FIND VOLUME OF CUBOIDS USING TOF CAMERA

By

Nidhi T. Vaishnav (13CEUOS064)

A project submitted

In

Partial fulfilment of the requirements

For the degree of

BACHELOR OF TECHNOLOGY

In

Computer Engineering

Internal Guide	**External Guide**
Dr. C. K. Bhensdadia	Mr. Samarth Modi
Professor,	Project Manager,
Dept. of Computer Engineering	Tvarit Circuit Explore

FACULTY OF TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

DHARMSINH DESAI UNIVERSITY

April 2017

CERTIFICATE

This is to certify that the project work titled

Find Volume of cuboid using ToF camera

Is the bonafide work work of

Nidhi Vaishnav (13CEUOS064)

carried out in the partial fulfilment of the degree of Bachelor of Technology in Computer Engineering at Dharmsinh Desai University in the academic session

December 2016 to April 2017

Dr. C. K. Bhensdadia

Professor and Head,

Dept. of Computer Engg.

Faculty of Technology

Department of Computer Engineering

Dharmsinh Desai University

April 2017

ACKNOWLEDGEMENT

It is indeed an immense pleasure to express my gratitude to all those remarkable people who had significant impact on this work as well as on me.

I consider myself fortunate to be part of our institute Dharmsinh Desai University that provided me an opportunity to apply my theoretical knowledge through the project. This was the place of intense learning for me, not only in the computer engineering, but also on a personal level. I feel obliged in submitting this project as part of our curriculum.

First I would like to express my profound gratitude to my internal guide and adviser Dr. C. K. Bhensdadia (Professor and HOD of Computer Department, Dharmsinh Desai University) for his supervision, advise and invaluable guidance from the very initial stages of this project. Above all and most needed, he acknowledged my interest in Artificial Intelligence and provided me the opportunity to work on the subject of Computer Vision. I am extremely thankful for his unflinching encouragement and support in several ways.

I would like to express my earnest appreciation to my external guide Mr. Samarth Modi (Project Manager, Tvarit Circuit Explore), who has provided me the opportunity to work on this project. I heartily obliged for his timely advice, friendly demeanor, brilliant supervision and wonderful technical discussion on regular basis.

My appreciative regards to my fellow interns Aditya Kanade and Advait Bhatwdekar (EC department, Dharmsinh Desai University). I am fortunate enough to have such great colleagues, who always supported me and shared my difficulties during our project work.

I thank Texas Instruments for helping us throughout the project regarding to ToF camera system.

I also take this chance to thank respectfully to all my teaching faculties of Department of Computer Engineering at Dharmsinh Desai University. Without their teaching and sheer scholastic presence through four year of B. Tech. study, it would be impossible for me to carry out this project.

I must express my very profound gratitude to my beloved parents, Dr. T. K. Vaishnav and Dr. Parul Joshi Vaishnav, for their indissoluble support and guidance. They were the people who rooted the fundamentals of strength of my character, showing me the joy of intellectual pursuit ever since I was a child. I would like to thank my beloved sister Mauli Vaishnav, the first reader as well as proof reader of this report.

Nidhi Vaishnav

ABSTRACT

In industry, the objects are packed in the box of cuboid shape. To store or to transform them efficiently, we must have the knowledge of their volume. To measure the volume of every cube is a very tedious and error prone task. Here we propose a computer vision based system to find volume using Time of Flight camera. Here, given system captures cube with background and background is captured using Time of Flight camera and find the volume of cube in centimetre within 1 centimetre accuracy disregarding size of cube.

Keywords: Time of flight camera, Computer Vision, Cube, Volume

TABLE OF CONTENTS

Certificate .... **I**

Acknowledgements .... **II**

Abstract .... **III**

Table of contents .... **IV**

List of Tables .... **VIII**

List of figures .... **IX**

1 Introduction **1**

1.1 Purpose *1*

1.2 Scope *1*

1.3 Roles and responsibilities *1*

1.4 Overview *1*

2 Time of Flight Camera **2**

2.1 Introduction *2*

2.2 Theory of Operation *2*

2.3 Specifications *3*

2.4 Output data types *4*

2.4.1 Amplitude Map *4*

2.4.2 Phase Map *4*

2.4.3 Depth Map *4*

2.4.4 Point Cloud *5*

2.4.5 Advantages of PCD over other file formats *6*

2.5 Applications of ToF camera *7*

2.5.1 Gesture recognition applications *7*

2.5.2 Non-gesture recognition applications *7*

3 About System **8**

3.1 Product Perspective *8*

3.1.1 System Interface *8*

3.1.2 User Interface *8*

3.1.3 Hardware Interface *8*

3.1.4 Software Interface *8*

3.1.5 Communication Interface *8*

3.1.6 Memory Constraints *8*

3.1.7 Operations *8*

3.2 Product Functions *9*

3.3 User characteristics *9*

3.4 Assumptions and dependencies *9*

3.5 Constraints *9*

3.6 Functional Requirements *9*

3.6.1 Hardware Requirements *9*

3.6.2 Software Requirements *10*

3.6.3 Non-Functional Requirements *10*

3.6.3.1 Reliability *10*

3.6.3.2 Usability *10*

3.6.3.3 Integrity *10*

3.6.3.4 Availability *11*

3.6.3.5 Maintainability *11*

3.6.3.6 Portability *11*

4 Analysis **12**

4.1 Software Process Model *12*

4.2 Use case diagram *13*

4.3 Process Flow *14*

5 Algorithms **15**

5.1 PCD interface *15*

5.2 Cube capture *15*

5.3 2D approach *16*

5.3.1 Edge Detection *16*

5.3.2 Corner Detection *17*

5.4 3D approach *18*

5.4.1 Cube Extraction *18*

5.4.1.1 Depth Thresholding *18*

5.41.2 Background Subtraction *19*

5.4.2 Plane Segmentation *20*

5.4.2.1 Cube plane Segmentation *20*

5.4.2.2 Ground plane Segmentation *21*

5.4.2.2.1 From background PCD *21*

5.4.2.2.2 From cube with background PCD *21*

5.4.3 Height Measurement *23*

5.4.4 Length Breadth Measurement *24*

5.4.4.1 Min-Max algorithm *24*

5.4.4.2 Line density algorithm *25*

5.4.4.3 Plane density algorithm *26*

5.5 Hybrid approach with Top View algorithm *28*

5.5.1 3D plane to 2D plane image conversion *29*

5.5.2 2D contour *29*

5.5.3 Inter Edge Distance Average algorithm *30*

6 Implemented Design 31

6.1 Class diagram *31*

6.2 Sequence diagram *32*

6.3 Activity diagram *35*

6.4 Complete volume algorithm *39*

7 Testing 40

7.1 Testing *40*

7.1.1 Algorithm Testing *40*

7.1.2 Unit Testing *40*

7.1.3 Acceptance Testing *40*

7.2 Observations and Results *41*

7.2.1 Height Results *41*

7.2.2 Length Breadth Results *42*

7.2.3 Volume Results *43*

8 Problem in ToF Technology 44

8.1 Signal Quality 44

8.2 Superposition 44

8.3 Noise 45

8.3.1 Time variant Noise 45

8.3.2 Time invariant Noise 45

8.3.3 Signal Noise 46

8.4 System Error 46

8.5 Quantization Effects 46

8.6 Intensity related Error 46

8.7 Position Mismatch 47

9 Conclusion 48

9.1 Conclusion 48

9.2 Future work: ToF camera calibration 48

APPENDIX A 49

APPENDIX B 50

APPENDIX C 51

BIBLIOGRAPHY 53

LIST OF TABLES

1.1 Roles and responsibilities 1

2.1 Specifications of ToF camera 3

7.1 Height Results 41

7.2 Length Breadth Results 42

7.3 Volume Results 43

LIST OF FIGURES

2.1 3D ToF camera operation *2*

2.2 Components of ToF OPT8241 camera *3*

2.3 Output data types of ToF camera *4*

2.4 PCD file data format *5*

4.1 Use case diagram *13*

4.2 Process Flow *14*

5.1 2D edge detection *16*

5.2 2D corner detection *17*

5.3 Wrong corner detection *17*

5.4 Generic capture of cube *18*

5.5 captured cube *18*

5.6 Depth segmented cube *19*

5.7 Background subtraction *19*

5.8 Real extracted cube with its three detected planes *20*

5.9 Ground plane extracted from BG *21*

5.10 Ground plane segmentation from cube with BG *22*

5.11 Ideal cube with highlighted intersection line *23*

5.12 Top and ground plane to calculate height *23*

5.13 Planes look like lines after projection *24*

5.14 Line density algorithm *25*

5.15 Ideal line slider density graph *26*

5.16 real data line slider density graph *26*

5.17 Ideal plane density matrix *27*

5.18 capture orientation of cube *28*

5.19 High density top plane PCD *28*

5.20 Plane PCD converted to Image format *29*

5.21 contour on 2D *29*

5.22 Edge images of Top plane *30*

6.1 class diagram *31*

6.2 sequence diagram for find volume of cuboid using ToF camera *32*

6.3 sequence diagram for camera capture and cube extraction 32
6.4 sequence diagram for ground plane segmentation 33
6.5 sequence diagram for top plane segmentation..33
6.6 sequence diagram for height calculation 33
6.7 sequence diagram for length breadth calculation..34
6.8 Activity diagram for find volume of cuboid using ToF camera................................35
6.9 Activity diagram for camera capture and cube extraction.......................................36
6.10 Activity diagram for ground plane segmentation...36
6.11 Activity diagram for top plane segmentation 37
6.12 Activity diagram for height calculation..37
6.13 Activity diagram for length breadth calculation 38
8.1 Flying pixel effect along object contours due to a superimposed signal 45
8.2 Quantization effects with varying distance..46
8.3 Intensity related distance deviations due to varying object reflectivity 47
8.4 Position mismatch between two consecutive captures 47

CHAPTER-1

INTRODUCTION

In industry, the objects are packed in the boxes. Now it is possible that all these boxes are not of similar size. To store these boxes or to transfer them, we need to know the volume of box, so that we can determine required space, and store them efficiently. As we know, mostly these boxes are in the shape of cuboids.
It is very tedious and error prone task to find the volume manually for each box.

1.1 Purpose

Here we propose a computer vision based object oriented system using a Time of Flight camera to achieve this task. Our objective is to find volume of cuboids using 3D ToF camera within 1% error in volume.
We are using ToF camera, because we do not have depth information (z axis distance) in 2D capture. ToF camera provides us the real x, y, z co-ordinates of capture in metre with its intensity value.
In this system, the computer operator just needs to run the system with a single click and the system will capture the cube, find its dimensions and provide the volume of cube. Thus, this system reduces the human task to measure the dimensions of each box, and write down those data in one file.

1.2 Scope

Volume of cube using 3D ToF camera can be used for industrial purpose. It can be used in ware-houses where multiple boxes of various size are stored. It can be used for the transportation purpose, where we need to know the number of boxes which can be stored in each vehicle with maximum efficiency. This object-oriented application is developed in python language.

1.3 Roles and responsibilities

Name	Planning	Analysis	Design	Coding	Testing
Nidhi Vaishnav	✓	✓	✓	✓	✓

Table 1.1 Roles and responsibilities

1.4 Overview

The rest of the document deals about all the research work of this computer vision application. It also provides details about the algorithms applied in this application with their outcomes and issues.

CHAPTER-2

TIME OF FLIGHT CAMERA

2.1 Introduction

In this application, we are using time of flight camera to capture the cube.

Time of flight (**TOF**) describes a variety of methods that measure the time that it takes for an object, particle or acoustic, electromagnetic or other wave to travel a distance through a medium. This measurement can be used for a time standard to measure velocity or path length through a given medium, or to learn about the particle or medium. [1]

A **time-of-flight camera** (ToF camera) is a range imaging camera system that resolves distance based on the known speed of light, measuring the time-of-flight of a light signal between the camera and the subject for each point of the image. [2] The resolution of ToF camera is usually low (320*240 pixels) compared to 2D video camera.

2.2 Theory of Operation

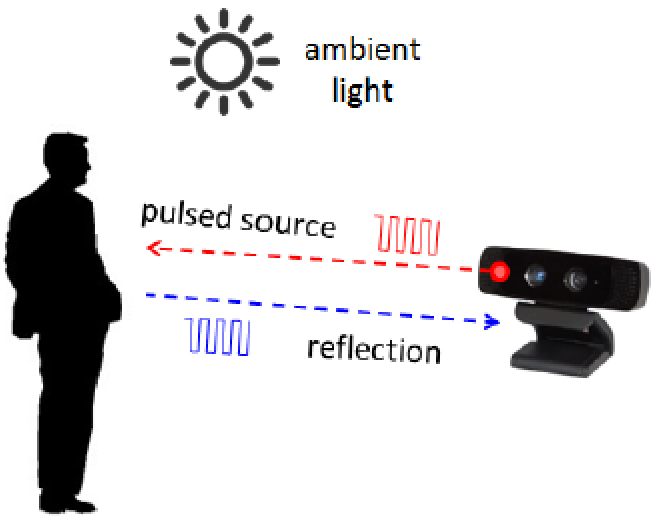


Figure 2.1 3D ToF camera operation

As shown in figure 2.1, 3D ToF camera illuminate the scene with modulated light source like solid state laser or LED operating in near infrared range (~850 nm). [3]

Than it observes the reflected light using some imaging sensor (lens) designed to respond to same spectrum, which converts photonic energy to electric current.

Based on the distance, the incoming light experience delay. This delay is very short. Due to this delay, the phase shift occurs between illumination and reflection, which is translated in distance. To detect this phase shift, the light source is pulsed or modulated by a continuous wave (CW) source, typically in form of a sinusoid or square wave. Square wave is more common as it can be easily realised using digital circuits.

2.3 Specification

Here we are using ToF camera model OPT8241 manufactured by Texas Instruments with following specifications. [4][5]

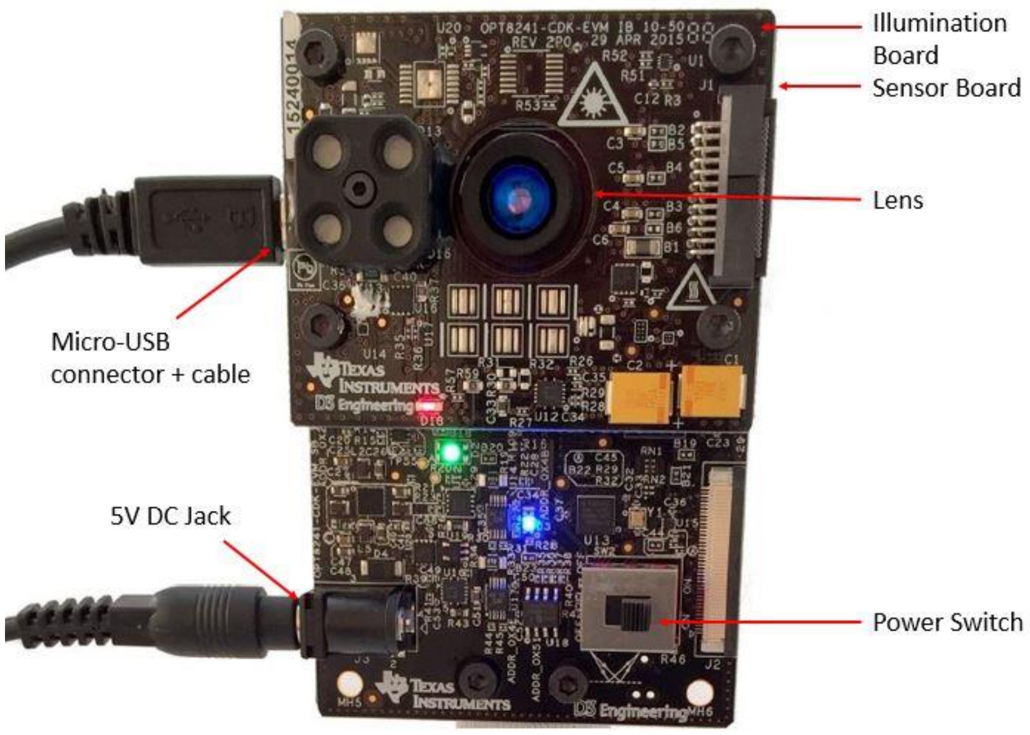


Figure 2.2 Components of ToF OPT8241 camera

Attribute	**Value**
Device	ToF Camera OPT8241 manufactured by TI
Resolution	QVGA 320*240 pixels
Frame rate	30 frames per second (up to 150 frame per second)
Working range	0m to 8m (Metrilus Long Range)
Interface	USB 2.0 Interface
Power supply	3A at 5V
Operating Temperature	0°C to 40° C (Ambient)

Table 2.1 Specification of ToF camera

2.4 Output data type

ToF camera can provide 3 types of outputs.

I. Amplitude Map and Phase Map
II. Amplitude Map and Depth Map
III. Point Cloud Data

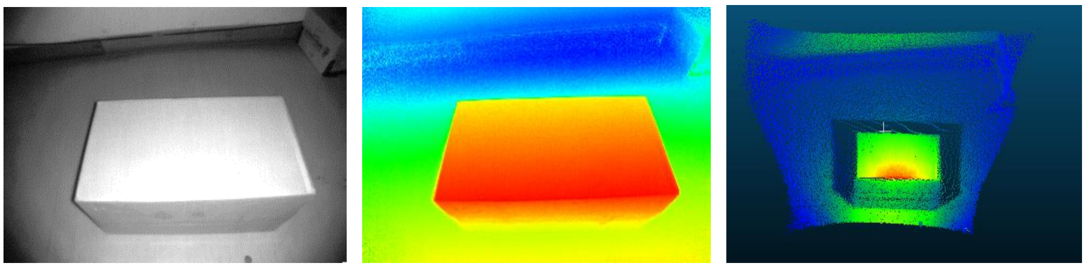

(a) Amplitude image (b) Depth Map (c) Point Cloud Map

Figure 2.3 (a), (b), (c) Output data types of ToF camera

2.4.1 Amplitude Map

The amplitude map represents the amount of reflective light captured by each pixel. This image is similar to the grayscale image captured by 2D camera which has intensity value for each pixel.

2.4.2 Phase Map

ToF camera measures the phase difference between illuminated ray any reflected ray. Phase map represents the relative phase measured at each pixel. This phase map is converted into depth map.

2.4.3 Depth Map

In 3D computer graphics, a **depth map** is an image or image channel that contains information relating to the distance of the surfaces of scene objects from a viewpoint. The term is related to and may be analogous to depth buffer, Z-buffer, Z-buffering and Z-depth. [6] The "Z" in these latter terms relates to a convention that the central axis of view of a camera is in the direction of the camera's Z axis, and not to the absolute Z axis of a scene.

Depth map for each pixel is internally calculated from phase map using following formula: [7]

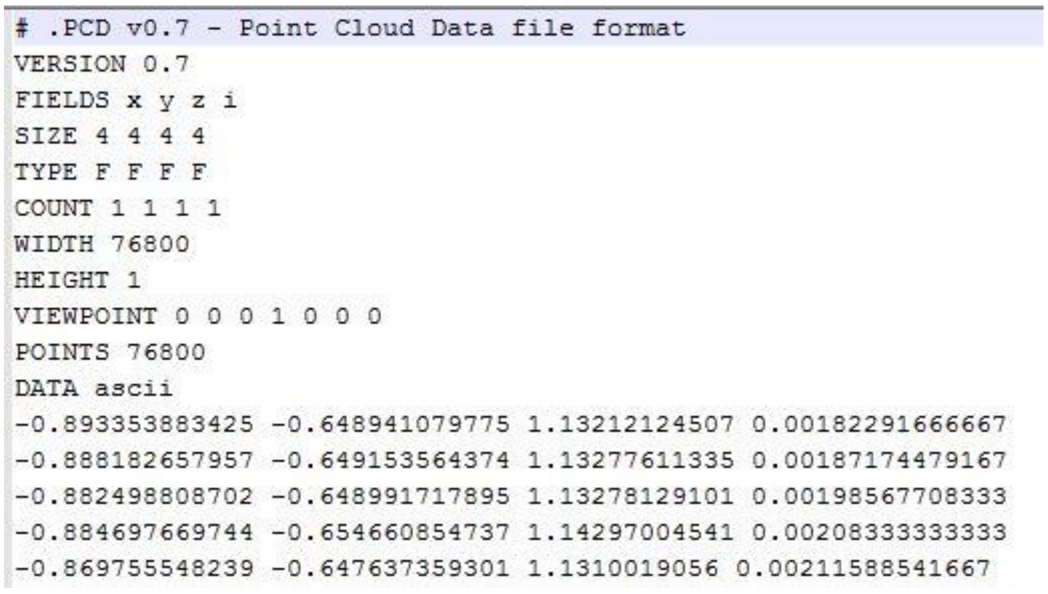
$$d = \frac{c}{4\pi f}\varphi$$

- c is speed of light ($3 * 10^8$ m/s)
- f is modulation frequency
- φ is phase shift

2.4.4 Point Cloud

A **point cloud** is a set of data points in some coordinate system. In a three-dimensional coordinate system, these points are usually defined by *X*, *Y*, and *Z* coordinates, and often are intended to represent the external surface of an object. [8]

The depth map is internally converted into point clouds co-ordinates by applying proper geometry transforms based on the camera optics properties such as, the Field of View (FOV), focal length, and lens intrinsic which accounts for optical distortion. With these co-ordinate values, intensity values of each pixel from amplitude image is added. Thus, here in Point Cloud Data (PCD) Format captured by ToF camera, we have *X*, *Y*, *Z* co-ordinate values and *I* as intensity as shown in below image. [9]



```
# .PCD v0.7 - Point Cloud Data file format
VERSION 0.7
FIELDS x y z i
SIZE 4 4 4 4
TYPE F F F F
COUNT 1 1 1 1
WIDTH 76800
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 76800
DATA ascii
-0.893353883425 -0.648941079775 1.13212124507 0.00182291666667
-0.888182657957 -0.649153564374 1.13277611335 0.00187174479167
-0.882498808702 -0.648991717895 1.13278129101 0.00198567708333
-0.884697669744 -0.654660854737 1.14297004541 0.00208333333333
-0.869755548239 -0.647637359301 1.1310019056 0.00211588541667
```

Figure 2.4 PCD file data format

Here, we can see the PCD file attributes.

VERSION: specifies the PCD file version

FIELD: Specifies the name of each dimension/field that a point can have

Here it has x, y, z co-ordinates (in metre) and intensity value (grayscale)

SIZE: Specifies the size of each dimension in byte. Ex.

- Unsigned character has 1 byte
- Unsigned short / short has 2 bytes
- Unsigned int / int / float has 4 bytes
- Double has 8 bytes

TYPE: specifies type of each dimension as a character. The current accepted types are,

- I - represents signed types int8 (char), int16 (short) and int32 (int)
- U - represents unsigned type uint8 (unsigned char), uint16 (unsigned short), uint32 (unsinged int)
- F - represents float type

COUNT: specifies how many elements does each dimension have. For example, *x* data usually has 1 element. By default, if *COUNT* is not present, all dimensions' count is set to 1.

WIDTH: specifies the width of the point cloud data set in the number of points. WIDTH has 2 meanings:

• It can specify the total number of points in the cloud (equal with POINTS) for unorganized datasets. In given PCD, width is 76800. (240 * 320 pixels)

• It can specify the width (total number of points in a row) of an organized point cloud dataset

HEIGHT: specifies the height of the point cloud data set in the number of points. Height has 2 two meanings:

• It can specify the height (total number of rows) of an organized point cloud dataset;

• It is set to 1 for unorganized datasets (Thus used to check whether a dataset is organised or not)

VIEWPOINT: specifies an acquisition viewpoint for the points in the dataset. This could potentially be used for building transform between different co-ordinate systems, or for aiding with features such as surface normal, that need a consistent orientation.

POINTS: specifies total number of points in the cloud. As of version 0.7, its purpose is a bit redundant.

DATA: specifies the data type that point cloud is stored in. As of version 0.7, two data types are supported: *ASCII* and *Binary.*

2.4.5 Advantage of PCD over other file formats

Some of the clearly stated advantages [9] of PCD files include:

• It contains co-ordinate values in real dimensions, which makes it easier to calculate dimensions

• The ability to store and process organized point cloud datasets, which is extremely important for real time applications, and research areas such as augmented reality, robotics etc.

• Storing different data types (all primitive data types are supported: char, short, int, float, double) allows the point cloud data to be flexible and efficient with respect to storage and processing. Invalid point dimensions are usually stored as NAN types.

• n-D histograms for feature descriptors are very important for 3D perception / computer vision applications.

An additional advantage is that by controlling the file format, we can best adapt it to PCL, and thus obtain the highest performance with respect to PCL applications, rather than adapting a different file format to PCL as the native type and inducing additional delays through conversion functions.

2.5 Applications of ToF camera

Application of ToF technologies are categorised into 2 parts,

I. Gesture recognition applications, which emphasise human interaction and speed

II. Non-gesture recognition applications, which emphasise measurement accuracy

2.5.1 Gesture recognition applications

It can be used in **gesture recognition** applications to translate human movement (face, hand, fingers, or whole body) into symbolic directives. These applications required fast response time, low to medium range accuracy and power consumption. Following are such applications.

- Game consoles, Ex. Xbox One
- Smart televisions, where channels are surfed using a hand wave
- Portable computer devices, where presentation can be scrolled using finger flickering

2.5.2 Non-gesture recognition applications

ToF technology can be used in non – gesture applications as well, where accuracy is very important. Following are such applications.

- **Automotive applications**, where ToF camera can detect objects and people as driving assistant.
- In **robotics**, it can be used to create quick map of surrounding, which helps robots to avoid obstacles during movement.
- During 3D printing, ToF camera can be used as efficient **3D scanner**.

CHAPTER-3

ABOUT SYSTEM

3.1 Product Perspective

This project finds the volume of a cuboid using ToF camera. All requirements describe a self-contained extension. The main goal of this project is to find volume by taking capture of cuboid within 1% error, hence reduce human effort.

3.1.1 System Interface

System interface is divided in 2 parts.

I. Cuboid Capture Interface
II. PCD interface

3.1.2 User Interface

No graphical user interface. To run the application, we are using "complete Volume" interface called by terminal.

3.1.3 Hardware Interface

ToF camera OPT8241 or any other ToF camera with specification of OPT8241 ToF camera. Here for the capture purpose, cuboid capture interface is used. For power supply, we are using 5V DC jack. To connect camera with PC USB 2.0 cable is used.

3.1.4 Software Interface

To capture PCD or depth image, in windows we can use Voxel Viewer software provided by TI. In Linux, for capture purpose, VoxelCLI is present. In given project, for capture purpose, VoxelCLI is internally used.
To see PCD file, Cloud Compare and Blender software are available.

3.1.5 Communication Interface

For the communication between camera and given application, USB 2.0 cable is used.

3.1.6 Memory Constraints

No memory constraints.

3.1.7 Operations

For Linux users, VoxelCLI must be installed.

3.2 Product Functions

The product can be divided in 4 parts.

I. Capture background and cube with background
II. Cube extraction
III. Find height of cube
IV. Find length breadth of cube

3.3 User characteristics

I. End User: General knowledge of ToF camera, and the way to connect ToF camera with computer is required.

II. Administrator: He must have the sound knowledge of Python language, image processing and ToF camera working mechanism. He should be able to operate Cloud compare, blender or any PCD viewer software.

3.4 Assumptions and Dependencies

For this application following are the required assumptions:

I. This application is developed for ToF camera, OPT 8241 created by Texas Instruments. Thus, the interface of this project is compatible with captures by ToF OPT 8241 camera model or similar ToF camera models.
II. Here during capture, the environment should remain static.
III. We are using 2 captures, background capture and cube with background capture. Here background should remain same during both capture.
IV. In cube with background capture, cube can be in any orientation, but the top plane of cube should be properly visible.
V. While using this algorithm without fresh capture, cube, cube with BG and extracted cube PCD should be available.

3.5 Constraints

For this application, we are using ToF camera with low resolution. (320*240 pixels). Hence, low resolution may affect the accuracy of volume.

3.6 Functional Requirements

3.6.1 Hardware requirements

- ToF camera OPT8241 (manufactured by Texas Instruments) or any similar model
- Micro USB connector
- 5V DC jack
- Computer with System storage 256 GB and above, RAM with 500 MB and above

3.6.2 Software requirements

R1. User request find volume of cube

Input: cube capture command

Output: Length, breadth and height of cube

R1.1 User request to capture cube

Input: cube capture command with fresh Capture = True

*Output:*cube with BG (Background), BG and extracted cube

R1.2 User to find volume

Input: Cube with BG, BG and extracted cube PCD

*Output:*Length, Breadth and Height of cube

3.6.3 Non – Functional Requirements

3.6.3.1 Reliability

Volume of cube by 3D ToF camera shall be robust enough to have a high degree of fault tolerance. The system should not crash in case of invalid input and shall identify the input and produce a suitable error message by using validation procedure of input.
This project shall be able to provide proper message and terminate gracefully from camera failure. It should able to recover from other hardware failures, power failures and other natural catastrophes.

3.6.3.2 Usability

- It should able to run with a simple command called from terminal.
- Any notification or error message by this application should be clear and polite.

3.6.3.3 Integrity

Only system administrator has the right to change system parameters. The system should be secure. Though end user should able to notify system for fresh capture or run application on available captures and location of PCD files for storage purpose.

3.6.3.4 Availability

The application shell be available to use when,

- camera is connected with computer using micro USB having this application
- Camera is on

3.6.3.5 Maintainability

The application will be developed by implementing the concept of modularity which in turn reduces the complexity involved in maintaining it. The administrator should have a sound technical knowledge about Python language, ToF camera software maintenance. Further enhancement will be undertaken by the developer or Texas Instruments.

3.6.3.6 Portability

It is a portable python application which ensures its adaptability of use on different operating systems supported by Python.

CHAPTER-4

ANALYSIS

4.1 Software Process Model

For Finding volume of cuboid using ToF camera project, the process model which has been used is agile software model. Agile software development is an umbrella term for several iterative and incremental software development methodologies.

The team of 4 members including project manager were working on this project. The approach towards the problem was not available. So, Acceptance Test Driven Development (ATTD) was used where various algorithms were tried to achieve the goal.

Time spent on each project task was less than two weeks. For each algorithm, the development iteration involved planning, analysis, design, coding, unit testing and acceptance testing. Here, coding and unit testing was worked parallelly. Analysis and design was done on the initial phase of algorithm, where no documentation was done. The things were discussed and designed using white board only. Here coding and testing was proceeded in parallel manner. Here the team member who is coder for some algorithm was not allowed to write test cases of the same algorithm, as it might reduce error detection probability. For the acceptance testing, the algorithms were tested on the captured cube. At the end of each algorithm the results were sent to the customer. If the algorithm couldn't provide satisfactory results, the approach towards problem was changed. Thus, respond to changes over following the plan was preferred.

As the ToF camera and 3D image processing domain were new to team members except manager, initial days of project were spent to learn about ToF camera, its operation, the point cloud format etc. To work on the centralized project, version control was used. Here each team member worked on the central copy of the project, and once their task was completed, the code was committed to central, from where other team members can get updated code.

For given project the interaction between team was high as they were working together, face to face communication was possible. The team members were supposed to write their daily target. Report of the work by each team member was written thrice a day. This report was shared among all other members via google drive. The interaction between team and customer (Texas Instruments) was on the weekly bases, during which project report in the form of a short presentation slide was mailed. And as the ToF camera was created by TIs, for any camera related problem, the contact had been made to TI.

4.2 Use case diagram

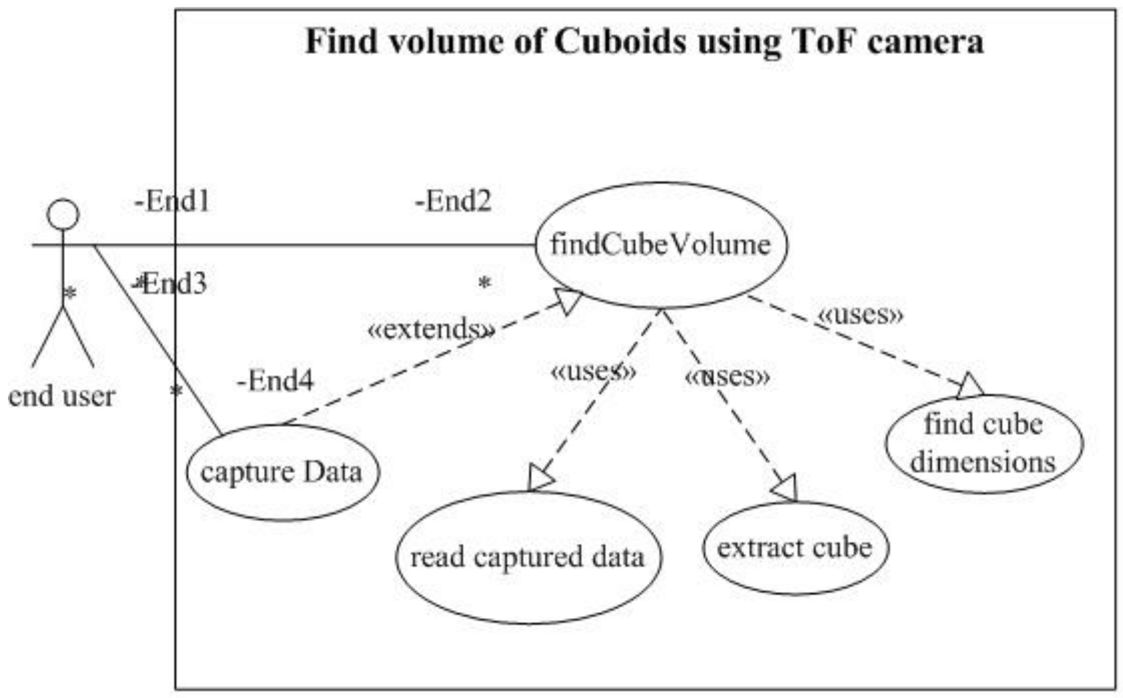


Figure 4.1 Use case diagram for find volume using ToF camera

4.3 process flow

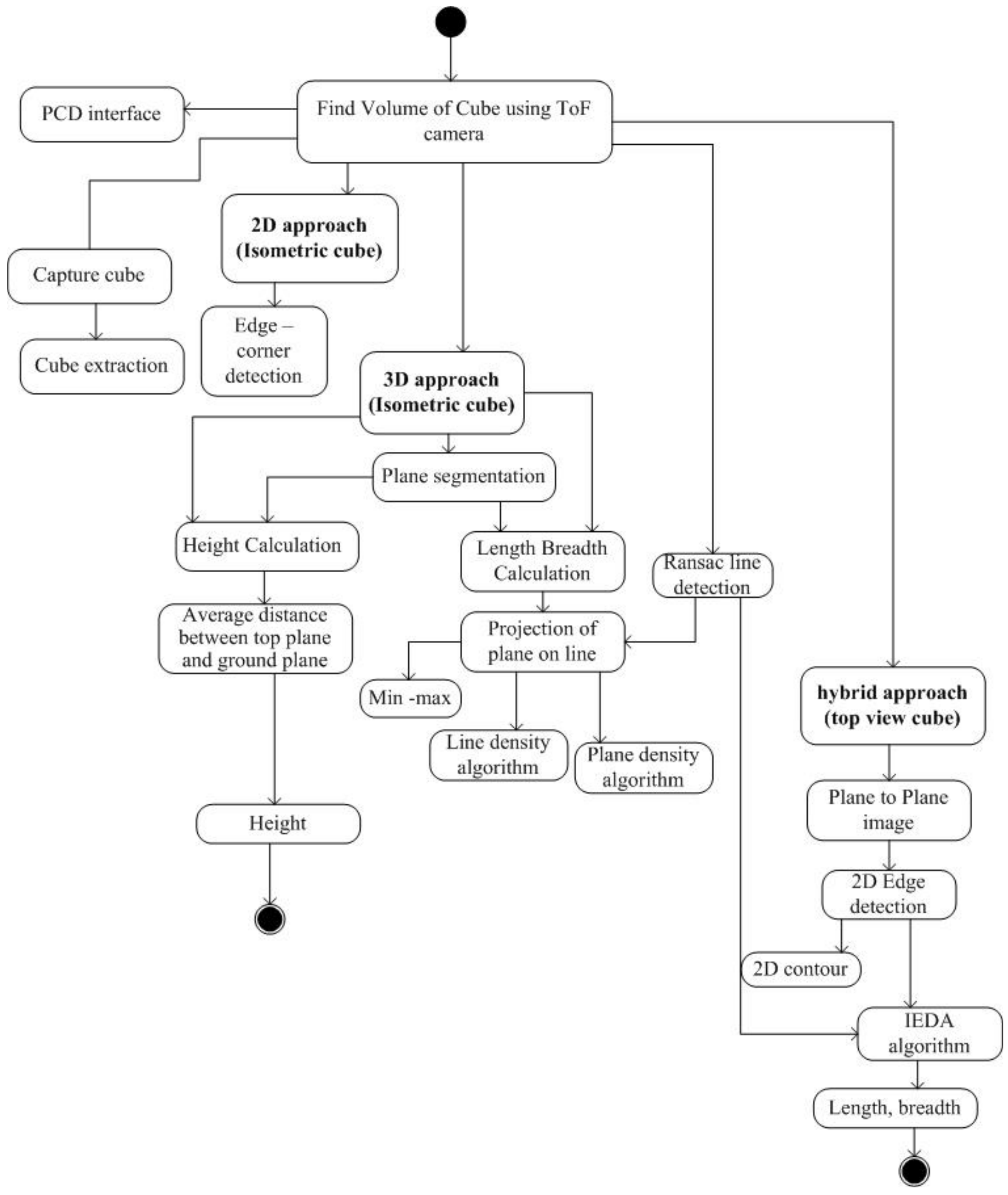


Figure 4.2 process flow to find volume of cuboid using ToF camera

CHAPTER-5

ALGORITHMS

Given application is implemented in Python programming language as it is a widely used high-level programming language, which can be used for general purpose programming. It is object oriented language with varieties of standard libraries.

To find the volume of cuboid, we needed basic functionalities first, such as read input data format, capture using ToF camera etc. After we had obtained the cube PCDs, we approached towards problem statement to find volume of cuboid from these captured PCD files. To view these PCD files, we have used available software such as Blender and Cloud Compare. In this chapter, we will describe various algorithms we have tried to get the volume of cuboid.

5.1 PCD interface

The ToF camera provides data in the form of PCD file, depth map and amplitude map. As we have seen previously, PCD file has advantage over other formats. So first we created a PCD interface, which could read PCD files and store its point into Point Cloud data list format. It has another function to write point cloud data list in PCD file. For this interface, we have used normal I/O file functions. To write appropriate headers we have stored information in a template file, and based on the number of points written in PCD file, this header changes.

5.2 Cube capture

The other basic functionality that was required was cube capture. We could capture the cube from Voxel Viewer application created by Texas Instruments. But this application was supported only in Windows operating system. And it was not possible to modify Voxel Viewer application so that its output capture will directly passed to our application. So, after collaborating with Texas Instruments, we wrote our own cube capture interface. For this purpose, we used inbuilt function of VoxelCLI after necessary camera attributes settings. With help of this capture interface, we were able to capture PCD files, moreover we were able to capture depth & amplitude images which we were planning to use during our 2D approach. Though these 2D images helped us in debug process of entire programming.

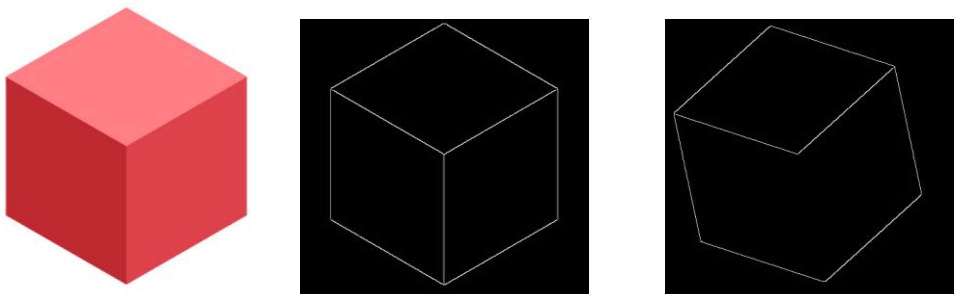
5.3 2D approach

As 2D image processing was familiar and our image had very low resolution which made 3D capture vary noisy, we first tried to find the volume of cube using 2D image processing. For 2D image processing, OpenCV is a well-established library. To find the volume of cube, we needed dimension of cube. To find dimensions, if we know the length of edge or the position of corner points, we can find the dimensions of cube.

5.3.1 Edge Detection

If we know the edge of cuboid, then we can measure its size and calculate the length. For edge detection, we tried to use popular canny edge detection function implemented by OpenCV.

Here we first convert cube image into grayscale image, based on some threshold we convert this grayscale image into binary image, and apply edge detection on this binary image.



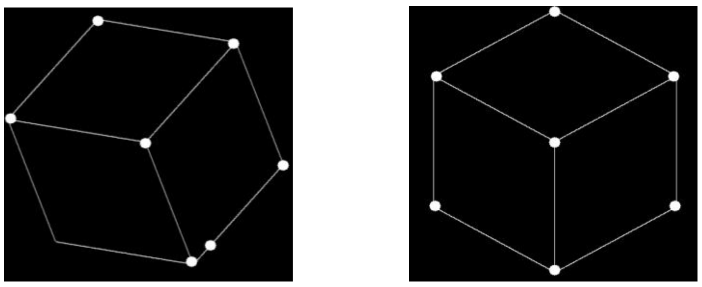
(a) (b) (c)

Figure 5.1 2D edge detection (a) Ideal cube image, (b) canny edge detection on ideal cube image (c) canny edge detection on rotated cube image

Here, for ideal cube for some threshold, we found all the edges. But our cube can be in any orientation. So, to make that case realistic with ideal cube, we rotated the original cube image. When we applied canny edge detection on this rotated image with previous thresholds, one edge was missing. (Figure 5.1 (c)) This is due to the requirement of different threshold during image binarization after rotation. For different capture, it is not possible to monitor the image and set different edge detection threshold. Thus, we were not able to use canny edge detection on the ideal cube image. So, we tried another approach.

5.3.2 Corner detection

In above figure 5.1 (c) one edge is missing, but even in this case also we can detect the corner. So, we tried to find the corners of image, we tried to use the Harris corner detection function implemented by OpenCV.



(a) **(b)**

Figure 5.2 2D corner detection (a) Harris corner detection on ideal cube image (b) Harris corner detection on rotated cube image

Here we can see that for the ideal cube, the proper corners were detected. But for rotated cube, the detected corners might not be the actual corner. The reason for this phenomenon was that, for straight edges the corners can be detected easily. But for the slanted edges, the function might consider the point on the slanted edge as the corner point as shown in figure 5.3.

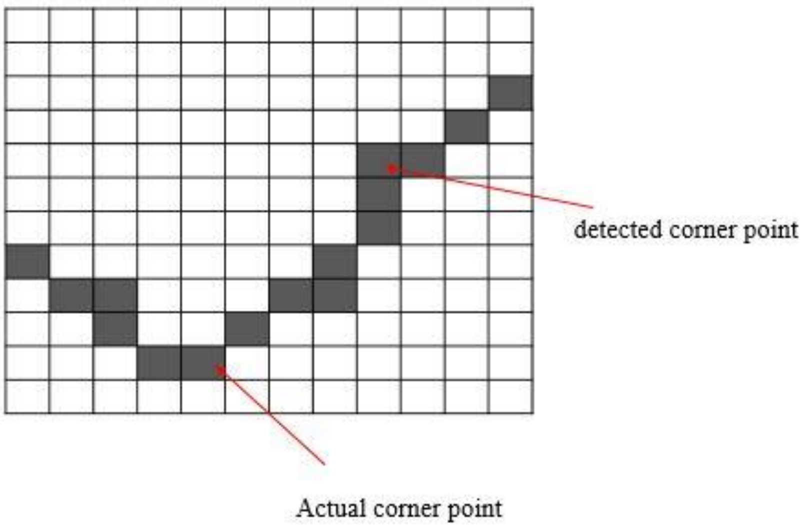


Figure 5.3 wrong corner detection

Thus, corner detection was not successful approach.

Moreover, we do not have the pixel to real dimension (metre) mapping. So even if we can get the dimensions using 2D approach in pixels, we have to map them in real dimensions. Which is not possible in 2D as we don’t have depth information. So, we dropped the 2D approach.

5.4 3D approach

For 3D approach, we are working on PCD file, which has x, y, z co-ordinate values along with intensity. Once PCD interface and capture interface was ready, we needed to capture the cube.

5.4.1 Cube extraction

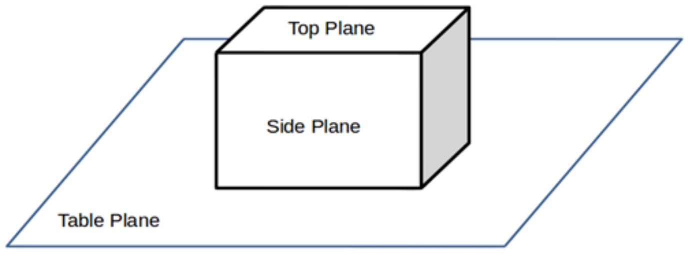


Figure 5.4 Generic capture of cube

Figure 5.4 represents the usual isometric setup of the cube and scene. Since background is the fundamental parts of any object capture, we need to first extract cube from scene before we can find its dimension and eventually volume.

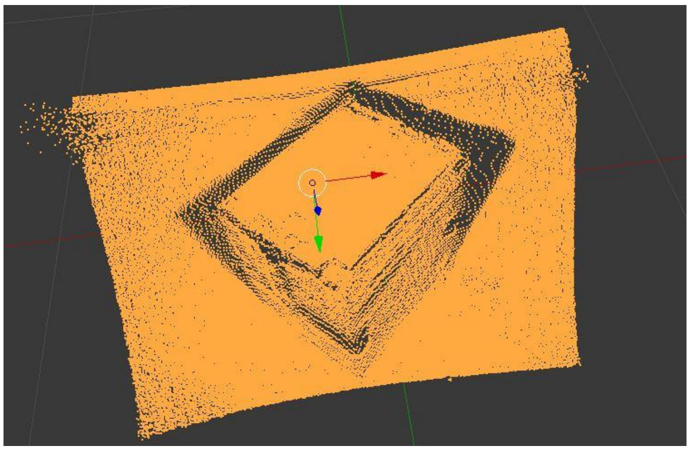


Figure 5.5 Captured Cube

Figure 5.5 shows how noisy the captured point cloud is. To segment the cube from this scene, we applied two methods. First method was intuitive. The cube should be at a different depth than the background, right? So, we just had to segment the PCD based on its depth, and in certain range we would have our cube.

5.4.1.1 Depth Thresholding

This method involved extracting only certain points that lied within a specified depth range. However, results of this method were not as expected; parts of table, on which the cube was standing, were also being counted as part of the cube.

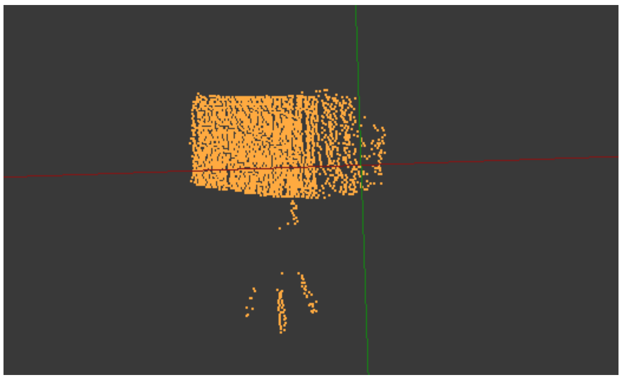
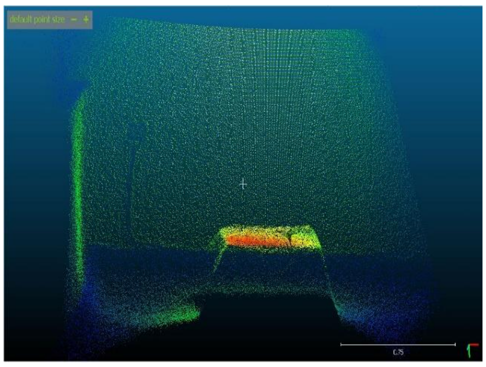


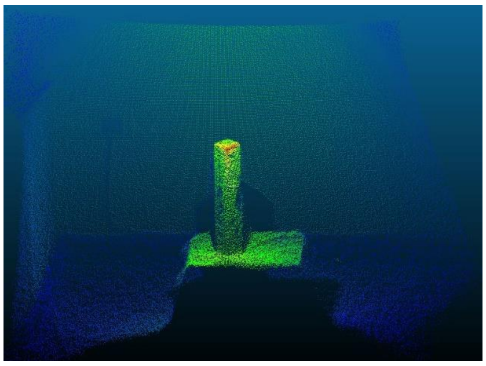
Figure 5.6 Depth Segmented cube

As can be seen from figure 5.6, the tripod onto which the cube was placed also got extracted. Because it was within the region of depth threshold. Beside this issue, we have to change depth threshold manually for each capture as for different cube or for the different position of same cube, the depth value will change, which is not feasible for the end user.

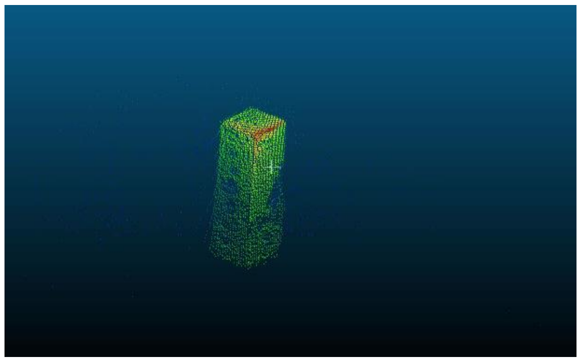
5.4.1.2 Background Subtraction



(a) Cube capture



(b) Background capture



(c) Extracted cube

Figure 5.7 Background Subtraction

Figure 5.7 (a) and 5.7(b) show the PCD maps of two scenes:

I. Cube with background, and

II. Same background, but without cube

Notice the differences in depth at the centre of both the figures. We filter these pixels that exhibit significant difference in depth, and consider them as part of the cube. Here the crucial point is that once we have captured background once, we are set for as many cubes as we want as long as the scene remains constant.

5.4.2 Plane segmentation

Planes are the constituent components of a 3D cuboid. Thus, to extract any information from the cuboid with reliable accuracy, we had to extract the cuboid's planes. For our algorithms, we extracted four planes- three of the cuboid, and of the base it would be standing on.

5.4.2.1 Cube plane segmentation

For most cases, cuboids will be kept in an isometric perspective relative to camera. To extract the visible planes, we used plane segmentation module of PCL library that used **Random Sample Consensus** (RANSAC) algorithm. Fig 5.8 shows the cube in black, and its detected planes in orange.

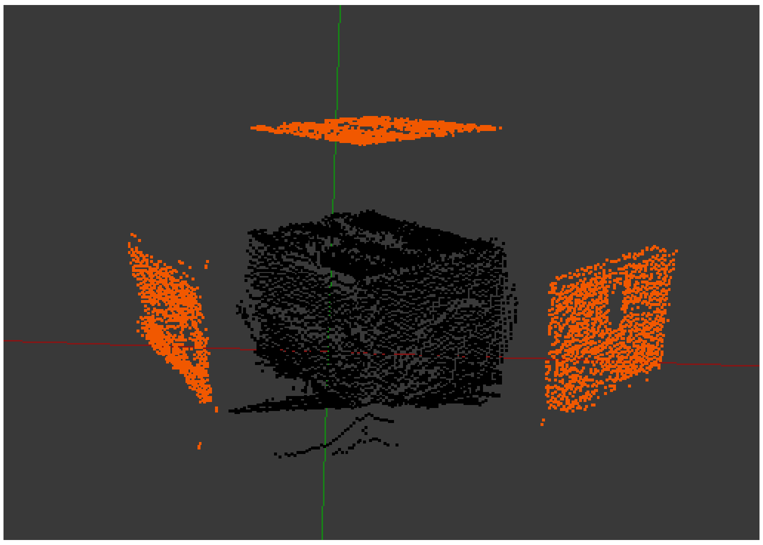


Figure 5.8 Real Extracted Cube with its Three Detected Planes

RANSAC algorithm can find a plane with maximum number of points in the plane. Details of RANSAC are explained in Appendix A. Once a plane was extracted from the cube, we removed that plane's points from the cube, and ran RANSAC again. Now we have two planes extracted. Doing the same process again, we can extract the third plane.

5.4.2.2 Ground plane segmentation

Since camera's position was fixed, to extract the base plane, we simply had to apply a PCL's plane segmentation module on the ground.

5.4.2.2.1 From background PCD

Here we are performing some depth thresholding, so that we get exactly bottom part of cube. After that we are applying PCL's plane segmentation module on the ground.

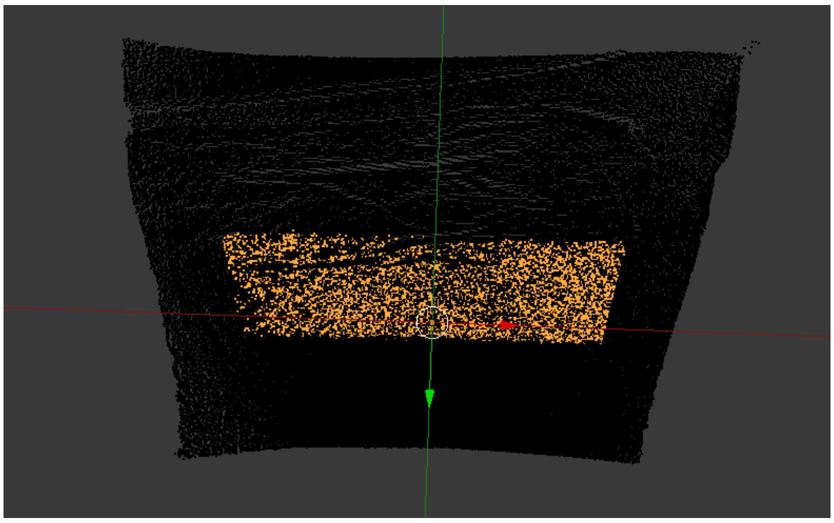
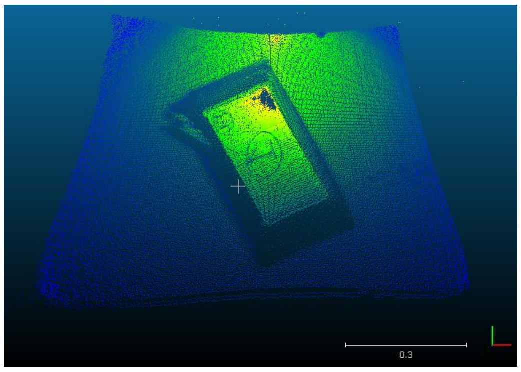


Figure 5.9 Ground Plane Extracted from BG

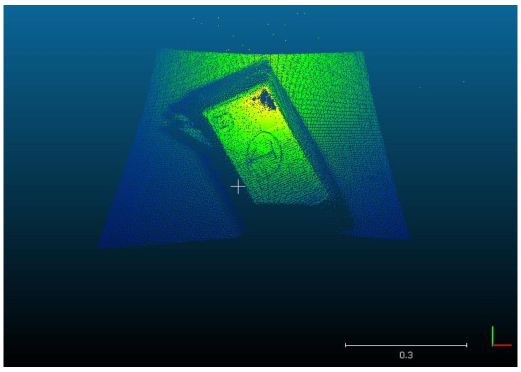
Once we have captured background once, we are set for as many cubes as we want as long as the scene remains constant. The problem with this approach was that, we had to add depth threshold manually for the first capture background and probable position of cube. But the bigger problem here was the position mismatch, which is described in the Problem with ToF camera chapter.

5.4.2.2.2 From cube with Background PCD

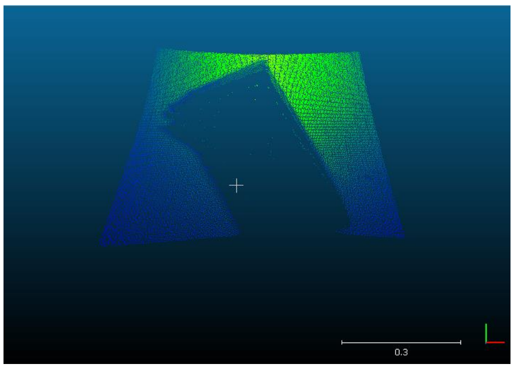
After failure of ground plane extraction from background plane due to position mismatch, it cross to our mind that plane is infinite. So, if we can extract the ground plane from background with cube PCD itself, then for height calculation, we wouldn't need background PCD. Here we are first creating filtered PCD by removing border of PCD, because at border PCD data have outliers, and PCD might be curved. Which might give wrong ground plane.
Now, we are removing extracted cube from this filtered PCD. So now we have the ground PCD. Now, we are running RANSAC plane segmentation on this ground PCD to receive ground plane.



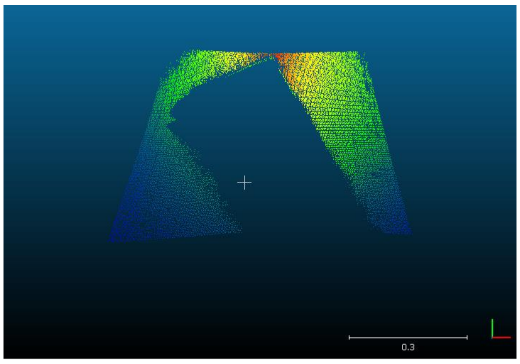
(a) Cube with background PCD



(b) filtered cube with background PCD



(c) Ground only PCD



(d) Extracted ground plane PCD

Figure 5.10 Ground plane segmentation from cube with BG

Here, for this approach, we don't need to perform depth thresholding anymore. AS we are extracting the ground around actual plane. And as plane is infinite, the plane equation will remain same even if points are not exact bottom part of cube.

In conclusion, using RANSAC on background PCD and extracted cube PCD, we were able to obtain four planes for our further analysis.

5.4.3 Height Measurement

If we think for a second, the top plane and the base plane, the table plane, should ideally be parallel with one another. Using that to our advantage, we can use all the points of top plane and find their distances from the table plane to get a massive data set that we can average for estimating height. Luckily, RANSAC algorithm, apart from returning points belonging to a single plane, is also capable of returning plane equation of the detected plane. This made height calculation a lot easier. Figure 5.11 shows an ideal cube with connecting lines of top and table planes highlighted.

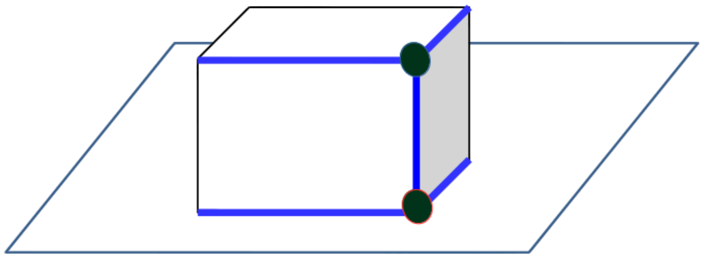


Figure 5.11 Ideal cube with highlighted intersection line

Height was the easiest of all dimensions, because number of points we had to measure height were in the order of thousands. The process for height calculation is as follows:

I. From the plane of the cube, find the top plane (parallel plane with ground plane)

II. Find the average distance of detected points in the top plane from ground plane

III. Find the average distance of detected points in the ground plane from top plane

IV. Average both the results to get height

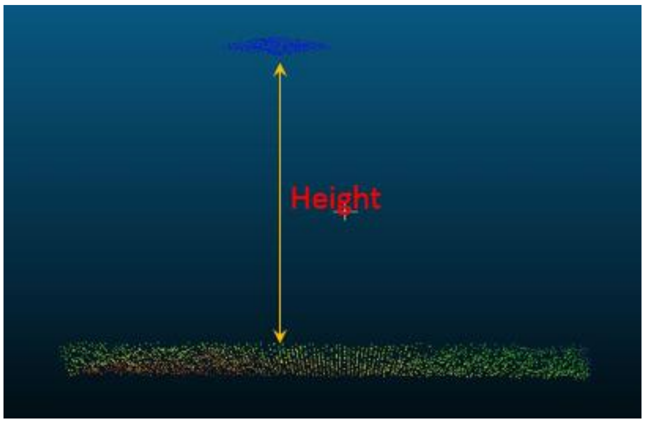


Figure 5.12 Top and ground plane to calculate height

Because of the extensive averaging of distances (which reduced noise considerably) to measure height, calculated height was found to be the most reliable of all three dimensions with less variance in error.

5.4.4 Length breadth measurement

Compared with calculation of height, which had ground plane as reference to the top plane, the side planes didn't have such reference. So, calculating these dimensions required a different approach. The first one we tried was to project the planes on their corresponding intersection lines with the table plane. This allowed for using line detection algorithms to calculate their dimensions.

5.4.4.1 Min -Max algorithm

Figure 5.13 shows the two side planes projected on the table plane. Once a plane is projected, we can find the extreme points in the line formed by its extreme (x, y) coordinate points. Clearly, a point with maximum x coordinate will be an end point of the line. Similarly, min x coordinate point and min y coordinate point would be the other end point. The intersection point between these two lines can be identified by min y point.

Once, these end points have been identified, we simply use Euler's distance formula to find the length of the line.

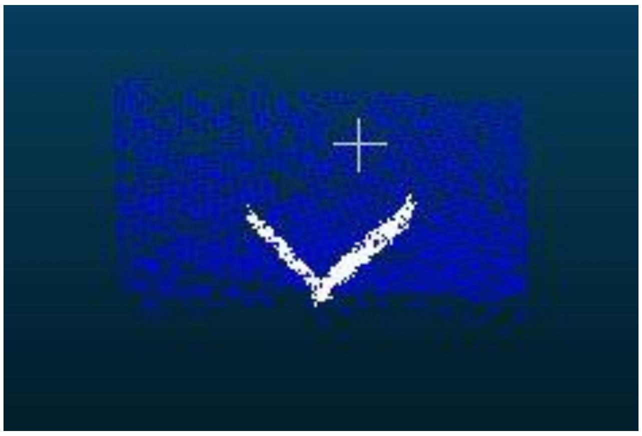


Figure 5.13 Planes look like lines after projection

However, there is a serious issue with this algorithm. Nowhere, we have considered the possibility that the end points detected might not be the actual end points of the line, but just noisy outliers. This lead to some serious errors in our answers, as often we would encounter noisy outlier points that weren't part of the plane at all. Figure 5.14 illustrates this for a real projected plane line.

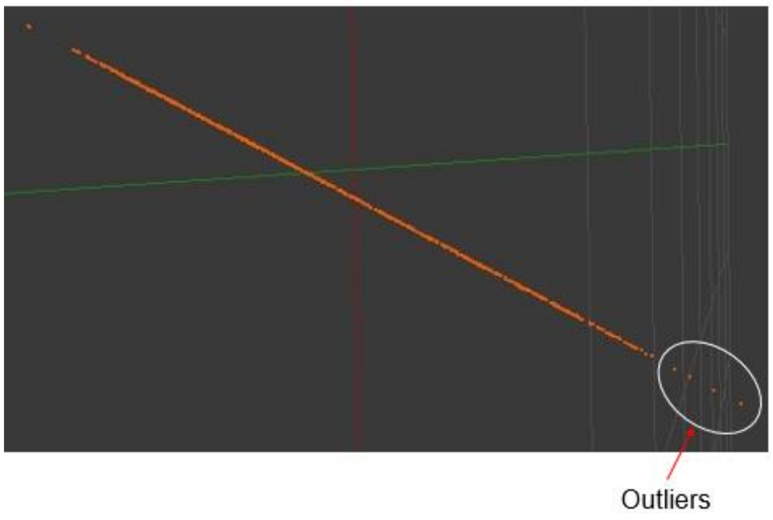


Figure 5.14 Line Containing Outliers (not part of plane)

5.4.4.2 Line density algorithm

The min-max approach didn't work because of the outliers. To circumvent this issue of outliers, we tried the line density method. If we project all the points of a plane on its edge line, what sort of density should we expect of that line? Ideally, density of the line should be constant at all points in the line. But in presence of outliers, line density would be low at the corners.

So first we projected the points of the plane on one of its edges. Next, we divided the projected line was divided into small sections and density of each section was calculated. Figure 5.15 shows the density of an ideal plane projected on line. In this figure one can easily point the end of line. It is the point where line density starts to drop.

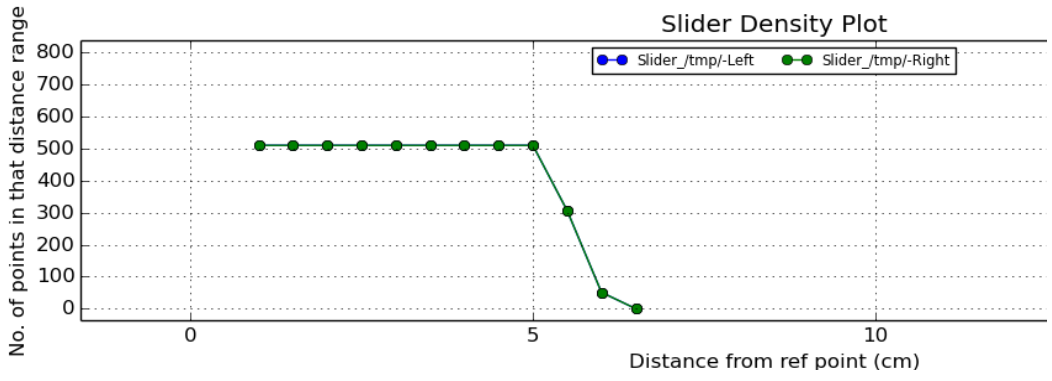


Figure 5.15 ideal line slider density graph

Now consider the graph of real plane.

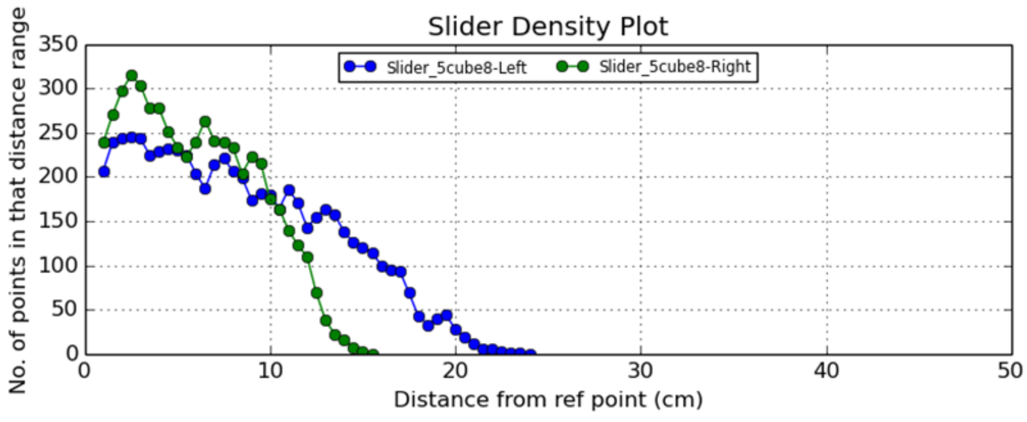


Figure 5.16 real data line slider density graph

The right line (green) is 10 cm long. However, from this graph, it is impossible to identify the location of its endpoint. Not only does plane's density starts to fall off far before its end point, there are multiple locations where density starts to fall, making a decision regarding true length ambiguous.

5.4.4.3 Plane density algorithm

Due to ambiguity in results of the line density matrix, a similar approach of density was applied to plane. In this approach, first we find the plane density of plane, and we project only those regions where plane density is high.

Here, the plane was divided into smaller planes, and points in each sub-plane were counted as shown in Figure 5.17. Ideally, a column with only a few sub-planes with respectable density of points would mean that that column consists of outlier points. Since we had a stable algorithm for calculating height, the number of columns were taken as per the size of height, we estimated that if more than 80% of rows in a column are empty, it means that column can be considered containing outliers and can be ignored.

10	12	14	3
15	12	15	0
0	11	17	0
11	11	16	0
0	10	16	0
17	0	15	1
14	0	11	0
11	12	0	0

Figure 5.17 Ideal Plane Density Matrix

From Figure 5.17, we can conclude that the last column consists of outliers since only two sub-planes have points in them. Thus, it can be rejected in dimension calculation.

Once the outlier columns are rejected, the dimension was calculated with the formula:

$$dimensions = correct\ column\ count \times sub\ plane\ resolution \text{ (in cm)}$$

The results however were not satisfying. It was observed that the calculated dimensions were almost always lesser than the actual dimensions. This was due to innate limitation of ToF technology.

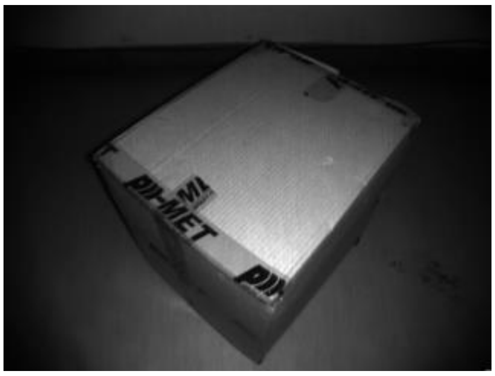
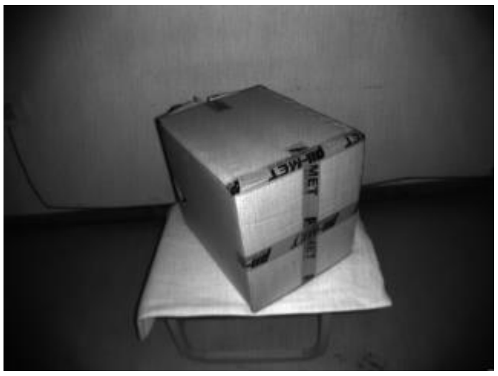
Consider two planes A & B: Plane A is kept almost parallel with the camera plane and Plane B is more inclined to the camera plane. It is obvious that Plane A will have more points than Plane B. Also, due to variation in depth point density, number of points in Plane B will reduce for columns away from the camera. Due to this, even valid columns at the edge were getting rejected because of sparse point density.

5.5 Hybrid approach with Top View algorithm

Till now the cube was placed isometric view. This however resulted in sparsely populated planes. To increase plane density, we switched to top plane focused captures and algorithms associated with it. Figure 5.18 (a) represents isometric view, while Figure 5.18 (b) show top focused capture, to increase top plane's density.

Here conversion from isometric view to top focused view won't be any problem, as

In industry, the position of camera is at some height from the object.

(a) Isometric view of scene **(b) Top focused view of scene**

Figure 5.18 capture orientation of cube

In the capture of isometric view, each plane only consisted of 3-5% of total points in the PCD. However, with the top focused capture, we could obtain 20% points in the top plane. This lead to less number of outliers in the top plane, and its edges remained intact and distinguishable. Figure 5.19 shows the extracted plane. Notice how edges are straight and clearly identifiable.

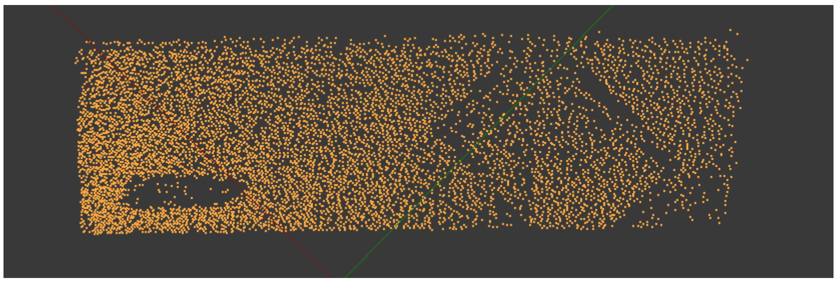


Figure 5.19 High Density Top Plane PCD

5.5.1 3D plane to 2D plane image conversion

Once the top plane was extracted from the cube, it was converted to a 2-D image. This was done so that we can use some standard OpenCV functions for image processing to compute the plane's area.

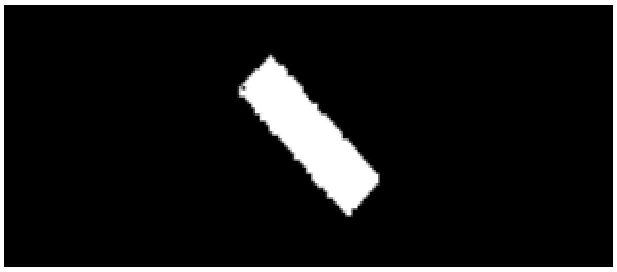


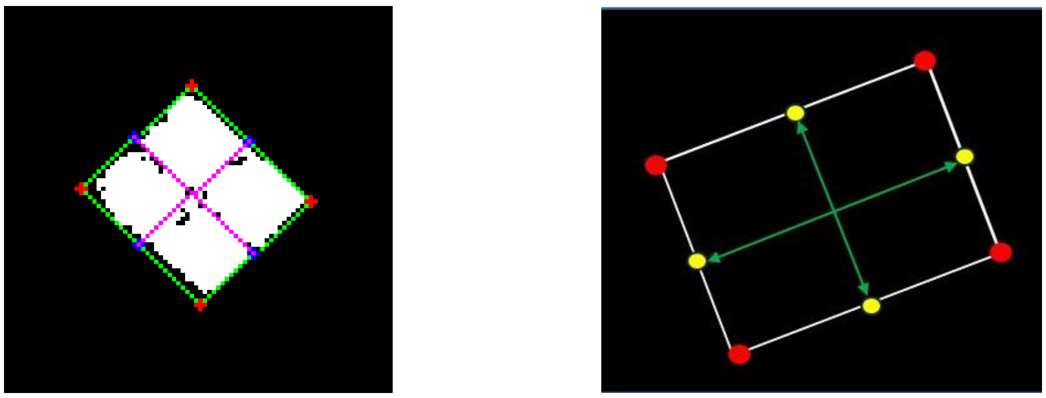
Figure 5.20 Plane PCD Converted to Image Format

To convert plane PCD into image, we took each point on the plane, mapped them within 1 cm. Thus, each pixel in the pixel is equivalent to one centimetre square box.

Here, in figure 5.20 the white area is plane. Two algorithms were applied on the 2-D image to get the dimensions/area.

5.5.2 2D contour

Contour is a boundary surrounding an object. Figure 5.21 shows an ideal box and its contour. We used Open-CV library for computing contour on our image. Once we have surrounding boundary of an image, we can calculate its length by measuring distance between mid-points two different lines in the contour.



(a) **(b)**

Figure 5.21 (a)Ideal Box and its Computed Contour with OpenCV Function

(b) Contour detected in real top plane

Here we are not approaching for corner points as we can see, the corner of the plane image might not be accurate.

5.5.3 Inter Edge Distance Average algorithm

It was clear that any algorithm that attempted to calculate dimension without participation of maximum number of points would fail because of non-ideal plane. So, we developed a new algorithm Inter Edge Distance Averaging (IEDA). This algorithm is as follows:

I. Apply canny edge detection on the plane image (Open-CV canny edge detection)

II. Estimate line equation of all the edges via a line fitting algorithm

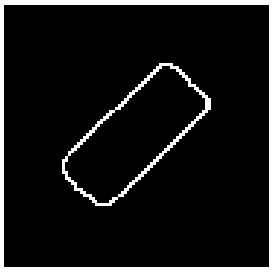
III. Group parallel edges together

IV. Select one edge, and find average distance of its points from its parallel edge

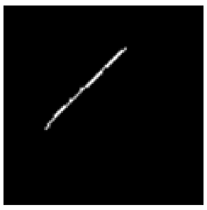
V. Do this for the ether edge in the group

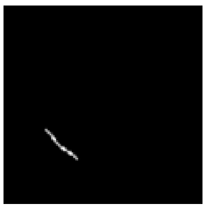
VI. Average both the results to get the dimension

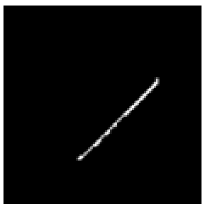
VII. Do same for other pair of parallel edges for the dimension

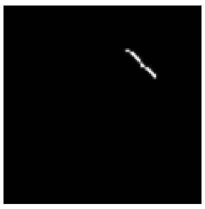


(a)









(b) (c) (d) (e)

Figure 5.22 (a) edge image of top plane, (b), (c), (d), (e) Detected edges from edge image of top plane

One can clearly see the resemblance of the algorithm to the height algorithm. Averaging is done in all possible ways to reduce any effect of noise. The results were tabulated and it was found out that the maximum error was restricted to ±1cm.

CHAPTER-6

IMPLEMENTED DESIGN

6.1 Class Diagram

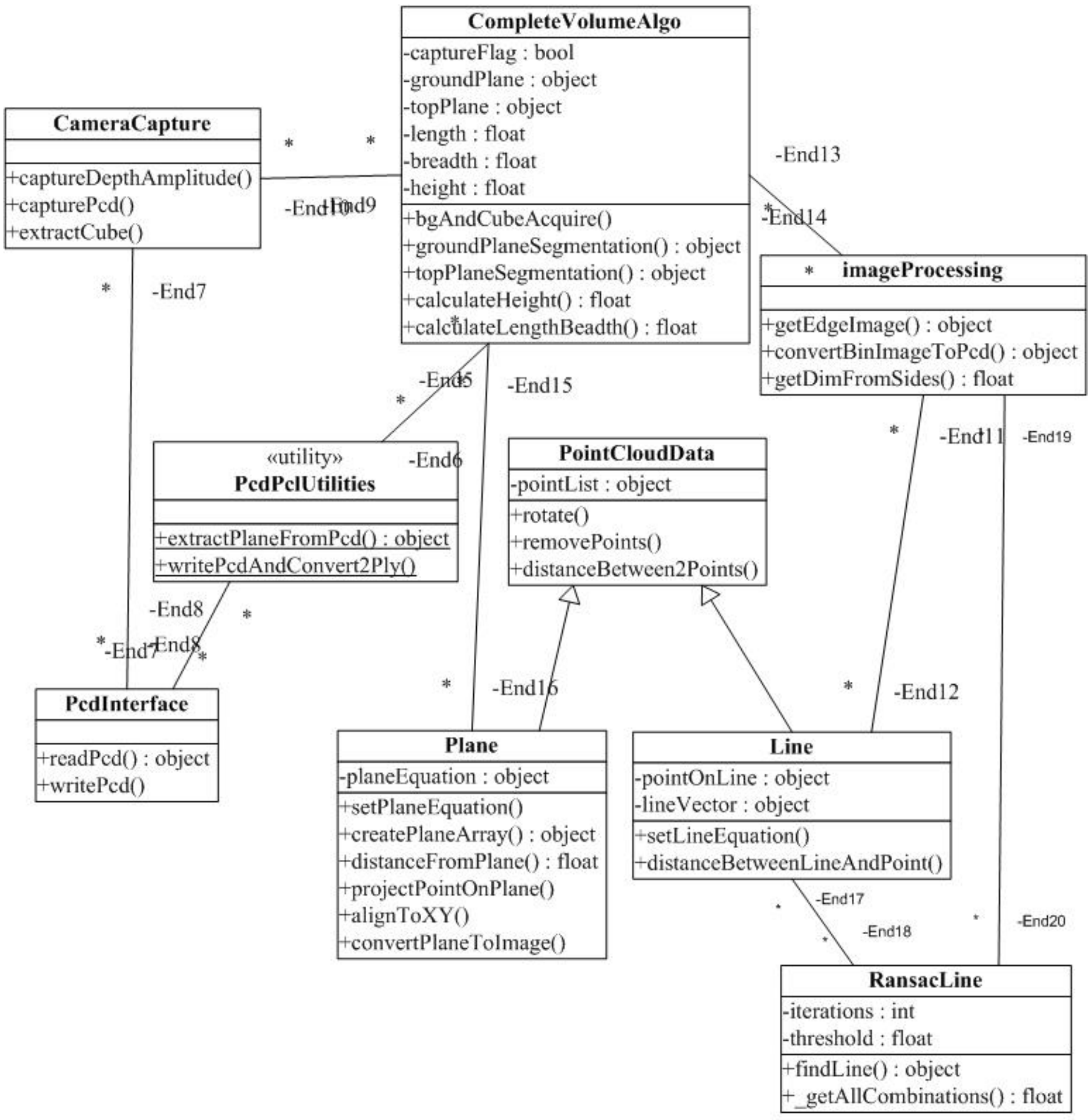


Figure 6.1 Class diagram for Find Volume of Cuboids using ToF camera

6.2 Sequence Diagram

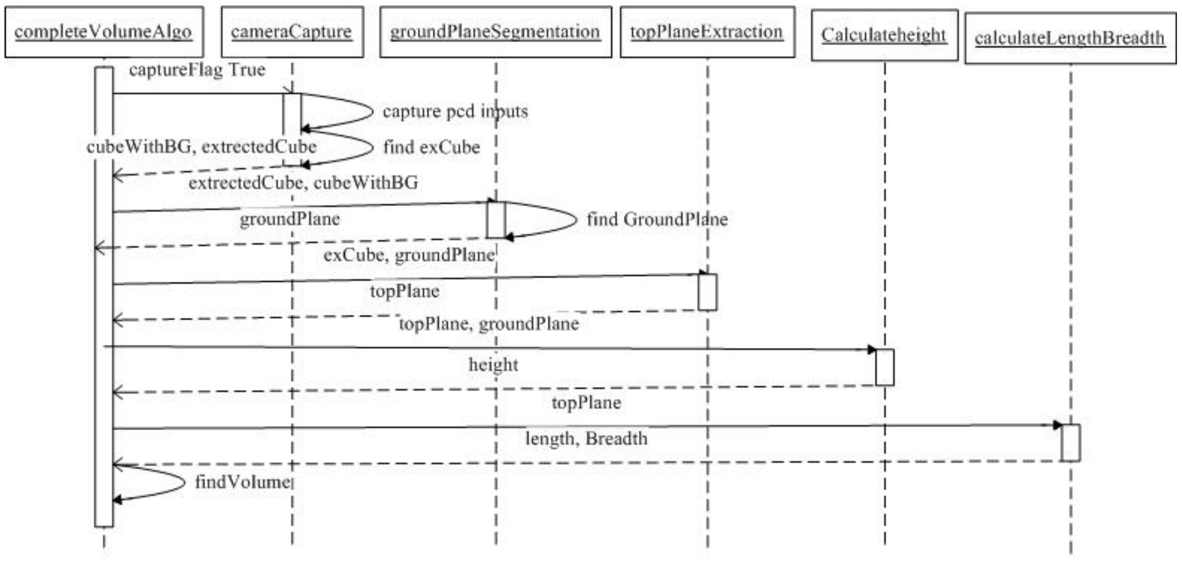


Figure 6.2 Sequence Diagram for find Volume of Cuboid using ToF camera

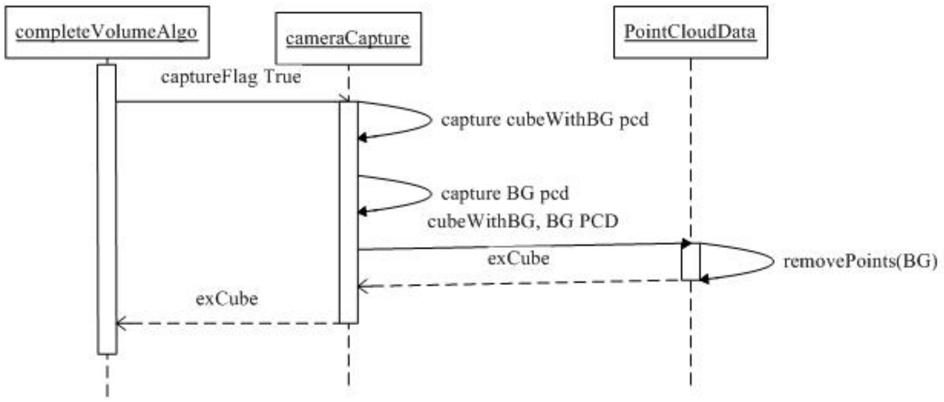


Figure 6.3 Sequence Diagram for Camera Capture and cube extraction

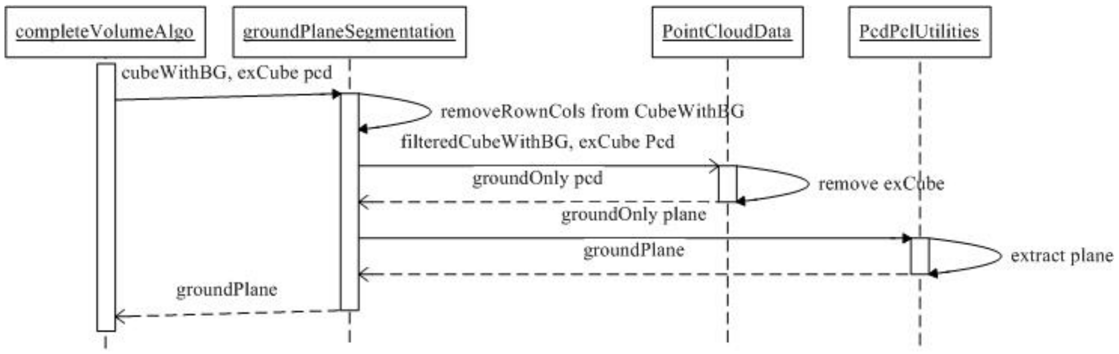


Figure 6.4 Sequence Diagram for Ground Plane segmentation

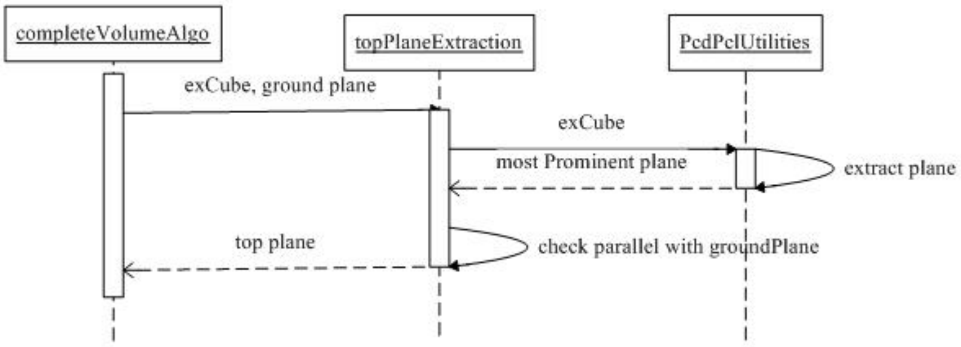


Figure 6.5 Sequence Diagram for Top Plane segmentation

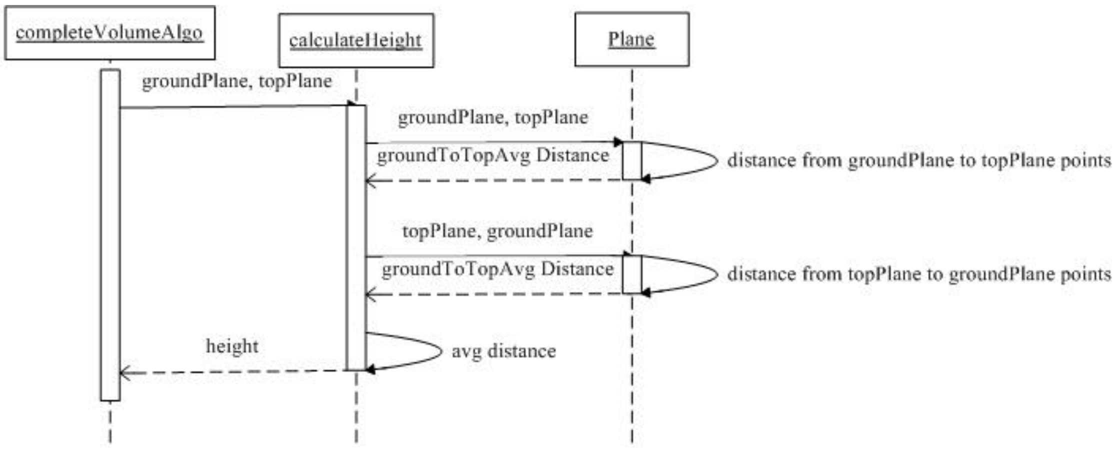


Figure 6.6 Sequence Diagram for Height Calculation

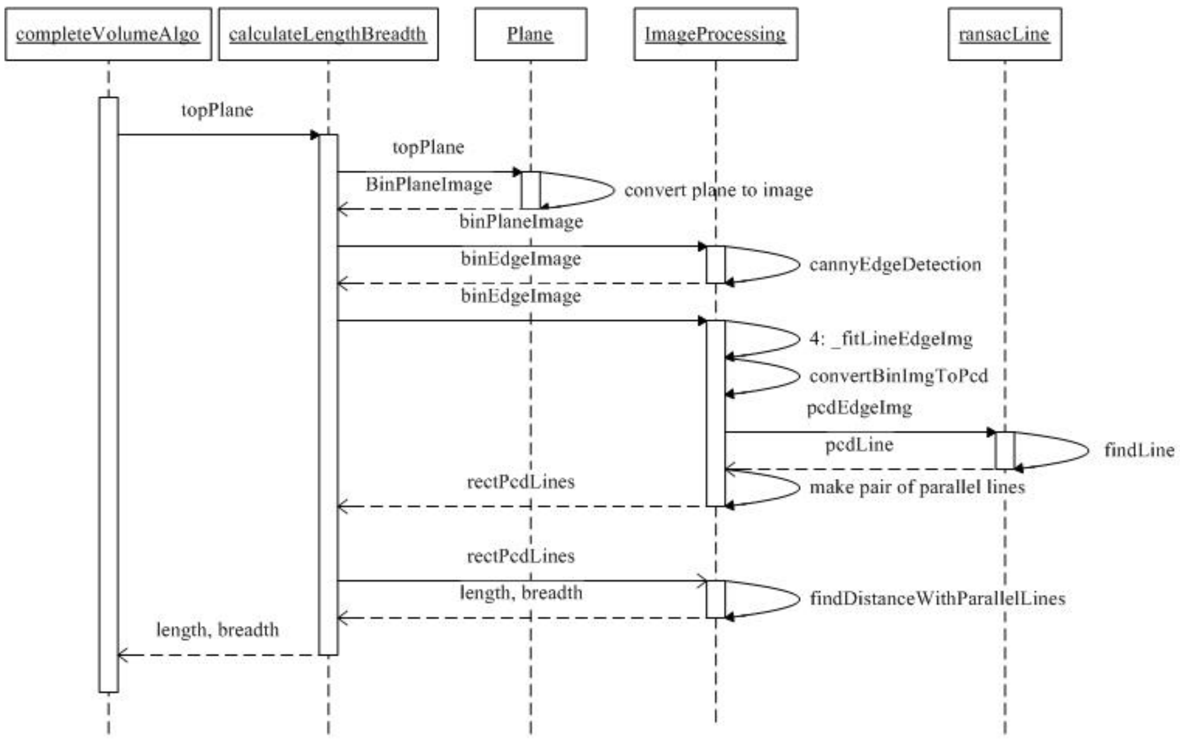


Figure 6.7 Sequence Diagram for Length Breadth Calculation

6.3 Activity Diagram

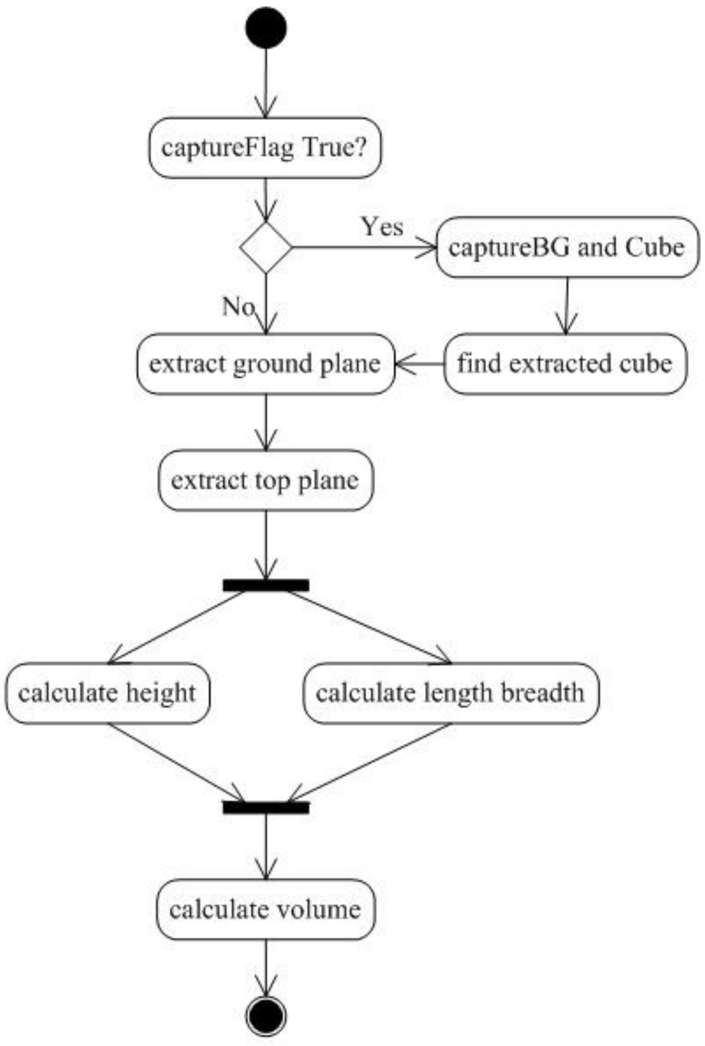


Figure 6.8 Activity diagram for find Volume of Cuboid using ToF camera

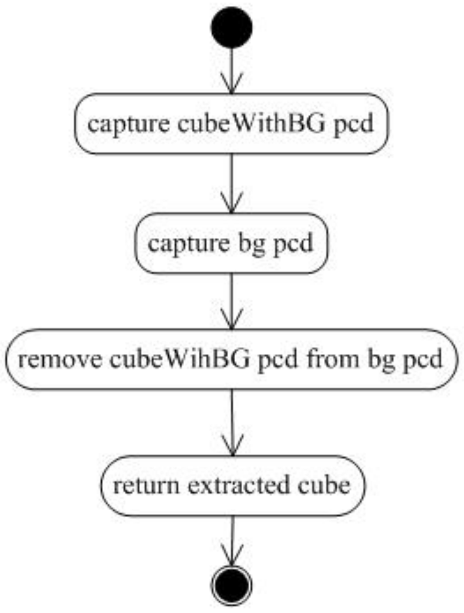


Figure 6.9 Activity Diagram for Camera Capture and cube extraction

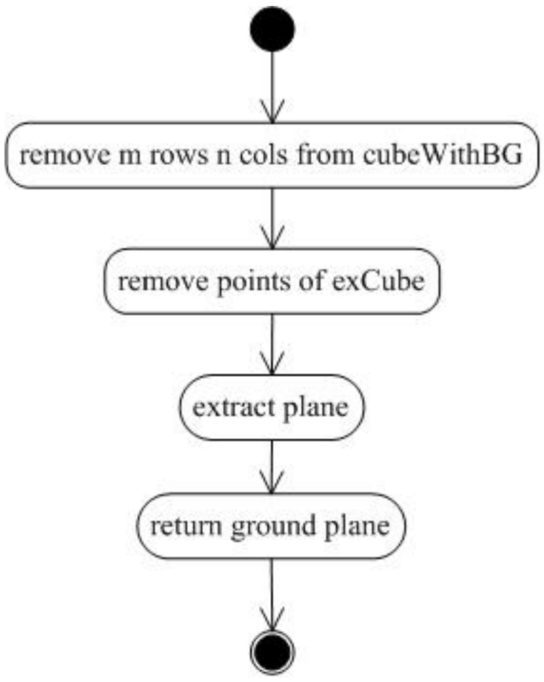


Figure 6.10 Activity Diagram for Ground Plane segmentation

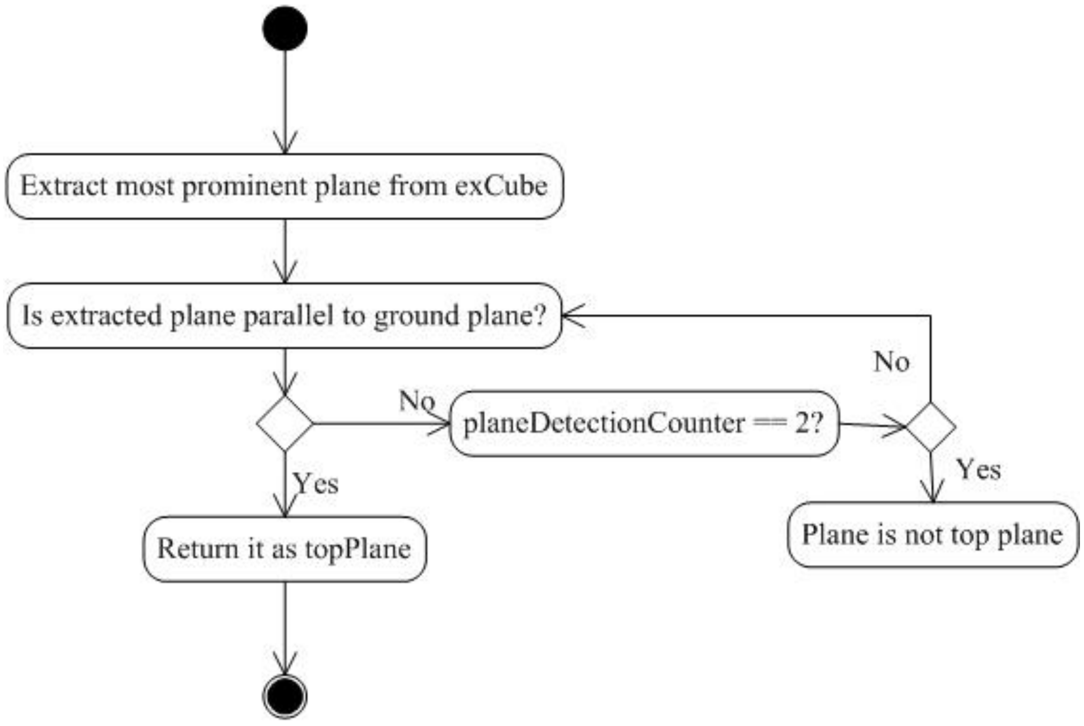

Extract most prominent plane from exCube
Is extracted plane parallel to ground plane?
No
planeDetectionCounter == 2?
No
Yes
Yes
Return it as topPlane
Plane is not top plane

Figure 6.11 Activity Diagram for Top Plane segmentation

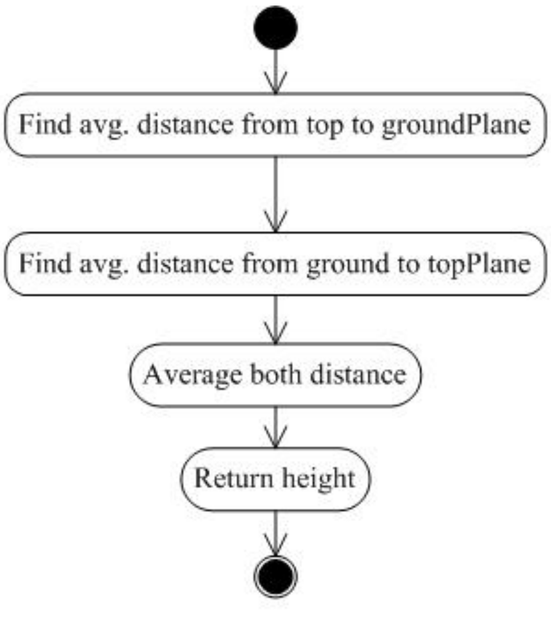

Find avg. distance from top to groundPlane
Find avg. distance from ground to topPlane
Average both distance
Return height

Figure 6.12 Activity Diagram for Height Calculation

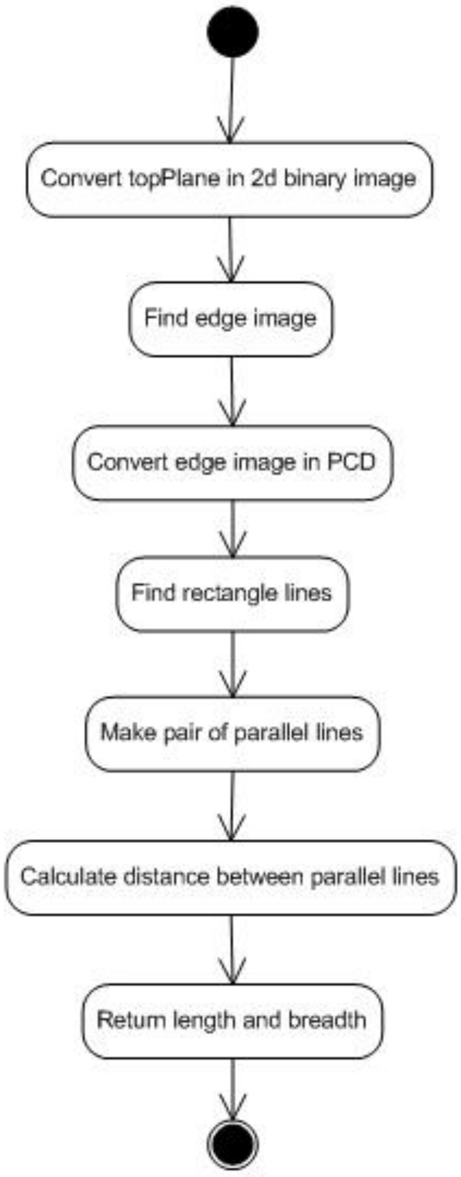

Convert topPlane in 2d binary image
Find edge image
Convert edge image in PCD
Find rectangle lines
Make pair of parallel lines
Calculate distance between parallel lines
Return length and breadth

Figure 6.13 Activity Diagram for Length Breadth Calculation

6.4 Complete Volume Algorithm

The following is the final pseudo code of the project, the actual code was written in python.

```
1.  Take Cube with background capture
2.  Remove Cube and take only background capture
3.  Extract cube using background subtraction
4.  Extract ground plane by applying RANSAC on background capture
5.  Extract three planes of cube using RANSAC iteratively
6.  Find the plane parallel to the ground plane (the top plane)
7.  Find distance between top and ground plane (this distance is height)
8.  Align top plane to the X-Y plane
9.  Project the aligned plane on the X-Y Plane
10. Convert the projected plane into image
11. Apply open-cv canny edge detector on the image
12. Apply line fitting on the edge image
13. Split the four edges in two groups each group containing parallel edges
14. Find distance between the parallel edges (one distance is length other breadth)
15. Find Volume $V = l \times b \times h$
```

CHAPTER-7

TESTING

7.1 Testing

When we create an application, the testing plays a very crucial part. It helps to find the bugs from the code while writing the codes and at the end, it provides the information about efficiency of the system. Which ensures the quality and reliability of the system that has been developed.

In our system, Find Volume of cuboid using ToF camera, we are using unittest module provided by Python.

7.1.1 Algorithm testing

After creating a new algorithm, we are first performed a trial run of that algorithm on the ideal dataset, i.e. in 2D approach, for edge detection, first we applied edge detection algorithm on the ideal cube. To check the different orientations, we rotated the cube on different angles. Here, with some rotations of the ideal cube the algorithm did not work. So, it was clear that for real world cube where we have different types of cube with background, the algorithm will not work as expected.

Similarly, for cube extraction, plane segmentation and dimension calculation, to validate the algorithm, first we performed a trial run on the ideal PCD cube. Once the algorithms were able to work on the ideal cube, we were applying those algorithm in the real-world cube.

7.1.2 Unit Testing

To check the sanity of the code, we have to write test bench for each function from each class. After writing test bench for a class, first we are checking that whether the methods inside class passes the tests or not. we are checking its coverage report, which informs us that how much part of the class has been covered, and how many lines are remaining to test. The important note here is that 100% coverage is necessary condition but not the sufficient one.

To test the method, first we used the ideal test cases, once they passed the test, we checked the boundary conditions for that method. And to make sure that we don’t miss any case, we test the method with semi – random inputs and compared them with expected outputs.

7.1.3 Acceptance Testing

Once an algorithm is verified with ideal cube, and its test bench has been passed, we applied that algorithm on the captured cube. To check the different cubes with different parameters we used a config.txt file, from which we were able to read the static value of the algorithm.

The benefits of this config file:

- No hard-coded values, as they are stored in config files in form of dictionary
- To check the cube with different parameters, no need to change the code
- Every cube capture had its own config file, so while testing a cube with some different parameters, run on other cube captures were not getting affected.

7.2 Observations and Results

Once the algorithm worked properly on one cube, we applied the algorithm on all the other cubes. We had more than 30 cube captures of 9 different cubes, from which 30 captures were acceptable captures. So, based on those 30 observations, we determined the accuracy of that algorithm.

7.2.1 Height Results

The height results were calculated for 9 distinct cubes in different positions. Some the results are shown below.

Cube	Actual Height (cm)	Calculated Height (cm)	Absolute Height Error (cm)
1	17.6	18.6	-1
2	14.8	15.2	-0.4
3	26.2	26.7	-0.5
4	10.7	11.5	-0.8
5	18.2	19.1	-0.9
6	21.1	21.7	-0.6
7	36.6	37	-0.4
8	21	21.2	-0.2
9	6.4	6.9	-0.5

Table 7.1 Height Results

(Note: Actual Height was measured via a standard scale)

The height has maximum error ±1.0 cm.

7.2.2 Length Breadth Results

Length and Breadth Results were calculated using the **IEDA** algorithm. The results for same cubes shown in height results are given in table 7.2.

Cube	Actual (cm)		Calculated (cm)		Absolute Error (cm)	
	Length	Breadth	Length	Breadth	Length	Breadth
1	14.8	34.8	15.7	33.8	-0.9	1
2	19.6	22.8	19.4	22.3	0.2	0.5
3	26.2	46.5	26.5	45.8	-0.3	0.7
4	10.7	52.7	10.9	53	-0.2	-0.3
5	21.1	28.3	21.1	27.9	0	0.4
6	15	50.6	15.5	51.5	-0.5	-0.9
7	32	40	31.2	39.6	0.8	0.4
8	13.6	52	14.2	52.4	-0.6	-0.4
9	26.4	26.8	26.8	26.4	-0.4	0.4

Table 7.2 Length, Breadth Results

(Note: Actual Length and Breadth were measured via a standard scale)

The Length and Breadth Results have a maximum error of ±1.0cm. Length Breadth Algorithm also fails only if there is some problem in the capture due the ToF technology or if the cube is uneven in its dimensions across different directions. We are currently uncertain what is causing the incorrectness of sub-1cm error. It could either be the resolution limit of camera, or incorrect point detection by camera. It is something we will consider Future Work.

7.2.3 Volume Results

Volume results were tabulated from the obtained length, breadth and height.

$$Volume\ of\ cuboid = Length * Breadth * Height$$

Cube	Actual Volume (cm)	Calculated Volume (cm)	Absolute Volume Error (cm)	% Volume Error
1	9064.70	9870.28	-805.57	-8.89
2	6613.82	6575.82	38.00	0.57
3	31919.46	32405.79	-486.33	-1.52
4	6033.62	6643.55	-609.93	-10.11
5	10867.77	11243.98	-376.21	-3.46
6	16014.90	17322.03	-1307.13	-8.16
7	46848.00	45714.24	1133.76	2.42
8	14851.20	15774.50	-923.30	-6.22
9	4528.13	4881.89	-353.76	-7.81

Table 7.3 Volume Results

Volume Results were found to be ±10% of the actual volume.

Our objective was to find the volume in 1% error. However, we discovered that dimension errors were all in absolute error, rather than in percentage errors. This means that for a 10cm dimension, we would get a 1cm error, but for a 100cm dimension, we would still get 1cm error. It was 10% error in the former, while only 1% in the latter.

CHAPTER-8

PROBLEM IN TOF TECHNOLOGY

Like almost every sensing device, the ToF camera as well is affected by several error sources that influence the accuracy of measured distance information. Basically, errors can be grouped into general TOF- as well as implementation-specific impacts.

8.1 Signal Quality

Due to the underlying time-of-flight principle, accurate distance information highly depends on a correct detection of the emitted signal. For this reason, ToF cameras are generally quite sensitive to external influences affecting the emitted signal either in shape or intensity. Especially the IR-reflectivity of observed surfaces has strong impact on the estimation result and often leads to under- or overexposed pixels. While underexposed pixels suffer from bad signal-to-noise ratios, i.e. noise, overexposed pixels are not capable to provide any distance information at all. Both are generally detectable via the amplitude information. Over-exposed pixel, however, sometimes provide a misleading good amplitude value.

8.2 Superposition

The superposition property of linear systems states that the response of a linear system to a sum of signals is the sum of the responses to each individual input signal. [10]
Superposition Signal-related errors may also occur due to interferences with other existing NIR light sources as well as multiple reflections within the scene, e.g. within corners. In both cases, superimposed signals affect the phase demodulation, yielding falsified distance information. A special case of superposition is basically related to the solid angle of a PCD pixel. Analog to multi-reflections, distance jumps inside a solid angle result in superimposed signals leading to a false phase estimate, often referred to as flying pixel. Usually, the distance information of flying pixels lie in-between fore- and background, but can also tend towards the camera depending on the surface’s true distance. A simple segmentation of flying pixels using a pixel’s amplitude is not

possible, for which reason more sophisticated processing techniques are necessary, that for example consider a pixel’s neighborhood relation.

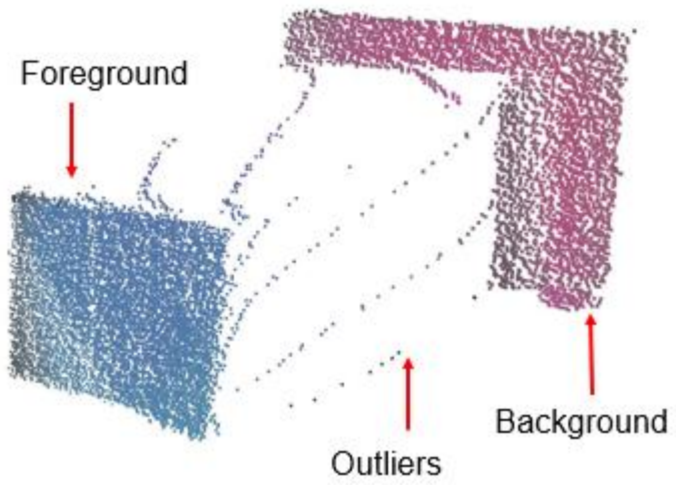


Figure 8.1 Flying pixel Effect Along object contours due to a superimposed signal

8.3 Noise

A general problem in the context of range sensing, especially for underexposed pixel, is given by noise affecting the accuracy of the measured distance information. With respect to the underlying CMOS design, ToF-related noise can be generally classified into three categories: time variant and time invariant noise as well as signal noise.

8.3.1 Time variant Noise

It covers thermal noise and reset noise. All these error sources are signal independent and- increase with rising temperature. Time variant noise can be significantly reduced or eliminated by proper cooling and signal processing techniques.

8.3.2 Time invariant Noise

Time Invariant Noise includes constant fixed pattern noise and leaker pixel noise.
Constant fixed pattern noise: defect pixels noticeable as static white or black pixel
Leaker pixels: pixels which are significant brighter than the neighbourhood

8.3.3 Signal Noise

Signal Noise, emerging from photon shot noise, is the most dominant noise and has a significant impact onto the effective signal-to-noise ratio. It cannot be suppressed and (more significantly) increases with the number of incoming photons.

8.4 System Error

Beside their general dependencies to the signal quality, current ToF cameras are also affected by system specific error sources that additionally influence the distance accuracy in a negative way.

8.5 Quantization Effects

Quantization effects occurs during the acquisition of subsequent phase images. Here, if phase image is wrong, then it will affect depth image followed by PCD image. Theoretical investigations and experiments show, that the corresponding quantization has a significant impact the captures which has more depth, i.e. for large distances, points are distributed in sparse manner as shown in figure 8.2.

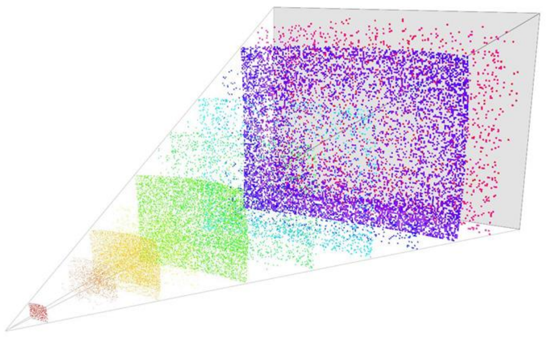
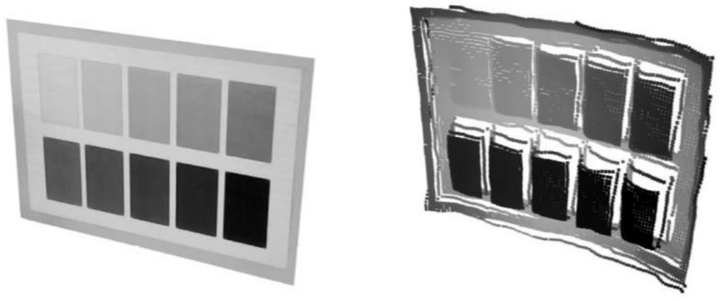


Fig. 8.2 Quantization Effects with varying distance

8.6 Intensity related Error

In ToF camera, distance information can also be altered by an intensity-related error inducing non-linear distance shifts with different reflective surfaces. The reason for the intensity-related false measurement is still unknown, but considered to be related to the semiconductor and the camera electronics. [14]



(a) Original Panel (b) distance shift

Fig. 8.3 Intensity-Related Distance Deviations due to Varying Object Reflectivity

8.7 Position Mismatch

During ground plane segmentation, we encountered a camera error of position mismatch between 2 consecutive captures. As we know we are capturing 2 captures, cube with PCD and background PCD. First, we encountered this problem at the time of cube extraction. Here, we tried to subtract two points from same location, but we didn't succeed. So, we subtracted them with some range threshold.

But when we were trying to measure height using the ground plane obtained by background PCD, we had 1-3 cm absolute error. So, to understand the problem, we tried to obtain ground from background with Cube PCD. Here, position of camera is not changed. Thus, background is same, and ideally position of background should not be changed.

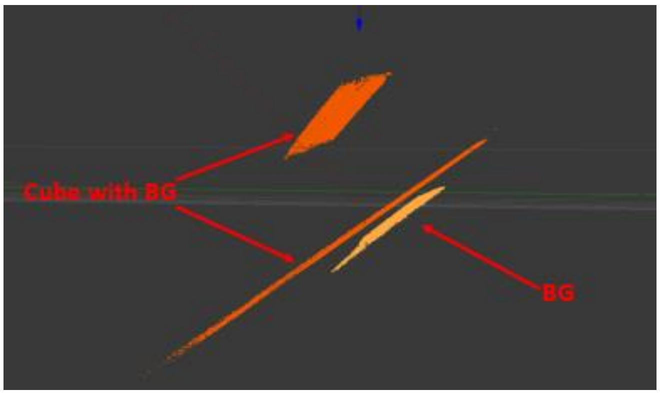


Figure 8.4 Position mismatch between two consecutive captures

But our observation was different from ideal condition shown in figure 8.4. Here, ground plane obtained by two consecutive captures, cube with background PCD and background PCD, there is some distance varying from -1 cm to 3 cm. Due to this reason, we changed our way to obtain the ground plane, so currently we are not facing this problem in height calculation. This problem can be solved camera calibration.

Thus, with many advantages of ToF technology, there are some disadvantages too. The experts in 3D vision technology are working together to solve these problems.

CHAPTER-9

CONCLUSION

9.1 Conclusion

We started off our journey of this project with the objective to find volume within ±1% error. However, as we jumped from algorithm to algorithm, few things were very clear: real data is very noisy and unpredictable compared with ideal data. For instance, we might expect a corner point to be at corner, but we might get a cluster of points in the same location! Thus, for any realistic measurements, we absolutely had to consider contribution of all the points present in the captured Point Cloud Data. Secondly, we discovered that dimension errors were not relative to size of the object. We found them to be absolute (±1.0 cm), regardless of the size of the object. Above all, we were able to prove that near accurate volume/dimension measurement is possible using ToF camera, which was the primary goal of this project.

9.2 Future work: ToF camera Calibration

For better dimensions, we need more accurate inputs. To improve accuracy of camera, we can do camera calibration.

Camera Calibration is the process of determining the internal camera parameters, and lens-distortion. Once the camera has been calibrated, the 2D depth-image can be transformed into a point cloud, which encodes the distance to the scene along each optical ray. It is convenient to use established calibration methods, which are based on images of a chequerboard pattern. The low-resolution of the amplitude image, however, makes it difficult to detect the board reliably. Having correct camera calibration is important as we are working majorly on point cloud, point cloud as described earlier is derived from depth map, if any error is present in the depth map it will be reflected in the point cloud.

Besides, it is important to solve the reflectivity problem. As due to the application of the project in the industry where the colour and packaging of cube cannot be restricted. The program should function for any given colour, packaging of cube.

APPENDIX A

DEFINATIONS, ABBRIVIATIONS AND ACRONYMS

Name	Description
BG	Background
Computer Vision [11]	**Computer vision** is the science that aims to give a similar, if not better, capability to a machine or computer. Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images.
Depth Map [6]	In 3D computer graphics, a **depth map** is an image or image channel that contains information relating to the distance of the surfaces of scene objects from a viewpoint.
IR	Infra-Red
NIR	Non-Infra-Red
PCD [8]	**Point Cloud Data** A **point cloud** is a set of data points in some coordinate system. In a three-dimensional coordinate system, these points are usually defined by *X*, *Y*, and *Z* coordinates, and often are intended to represent the external surface of an object. Here we have *X, Y, Z* co-ordinate values and *I* as intensity.
PCL	Point Cloud Library
Range Imaging [12]	**Range imaging** is the name for a collection of techniques that are used to produce a 2D image showing the distance to points in a scene from a specific point, normally associated with some type of sensor device.
Time of Flight [1]	**Time of flight** (**TOF**) describes a variety of methods that measure the time that it takes for an object, particle or acoustic, electromagnetic or other wave to travel a distance through a medium.
TI	Texas Instruments
ToF camera [2]	**Time of Flight camera** A **Time of Flight camera** (ToF camera) is a range imaging camera system that resolves distance based on the known speed of light, measuring the time-of-flight of a light signal between the camera and the subject for each point of image.

APPENDIX B

RANSAC PLANE SEGMENTATION

RANSAC plane segmentation takes input of an ideal data model from the user (in our case, a plane) and tries to fit that model in given set of data. Since, our ToF camera returns us a cuboid with three faces, we can call RANSAC on our data with plane as desired model. For a plane, it randomly selects three points in given data set and generates a plane equation of form:$ax + by + cz = d$. Then it counts how many points in the given data set match this equation in a tolerance limit. If the distances of the points from the ideal plane are within the tolerance range, it is counted as part of the plane. If the count of currently selected model is larger than the previous one, the model is saved. Running till exhaustion, the algorithm iterates over the process ${}^{n}_{3}C$ times.

For a typical PCD data set of $n = 76800$, running till exhaustion is very computationally heavy. However, since the algorithm selects points randomly from a data set, it is very likely that within a 1000 or so trials, if the plane is prominent enough, the selected plane at $iteration = 1000$ would be the correct one.

APPENDIX C

MATHEMATICAL EQUATIONS USED FOR PROJECT

- Volume $V = l \times b \times h$
- Line Equation in 2D: $y = mx + c$
- Line Equation in 3D:

  $$\frac{x-x1}{l1} = \frac{y-y1}{l2} = \frac{z-z1}{l3} = (k), where\ l1 \neq 0\ l2 \neq 0, l3 \neq 0$$

  Where, $[l1, l2, l3]$ are line vector, and $[x1, y1, z1]$ are point on the line.
- Equation of line passing through 2 distinct points:

  $$\frac{x - x1}{x2 - x1} = \frac{y - y1}{y2 - y1} = \frac{z - z1}{z2 - z1}$$
- Plane Equation: $ax + by + cz + d = 0$
- Plane normal vector: $[a, b, c]$
- Shift point: $point'(x', y', z') = point(x, y, z) + newOrigin\ (a, b, c)$
- Euclidean distance between 2 points

  $$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2}$$
- Perpendicular distance between line and point outside the line:

  $$distance = \frac{|(\bar{p} - \bar{a}) \times \bar{l}|}{|\bar{l}|}$$

  Where $\bar{p}$: point on line,

  $\bar{a}$: point from which we are finding distance

  $\bar{l}$: Line vector
- Perpendicular distance between plane and point outside the plane:

  $$distance = \frac{|(\bar{p} \cdot \bar{n}) - d|}{|\bar{n}|}$$

  Where $\bar{p}$: point on line,

  $\bar{n}$: normal of plane

- Angle between 2 planes:

$$\theta = \cos^{-1}\frac{|\overline{n1}\cdot\overline{n2}|}{|\overline{n1}||\overline{n2}|}$$

Where $\overline{n1}$ and $\bar{n}2$ are normal of planes

- Rotate point around axis:
    - To rotate point $A(x, y, z)$ around axis $\bar{u}\ (u1, u2, u3)$, find rotation matrix,

$$\begin{pmatrix} cos\theta + u1^2(1-\cos\theta) & u1u2(1-\cos\theta) & +u2\sin\theta \\ u1u2(1-cos\theta) & cos\theta + u2^2(1-cos\theta) & -u1\sin\theta \\ -u2\sin\theta & u1\sin\theta & \cos\theta \end{pmatrix}$$

    - Multiply this rotation matrix with point matrix

$$A'(x', y', z') = Rotation\ matrix * A\begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- Align plane to XY plane:
    - Find angle θ between given plane having normal $\bar{n}\ (a, b, c)$ with some point $A(x, y, z)$ and XY plane with normal $\hat{k}\ = 0$
    - Find axis of rotation $\bar{u}$ using,

$$\bar{u}(u1, u2, u3) = \frac{\bar{n}\times\hat{k}}{|\bar{n}|} = \frac{1}{\sqrt{a^2+b^2+c^2}}(b, -a, 0)^T$$

    - Once we have found axis of rotation, rotate point around axis shown in above formula.

- Project point on Plane:

Suppose we have a line $\overrightarrow{AB}$, and we want to project point P on line $\overrightarrow{AB}$.

Projected point

$$P' = A + \frac{(\overrightarrow{AP}\cdot\overrightarrow{AB})}{(\overrightarrow{AB}\cdot\overrightarrow{AB})} * \overrightarrow{AB}$$

Bibliography

[1]"Time of Flight" Internet: https://en.wikipedia.org/wiki/Time_of_flight, Oct 4, 2016 [March 28, 2017]

[2]"Time of Flight Camera" Internet: https://en.wikipedia.org/wiki/Time-of-flight_camera March 12, 2017 [March 31, 2017]

[3]Li Larry (2014, May). "Time-of-Flight camera - An Introduction" *Technical white paper, Texas Instruments* [on-line] Available at http://www.ti.com/lit/wp/sloa190b/sloa190b.pdf

[4]"OPT8241 3D Time-of-Flight Sensor" *Texas Instruments* [on-line] Available at http://www.ti.com/lit/ds/sbas704b/sbas704b.pdf

[5] "OPT8241 CDK Evolution Module Quick Start Guide" *Texas Instruments* [on-line] Available at http://www.ti.com/lit/ug/sbou156b/sbou156b.pdf

[6]"Depth Map" Internet: https://en.wikipedia.org/wiki/Depth_map February 17, 2017 [March 31,2017]

[7] "Filtering for 3D Time-of-Flight Camera" [2016, Jan] *White Paper, Texas Instruments*[on-line] Available At http://www.ti.com/lit/wp/sloa230/sloa230.pdf

[8] "Point Cloud" Internet: https://en.wikipedia.org/wiki/Point_cloud February 22, 2017 [March 31, 2017]

[9] "The PCD (Point Cloud Data) file format" Internet: http://pointclouds.org/documentation/tutorials/pcd_file_format.php [April 5, 2017]

[10] Julius O. Smith III (2007, September) *Superposition" Introduction to digital filters with audio Applications* [on-line] Available at https://ccrma.stanford.edu/~jos/fp/Superposition.html

[11] "What is computer vision" *The British Machine Vision Association and Society for Pattern Recognition* Internet: http://www.bmva.org/visionoverview

[12] "Range Imaging" Internet: https://en.wikipedia.org/wiki/Range_imaging November 20, 2016 [March 31, 2017]

[13] Miles Hansard, Seungkyu Lee, Ouk Choi, Radu Horaud. "Time of Flight Cameras: Principles, Methods, and Applications". Springer, pp.95, 2012, Springer Briefs in Computer Science, ISBN 978-1-4471-4658-2. <10.1007/978-1-4471-4658-2>. <Hal-00725654> [on-line] Available at https://hal.inria.fr/hal-00725654/PDF/TOF.pdf

[14] Marvin Lindner, Ingo Schiller, Andreas Kolb, Reinhard Koch, "Time-of-Flight sensor calibration for accurate range sensing", in: Computer Vision and Image Understanding 114 (2010) 1318–1328. [on-line] Available at http://www.zess.unisiegen.de/cms/papers/Lindner2010TimeofFlightSensorCalibration.pdf

[15] Swaroop C H, *A byte of Python* [On-line] Available at http://files.swaroopch.com/python/byte_of_python.pdf

[16] Open Source Computer Vision Library, author: Itseez, version 2.4.9.0.