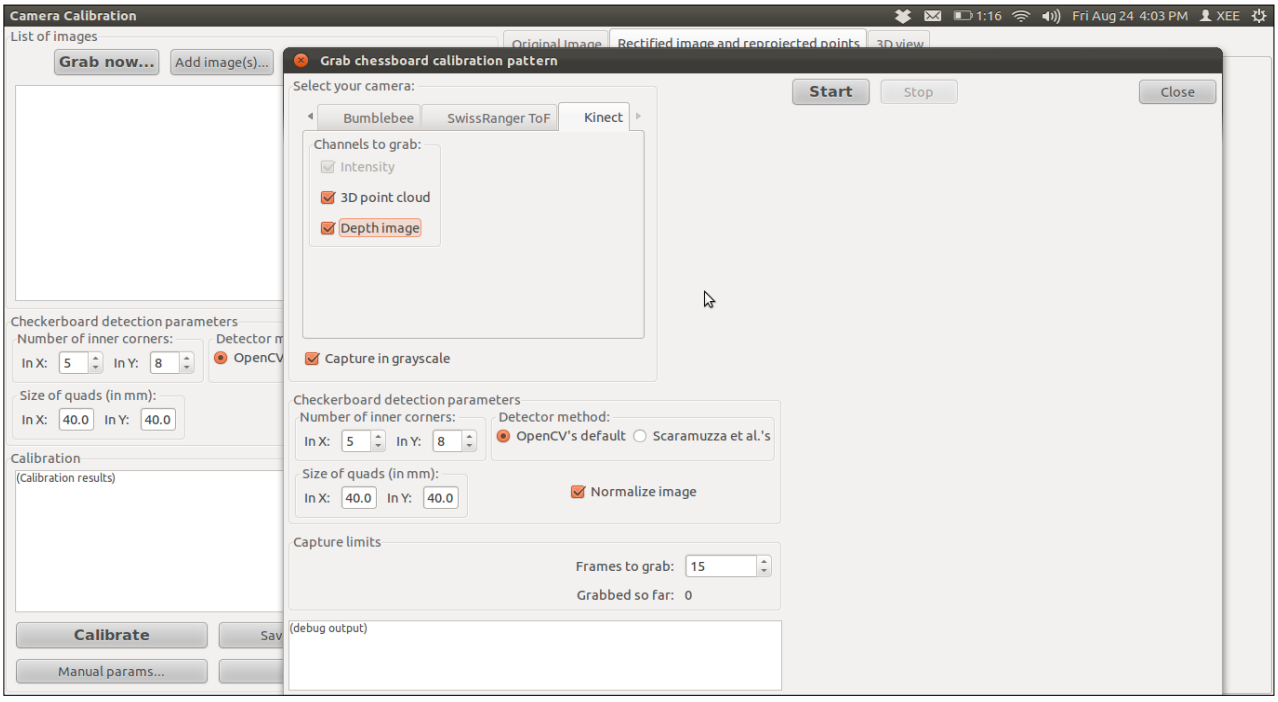

Figure 31: MRPT Camera Calibration_2

10 REFERENCES

[1] Uniform Sampling. Available at http://docs.pointclouds.org/trunk/classpcl_1_1_uniform_sampling.html#details (Accessed: 27 August, 2014)
[2] Distinctive image features from Scale Invariant Keypoints by David G. Lowe, University of British Columbia Vancouver, Canada
[3] Spin-Images: A Representation for 3-D Surface Matching by Andrew Edie Johnson, Carnegie Mellon University, Pittsburgh
[4] Unique Signatures of Histograms for Local Surface Description by Federico Tombari, Samuele Salti, and Luigi Di Stefano CVLab - DEIS, University of Bologna, Italy
[5] Improved SIFT – Features Matching for Object Recognition by F. Alhwarin, C. Wang, D Durrant A Gräser, University of Bremen
[6] 3-D free-form surface registration and object recognition by C. Chua and R. Jarvis, Int'l Jour. Computer Vision, vol. 17, no. 1, pp. 77-99, 1996
[7] Multi-Resolution Spin-Images by H.Q. Dinh and Steven Kropac, Stevens Institute of technology.
[8] KINECT DEPTH SENSOR EVALUATION FOR COMPUTER VISION APPLICATIONS, by M.R. Andersen, T. Jensen, P. Lisouski, A.K. Mortensen, M.K. Hansen, T. Gregersen and P. Ahrendt, Aarhus University
[9] Microsoft Kinect MSDN
[10] Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications, by Kourosh Khoshelham and Sander Oude Elberink , Faculty of Geo-Information Science and Earth Observation, University of Twente , Netherlands
[11] Removing outliers using a Statistical outlier removal filter. Available at http://pointclouds.org/documentation/tutorials/statistical_outlier.php (Accessed: 27 August, 2014)
[12] Ubuntu philosophy. Available at www.ubuntu.com/project/about-ubuntu/our-philosophy (Accessed: 27 August, 2014)
[13] Bjarne Stroustrup's homepage.
Available at http://www.stroustrup.com/#invention (Accessed: 27 August, 2014)
[14] TIOBE index for August 2014. Available at http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (Accessed: 27 August, 2014)
[15] QT project. Available at http://qt-project.org/ (Accessed: 27 August, 2014)
[16] Uniform Sampling. Available at http://docs.pointclouds.org/1.7.0/classpcl_1_1_uniform_sampling.html (Accessed: 27 August, 2014)
[17] A voxel-based lidar method for estimating crown base height for deciduous and pine trees by Popescu C. and Zhao Kaiguang. Available at http://www.sciencedirect.com/science/article/pii/S0034425707002532 (Accessed: 27 August, 2014)
[18] Inside a Kinect. Available at http://www.msxbox-world.com/xbox360/kinect/faqs/306/whats-inside-a-kinect.html (Accessed: July 2012)
[19] Point Cloud Images. Available at http://www.artistsguidetogimp.com/?page_id=145 (Accessed: 27 August 2014)
[20] Point Cloud Library has own website. Available at http://www.funnyrobotics.com/2011/03/point-cloud-library-has-its-own-web.html (Accessed: 27 August 2014)
[21] Mobile Robot Programming Toolkit. Available at http://www.mrpt.org/ (Accessed: 27 August, 2014)

[22] OpenGL Overview. Available at http://www.opengl.org/about/ (Accessed: 27 August, 2014)
[23] Keypoint Class Template Reference. Available at http://docs.pointclouds.org/trunk/classpcl_1_1_keypoint.html#details (Accessed: 27 August 2014)
[24] An As-Short-As-Possible Introduction to the Least Squares, Weighted Least Squares and Moving Least Squares Methods for Scattered Data Approximation and Interpolation by Andrew Nelen, TU Darmstadt
[25] Hacking the Kinect by Jeff Kramer, Nicolas Burrus, Florian Echtler, Daniel Herrera C., and Matt Parker
[26] Statistical Approach to Multi-scale pointcloud Processing, Ranjith Unnikrishnan, The Robotics Institute, Carnegie Mellon University, 2008
[27] Image Segmentation, Available Techniques, Developments and Open Issues, Tranos Zuva, Oludayo O. Olugbara, Sunday O. Ojo and Seleman M. Ngwira, Canadian Journal on Image Processing And Computer Vision Vol. 2, No. 3, March 2011
[28] Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography, Martin A. Fischler and Robert C. Bolles, Communications of the ACM Vol. 24 No. 6, June 1981
[29] Smoothening and Normal estimation based on polynomial reconstruction. Available at http://pointclouds.org/documentation/tutorials/resampling.php#moving-least-squares (Accessed: 27 August, 2014)
[30] A Method for Registration of 3-D Shapes, Paul J. Besl and Neil D. Mckay, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 14 No. 2, February 1992
[31] Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications, Kourosh Khoshelham and Sander Oude Elberink, Sensors 2012, 12, 1437-1454, February 2012
[32] Getting Started. Available at http://pointclouds.org/documentation/tutorials/basic_structures.php (Accessed: 27 August, 2014)
[33] Point Feature Histograms (PFH) descriptors http://pointclouds.org/documentation/tutorials/pfh_estimation.php#pfh-estimation (Accessed: 27 August, 2014)
[34] Estimating Surface Normals in a Pointcloud. Available at http://pointclouds.org/documentation/tutorials/normal_estimation.php#normal-estimation (Accessed: 27 August, 2014)
[35] Statistical Outlier Removal (Class template reference) http://docs.pointclouds.org/trunk/classpcl_1_1_statistical_outlier_removal.html (Accessed: 27 August, 2014)
[36] Towards 3DPointcloud based object maps for household environments by Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, Michael Beetz, August 2008
[37] Image Resampling by Jonathan Sachs, Digital Light & Color, 2001
[38] Performance enhancements of the Spin-Image Pose Estimation Algorithm, Adam R. Gerlach, March, 2007
[39] Surfaces generated by moving least squares methods, Lancaster P. and K. Salkauskas, Mathematics of Computation 37, 1981
[40] The approximation power of moving least-squares, Levin D, *Math. Comp*. 67, 1517-1531, 1998
[41] An As-Short-As-Possible Introduction to the Least Squares, Weighted Least Squares and Moving Least Squares Methods for Scattered Data Approximation and Interpolation by Andrew Nealen, Discrete Geometric Modeling Group, TU Darmstadt
[42] Approximate Moving Least-Squares Approximation: A Fast and Accurate Multivariate Approximation Method by Fasshauer, G. E, Curve and Surface Fitting: Saint-Malo, 2002

[43] Unsupervised human skeleton extraction from Kinect depth images by W. Shen, S. Xiao, N. Jiang, W. Liu, Proceedings of the 4th International Conference on Internet Multimedia Computing and Service , Pages 66-69, 2012
[44] Kinect based 3D object manipulation on a desktop display by M. Raj, S. H. Creem-Regehr, K. M. Rand, J. K. Stefanucci, W. B. Thompson, Proceedings of the ACM Symposium on Applied Perception, Pages 99-102, 2012
[45] Kinect in the kitchen: testing depth camera interactions in practical home environments by G. Panger, Proceedings of CHI '12 Extended Abstracts on Human Factors in Computing Systems, Pages 1985-1990, 2012
[46] Robotic vision for Bin-Picking applications of various objects by A. Pochyly, T. Kubela, M. Kozak, P. Cihak, 41st International Symposium on Robotics (ISR) and 6th German Conference on Robotics (ROBOTIK), Pages 1-5, 2010
[47] Vision based bin picking system for industrial robotics applications by K. Kim, J. Kim, S. Kang, Jaehong Kim and J. Lee, 9th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), 2012
[48] Development of structured light based bin picking system using primitive models by O. Jong-Kyu, K. Baek and S. Lee, IEEE International Symposium on Assembly and Manufacturing, Pages 46-52, 2009
[49] Vision-Based Bin-Picking: Recognition and Localization of Multiple Complex Objects Using Simple Visual Cues by K. Rahardja and A. Kosaka, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (Volume:3), Pages 1448-1457, 1996
[50] Model-based vision for robotic manipulation of twisted tubular parts: Using affine transforms and heuristic search by S. H. Wang, R. L. Cromwell, A. C. Kak, I. Kimura, and M. Osada. Proceedings of IEEE International Conference on Robotics and Automation, 1994
[51] A Handling System for Randomly Placed Casting Parts Using Plane Fitting Technique by T. Onda, H. Igura, and M. Niwakawa Proc. of IEEE/RSJ IROS Vol. 3, Pages 435-440, 1995
[52] RANSAM for Industrial Bin-Picking by D. Buchholz, S. Winkelbach and F. M. Wahl, 41st International Symposium on Robotics, Pages 1-6, June 2010
[53] Fisher, Perkins, Walker & Wolfart (2003). "Gaussian Smoothing" Available at http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm (Accessed: 02 February 2015)
[54] Laplacian of Gaussian. Available at http://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm (Accessed: 02 February 2015)