

Application of Intel RealSense based depth perception for pothole detection in MAVI system (Mobility Assistant for Visually Impaired)

Thesis submitted by

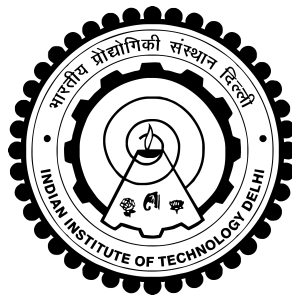
Sachin Yadav
2011CS50293

under the guidance of

Prof. M. Balakrishnan
Prof. Chetan Arora (IIIT Delhi)

*in partial fulfilment of the requirements
for the award of the degree of*

Bachelor & Master of Technology

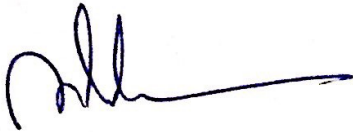


Department Of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY DELHI

May 2017

THESIS CERTIFICATE

This is to certify that the thesis titled **Application of Intel RealSense Based Depth Perception for Pothole Detection in MAVI System**, submitted by **Sachin Yadav**, Entry number **2011CS50293**, to the Indian Institute of Technology, Delhi, for the award of the degree of **Bachelor & Master of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.



Prof. M. Balakrishnan

Dept. of Computer Science & Engineering
Indian Institute of Technology, Delhi

Prof. Chetan Arora (ON LEAVE)
Dept. of Computer Science & Engineering
Indraprastha Institute of Information Technology, Delhi

THESIS CERTIFICATE

This is to certify that the thesis titled **Application of Intel RealSense based depth perception for pothole detection in MAVI system** , submitted by **Sachin Yadav**, Entry number **2011CS50293**, to the Indian Institute of Technology, Delhi, for the award of the degree of **Bachelor & Master of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. M. Balakrishnan

Dept. of Computer Science & Engineering
Indian Institute of Technology, Delhi

Prof. Chetan Arora

Dept. of Computer Science & Engineering
Indraprastha Institute of Information Technology, Delhi

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my deepest and sincere gratitude to my thesis supervisors **Prof. M. Balakrishnan** and **Prof. Chetan Arora** for their valuable suggestions, discussions and continuous support throughout the course of the project.

I would also like to thank **Mr. Rajesh Kedia** and **Mr. Anupam Sobti** for their useful ideas and insights.

Finally, I would like to congratulate the entire MAVI team for their continuing success and thank them for making my journey full of experiences and knowledge.

Sachin Yadav

2011CS50293

ABSTRACT

MAVI (Mobility Assistant for Visually Impaired)[20] is an ambitious project aimed at enabling mobility for visually impaired people, targeted at the Indian scenario, where due to widespread non-standard practices, a wide range of unique complexities arise. Particularly, the project aims to cover the aspects of 'security', 'social inclusion' & 'navigation' in a user friendly and affordable manner. It uses a GPS/IMU sensor along with a RGB camera as input channels to the processing unit on the Zedboard , which communicates with the user through non-visual feedback (beep, speech and vibrate).

Pedestrian safety is of paramount importance. In the Indian scenario, two major obstacles are posed by stray animals (cattle and dogs) and potholes on the roads. Therefore it is crucial to detect stray animals and potholes. The focus of this project is on pothole detection. Currently, the processing is done on the incoming RGB stream for classifying textures and identifying potholes. This thesis is part of a project which seeks to explore the possibility of adding Intel RealSense R200 depth and IR camera to the above system and process the depth image stream generated by the depth camera to identify variations in the depth field in order to locate, classify and measure potholes. This can either be used standalone or to compliment and enhance the existing RGB mechanism.

This thesis is focused on assessing the various ways to capture raw depth data from Intel RealSense R200 , pre-processing and processing of depth and corresponding color maps to segment potholes. This is first accomplished for the case where the depth maps are shot parallel to the plane of interest. The segmented potholes are extracted and their dimensions measured using the measurement API available publicly as part of the Intel Realsense SDK for Windows. The same is then attempted for the case where the camera is inclined to the plane of interest , as is the case when it is mounted on the chest. Various heuristics have been tried on the general case. The results and inferences have been thoroughly reported. The limitations of the approach as well as of the device , along with possible directions for further research have been discussed at the end.

Contents

ACKNOWLEDGEMENTS	i
ABSTRACT	ii
LIST OF TABLES	v
LIST OF FIGURES	1
1 INTRODUCTION	2
1.1 Mobility Assistant for Visually Impaired(MAVI)	2
1.2 Intel RealSense R200	2
1.2.1 Specifications	3
1.2.2 Intel RealSense SDK for Windows	4
1.3 Motivation and Scope	5
1.4 Thesis outline	6
1.5 Previous Work	6
2 DEPTHMAP EXTRACTION	8
2.1 Windows	8
2.1.1 Raw IR/Depth streams	8
2.1.2 eXtensible Device Metadata (XDM)	9
2.1.3 VLC streaming	11
2.2 Linux (Ubuntu 14.04)	12
2.2.1 Librealsense	12
2.2.2 OpenCV	13
2.3 3D Point Clouds	13
2.3.1 From XDM to Point Cloud	13
2.3.2 Real-Time Point Cloud Capture	15
2.4 Dataset	16

2.5	Challenges	16
2.5.1	Platform dependence	16
2.5.2	Effect of Ambient/Device Conditions	17
3	DEPTHMAP PROCESSING	19
3.1	Top Down view	19
3.1.1	XDM to DepthMap	19
3.1.2	Histogram Equalization	20
3.1.3	Canny Edge Detection	22
3.1.4	Hough Transform	23
3.1.5	MeasureDistance	24
3.2	Inclined View	26
3.2.1	Banding Effect	26
3.2.2	Point Cloud approach	28
3.2.3	Point Cloud outlier removal	29
3.2.4	Normals Estimation	30
3.2.5	RANSAC Plane Segmentation	31
3.2.6	Region Growing Segmentation based Cluster Extraction	33
3.2.7	Challenges	34
3.3	Summary Of Results	35
3.3.1	Results for Hough Transform	35
3.3.2	Results for PCD processing	36
4	CONCLUSION AND FUTURE WORK	37
4.1	Conclusion	37
4.2	Future Work	37

List of Tables

3.1	HoughCircle depthmap processing	35
3.2	HoughRectangle depthmap processing	35
3.3	Ransac results : 'FAIR' quality pcd	36
3.4	Ransac results : 'BAD' quality pcd	36

List of Figures

1.1	MAVI system	2
1.2	R200 components and streams	3
1.3	R200 color camera properties	4
1.4	R200 infrared camera properties	4
1.5	R200 typical power consumption	4
1.6	Setup used by Moazzam et. al.	7
2.1	Raw color and depth stream	9
2.2	Raw color and depth stream 2	9
2.3	XDM structure	11
2.4	Bandwidth Requirements at different configurations	12
2.5	XDM image	14
2.6	Corresponding Point Cloud	15
2.7	'GOOD' quality depth map, with discernible variation in depthmap	18
2.8	'FAIR' quality depth map, with less discernible variation	18
2.9	'BAD' quality depth map, with negligible variation in depthmap	18
3.1	XDM to depthmap	20
3.2	XDM to depthmap	21
3.3	Gaussian Blur smoothing	21
3.4	Canny Edge detection	22
3.5	Canny with increased threshold	22
3.6	Initial image and Final output	24
3.7	XDM to final result - Rectangle	25
3.8	XDM to final result - Circular	25
3.9	Banding effect w.r.t. gradient	26
3.10	Original XDM images	26

3.11	Downsized to 8 bit depthmap - banding effect	27
3.12	16 bit depthmap	27
3.13	Downsizing by OpenCV	27
3.14	Original Image XDM	28
3.15	Corresponding 8 bit, 16 bit and colormap	28
3.16	Original Image XDM	29
3.17	Outlier removal	30
3.18	Before plane segmentation (eg. 1)	32
3.19	Segmented Plane (eg. 1)	32
3.20	Before plane segmentation (eg. 2)	33
3.21	Segmented Plane (eg. 2)	33
3.22	Many outliers, 70 percent NaN inliers	35

Chapter 1

INTRODUCTION

1.1 Mobility Assistant for Visually Impaired(MAVI)

MAVI is an ambitious project aimed at enabling mobility for visually impaired individuals, particularly adapted to the Indian scenario. A major challenge in this regard is the wide range of complexities that arise in the Indian context due to rampant non-standard practices.[20]

'The objective of MAVI System is to conceptualize and design a smart-camera based vision system, capable of extracting useful information, e.g.faces, texture, signboard information from captured images. It consists of different modules integrated onto one platform, communicating to user through mobile interface and using cloud to translate the coordinates sent by Localization module into navigational information that can be annotated to signboard detected in a frame captured by the camera. The modules in MAVI consist of Face Detection and Recognition, Texture Recognition, Signboard Detection and Localization, with an aim to add more data channels and processing modules as per the requirements.'[11][20]

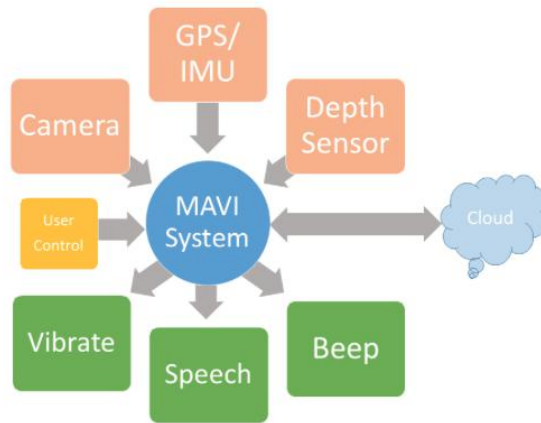


Figure 1.1: MAVI system

1.2 Intel RealSense R200

RealSense is an Intel platform built for implementations of human-computer interaction vision algorithms as a part of which 3 RGB-D cameras are released along with a supporting library of methods , runtimes, and APIs named IntelRealSense SDK.

An Intel RealSense camera contains the following four components: a conventional camera, an infrared laser projector, an infrared camera, and a microphone array. The infrared projector projects a grid onto the scene (in infrared light which is invisible to human eye) and the infrared camera records it to compute depth information. The microphone array allows localizing sound sources in space and performing background noise cancellation.[16][15]

Intel RealSense R200 is one of the three cameras announced with the fore said technology, and the only rear-mounted camera among the three. It is primarily aimed at applications such as Augmented Reality, and object scanning.’[16]

1.2.1 Specifications

The R200 camera is a USB 3.0 device that can provide color, depth, and infrared video streams. Depth video streams are like color video streams, except each pixel has a value representing the distance away from the camera instead of color information. It consists of an infrared laser projection system, two infrared and a full HD color imaging sensors. The depth video stream is generated with stereo vision technology assisted by the Infrared laser projector and the two infrared imaging sensors. Color data is provided by the full HD color imaging sensor. It is targeted at real world problems involving human computer interaction and depth + photography applications[7][23]



Figure 1.2: R200 components and streams

Parameter	R200 Color Camera
Active Pixels	1920x1080
Sensor Aspect Ratio	16:9
Filter Type	IR Cut Filter
Focus	Fixed
Shutter Type	Rolling Shutter
Vertical Field of View	43° +/-2°
Horizontal Field of View	70° +/-2°
Diagonal Field of View	77° +/-4°

Figure 1.3: R200 color camera properties

Parameter	Infrared Cameras
Active Pixels	640x480
Sensor Aspect Ratio	4:3
Filter Type	IR Band Pass
Focus	Fixed
Shutter Type	Global Shutter
Vertical Field of View	46° +/-5°
Horizontal Field of View	59° +/-5°

Figure 1.4: R200 infrared camera properties

Depth Mode	Infrared Mode	Color Mode	Power	Unit
OFF	OFF	OFF	0.334	W
OFF	OFF	1080P, 30FPS	0.75-.92	W
VGA, 60FPS	VGA, 60FPS	OFF	0.99-1.15	W
VGA, 60FPS	VGA, 60FPS	1080P, 30FPS	1.3-1.6	W

Figure 1.5: R200 typical power consumption

1.2.2 Intel RealSense SDK for Windows

The Intel RealSense SDK[8] is a computer vision library especially targeted towards pattern detection and recognition implementations. Its objective is to target next generation human computer communication.

Some framework APIs provided within the SDK are:

- Enhanced Photography
- Object Recognition
- User Segmentation
- 3D Scanning
- Full Hand Gesture Recognition

However, on the downside, this limits the platform choice to Windows, severely restricting the ability to use the API's provided within the SDK to be used in embedded systems mostly running linux. Also, the APIs stick within the prescribed object range, object-background separation range and optimal ambient conditions. Our use case requires the camera to be used to its range and processing limits, the challenges popped up by which are discussed further.

1.3 Motivation and Scope

This thesis focuses on the 'safety' objective of MAVI , especially in the Indian context where potholes, sewer holes and dugholes pose a major challenge for visually impaired pedestrians. Hence all the data has been acquired from Indian roads.

Intel RealSense R200 is used to capture the data, both as depth maps and as point clouds, which have the added advantage of 3D visualization of the scene and a platform independent PCL library boasting a wide range of 3D vision algorithms.

This project aims to develop a tool to detect, classify and quantify potholes by analyzing deviations in the depth field. This is motivated by the assumption that when combined with RGB based texture detection , it should significantly decrease the number of false positives arising out of shadows. Such shadow regions will not create distortions in the depth map.

The data and also the problem is divided into two classes :

- Top Down shots : These images are captured parallel to the ground , hence the background of the pothole is assumed to be at same depth. This isolates the pothole well.
- Inclined shots : These images are captured using the model prototype , inclined at 22.5 degrees to the ground normal. This is more complex as it is subject to varying depths and ambient conditions start to be a factor as the horizon enters the field of view.

1.4 Thesis outline

This thesis is divided into 5 chapters. Chapter 1 introduces the project, the device and previous work in this regard.

Chapter 2 details the methods used to capture the depth data, the various formats the data can be captured in with respect to different operating systems and the specifications of the final dataset

Chapter 3 explains the techniques used to process the depth maps for the required objective, for both the 'top-down' and 'inclined' cases, the heuristics attempted for the 'inclined' case and the challenges faced.

Chapter 4 shows the results obtained for both the approaches for the entire dataset. It also discusses and infers the same results before concluding with a comparison of the depthmap and point cloud approaches.

Chapter 5 concludes the thesis and discusses scope for future work. An added section summarises most recent work in the domain to provide further insights and direction for future work.

1.5 Previous Work

Prior to this work, no methods for detecting potholes using Intel RealSense technology exist. This can be attributed to the fact that this technology is fairly recent.

Existing methods for pothole detection can be broadly divided into vibration based methods by Yu and Yu, De Zoysa et al., Erikson et al. and Mednis et al., 3D reconstruction-based methods by Wang, Chang et al., Hou et al., Li et al., Staniek, and Moazzam et al., and vision-based methods by Koch and Brilakis, Jog et al., Lokeshwor et al., Koch et al., Buza et al., and Lokeshwor et al [3]

Few papers deal with application of Kinect for the purpose of pothole detection. Joubert et al.[1] proposed a low-cost sensor system using Kinect sensor and a high-speed USB camera to detect and analyze potholes. Some tests were done to determine the viability of using Kinect to examine potholes.

Moazzam et al. [19][2] used a low-cost Kinect sensor to collect the pavement depth images and calculate the approximate volume of a pothole. Using a low-cost Kinect sensor, the pavement depth images were collected from concrete and asphalt roads. Meshes were generated for better visualization of potholes. Area of pothole was analyzed with respect to depth. The approximate volume of pothole was calculated using trapezoidal rule on area-depth curves through pavement image analysis.

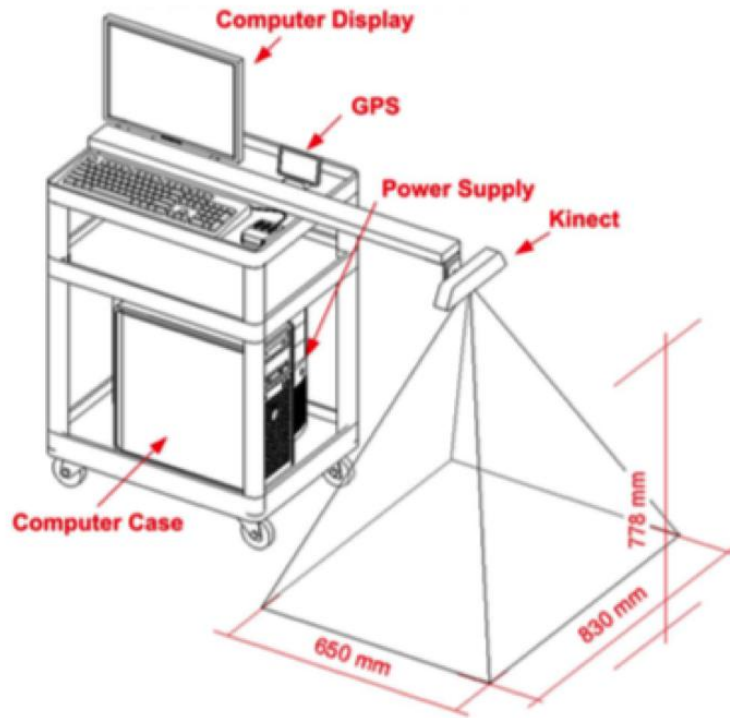


Figure 1.6: Setup used by Moazzam et. al.

Several papers are present using similar approaches , the common factor being that the the camera is assumed to be placed at a fixed position and angle with respect to the plane of interest , and the image is accordingly pre-processed. The expected depth of the plane therefore is already known and any variation indicates presence of obstacle/pothole, after suitably normalizing for error margin. However, for our objective the processing needs to be done without any fixed reference.

Chapter 2

DEPTHMAP EXTRACTION

2.1 Windows

Intel RealSense SDK for Windows[8][15] provides developers with depth data useful for extracting distance and size of objects from images and the ability to reconstruct a scene in 3D. It offers several abstractions for dealing with the image data, allowing easy access to and direct handling of the data buffers. However, these methods and APIs are exclusive to Windows OS, and not yet developed for Linux and MacOS systems.

2.1.1 Raw IR/Depth streams

To illustrate the flow, here's a quick walkthrough of the basic steps to capture the camera data as raw individual frames using the Intel RealSense SDK[4]:

1. Create and initialize an instance of PXCSenseManager

```
PXCSenseManager::CreateInstance()
```

2. Use sense manager to enable the depth and/or color stream specifying resolution and frame rate

```
senseManager->EnableStream(PXCCapture::STREAM_TYPE_DEPTH, 320, 240, 30);
```

3. Use sense manager to acquire / lock an available frame

```
camStatus = senseManager->AcquireFrame(true);
```

4. Use sense manager to query the available samples

```
PXCCapture::Sample *sample = senseManager->QuerySample();
```

5. Access the Sample's specific PXCIImage members to extract the desired data

```
colorImage = sample->color;  
depthImage = sample->depth;
```

6. Use member functions of the PXCIImage object, such as QueryInfo(), to get the parameters of the color or depth data.

7. The data buffer is organized in planes and pixels and specific pitches that correspond to the parameters in the ImageInfo mentioned above. To access the raw image buffers, you must use the `PXCImage::AcquireAccess()` function passing it your `PXCImage::ImageData` pointer to store the buffer location. The planes and pitches of the `ImageData` should be initialized to empty arrays.[4]
8. After obtaining the desired image data, you must release access to the data buffer.

```
PXCImage::ReleaseAccess();
PXCSenseManager::ReleaseFrame();
```



Figure 2.1: Raw color and depth stream



Figure 2.2: Raw color and depth stream 2

2.1.2 eXtensible Device Metadata (XDM)

The several abstractions provided by the SDK for handling and storing RGB-D data include the `PXCPhoto::SaveXDM()` function, which will simply write the depth enhanced JPEG file

format. Additionally Background Segmentation (BGS), Enhanced Photography and Video (EPV) and Scene Perception (SP) modules in the SDK have options to provide additional filtering of the depth data before saving it in XDM format, which is specified further.

'The eXtensible Device Metadata (XDM) specification is a standard for storing device related metadata in common image containers such as JPEG and PNG while maintaining compatibility with existing image viewers. The metadata that can be stored includes depthmap, pointcloud, device and camera pose, lens perspective model, image reliability data, and vendor related information about the device and sensors. The data storage format is based on the Adobe XMP standard. XDM is being developed as an open file format at xdm.org. The XDM specification includes support for multiple cameras, each with its own relative physical orientation. Each camera data structure can optionally contain an image and depth data if the device platform can provide them.'

[17]

Data Structure

- **Container Image** : The image external to the XDM, visible to normal non XDM apps
- **Device** : The root object of the RDF/XML document as in the AdobeXMP standard
 - **Revision** : Revision number of XDM specification
 - **VendorInfo** : Vendor related information for the device
 - **DevicePose** : Device pose with respect to the world
 - **Camera** : All the info for a given camera. There must be a camera for any image. The container image is associated with the first camera, which is considered the primary camera for the image.
 - * **VendorInfo** : Vendor related information for the camera
 - * **CameraPose** : Camera pose relative to the device
 - * **Image** : Image provided by the camera
 - * **ImagingModel** : Imaging(lens) model
 - * **DepthMap** : Depth related information including the depthmap and noise model
 - * **PointCloud** : Point cloud data

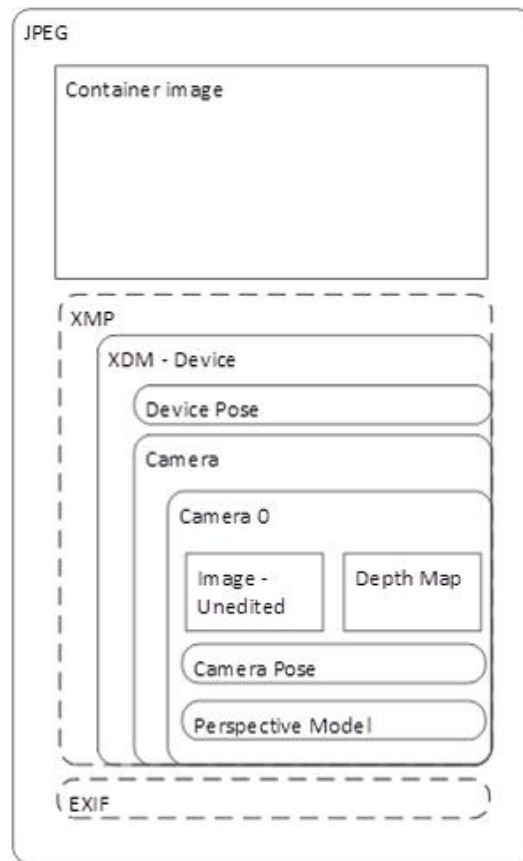


Figure 2.3: XDM structure

XDM depth parser (Java)

It parses the input image in the XDM format and outputs XML files including all XDM fields, color image(s), and depth map image(s). Usage example :

```
java -getcolorimage XDMfile
java -getdepthmap XDMfile
```

2.1.3 VLC streaming

The streams can also be played and saved to disk using VLC player. However , at the time of writing this , it only provided supports for RGB stream and even that requires huge disk write speeds in order to tolerate the high data bandwidth, as shown in the following table. Intel RealSense documentation suggests directly saving the stream only onto an SSD.

FPS	COLOR RESOLUTION		DEPTH/IR RESOLUTION		BANDWIDTH MB/SEC
	WIDTH	HEIGHT	WIDTH	HEIGHT	
60	640	480	320	240	61
60	320	240	480	360	32
60	640	480	480	360	72
30	640	480	320	240	30
30	1280	720	320	240	83
30	1920	1080	320	240	182
30	320	240	480	360	16
30	640	480	480	360	36
30	1280	720	480	360	88
30	1920	1080	480	360	187

Figure 2.4: Bandwidth Requirements at different configurations

2.2 Linux (Ubuntu 14.04)

Intel provides an experimental cross-platform library API named librealsense for working with RealSense cameras on Linux, Windows as well as Mac. On Linux we can use either librealsense or OpenCV to capture depth frames. The important features of both the methods are highlighted below:

2.2.1 Librealsense

'Librealsense is a cross-platform library (Linux, OSX, Windows) for capturing data from the Intel® RealSense™ F200, SR300 and R200 cameras. Many features of RealSense™ devices are implemented in librealsense, including multi-camera capture.'[18]

Librealsense requires two external dependencies, GLFW3 (all platforms) and libusb-1.0 (Mac/Linux). These and certain other dependencies as listed on librealsense github page. It connects to RealSense through UVC and USB protocol without having to link with SDK runtimes. Some backends required are listed below :

- A video4linux2 backend to provide kernel space control.
- A libuvc backend build using libuvc to provide user space authentication
- A Windows Media Foundation backend for Windows 8.1 and newer

Functionality

- Native streams: depth, color, infrared
- Synthetic streams: rectified images, depth aligned to color and vice versa, etc.
- Intrinsic/extrinsic calibration information
- Hardware-specific functionality for individual camera generations
- Multi-camera capture across varying archs. (multiple realsense cameras in the same application)

2.2.2 OpenCV

To process the depth frames using OpenCV, first they were grabbed using the SDK, and then converted to OpenCV format. If however, the stream capture feature of OpenCV is directly used, it only lists the RGB stream. Further, the entire stream is not saved to disk. Only after a specified interval, a frame is captured from the stream and saved. At the time of writing this, only RGB frames were captured using this method.

2.3 3D Point Clouds

A point clouds consists of a set of 3 dimensional data points defined by xyz coordinates to represent and visualize the surface of an object from an absolute perspective unlike a depth map which represents those points of the point cloud which are seen from a particular viewpoint.

Depth Maps and Point Clouds are distinct ways to visualize the same information. However each point cloud consists of an infinite number of depth maps, one each for each viewpoint.

'To describe any 3D point in a point cloud is by specifying it's x,y, and z components. An alternative representation of a 3D point is obtained by specifying angles theta, phi, and a distance. Theta and phi in this case specify the angles of a ray coming out of the origin (or any other viewpoint). The distance you must go along the ray to reach a point in the point cloud, is the depth value.'

[5]

2.3.1 From XDM to Point Cloud

This approach has two important steps:

- The depth values read from Intel RealSense camera are stored in a destination array.

- This data is then filtered in the second step and transformed into a three dimensional mesh consisting of 16 bit depth value (type int) in the two dimensional array.

Process

Once we have initialised the source and destination arrays, every 16 bit depth value for each X,Y pair is read from the 2D depth array.

Each depth value represents a distance , which while constructing a point cloud, we take to be the Z value (distance). Adding this to be the Z coordinate of the corresponding entry in the 3 dimensional destination array, we get a simple conversion from XDM to point cloud. This destination array is 320 vertices wide, 480 vertices high and represents the maximum depth range along the z axis.

CODE EXAMPLE 1. Creating the point cloud data structure

```
// basic vector structure and point cloud dataset array
struct point_cloud
{
float x;
float y;
float z;
}
vec3* dataset = new vec3[depthwidth*depthheight];
dataset[(y*depthwidth)+x].x=(float)x;
dataset[(y*depthwidth)+x].y=(float)y;
dataset[(y*depthwidth)+x].z=(float)depthdistance;
```



Figure 2.5: XDM image

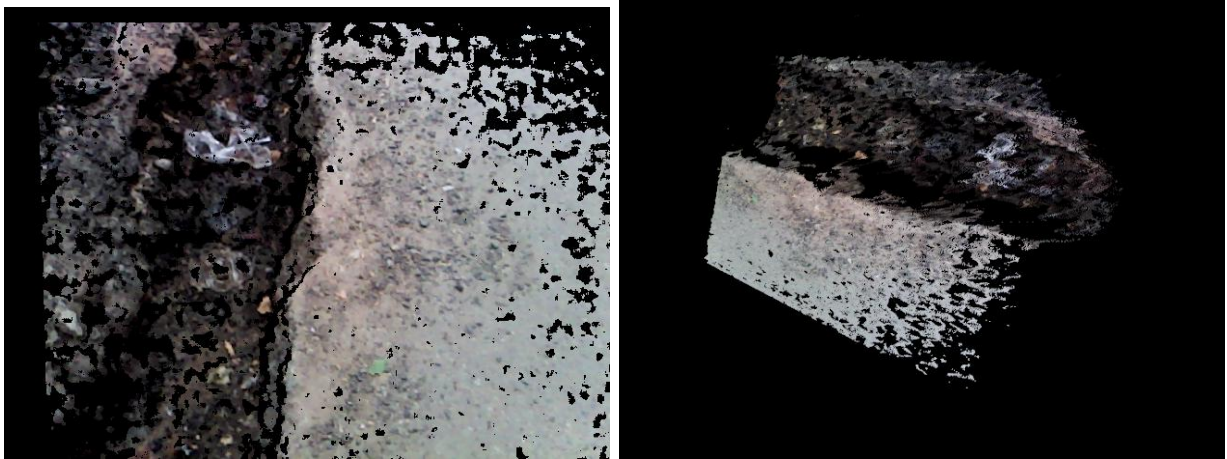


Figure 2.6: Corresponding Point Cloud

2.3.2 Real-Time Point Cloud Capture

Converting XDM generated from the SDK to point clouds is simple once the code has been compiled however it forbids capture of point clouds directly. For this, `realsenseviewer.exe` provides the necessary implementation[26]. This has been implemented on Windows since it uses many library methods provided within the Intel RealSense SDK exclusively for Windows OS. Apart from Windows it also requires PCL version 1.4 or later. Its usage is as follows :

```
real_sense_viewer.exe [Options] device_id
```


'If device-id is not given, then the first available device will be used. If capture mode is not given, then the grabber will try to enable both depth and color streams at VGA resolution and 30 Hz framerate. If this particular mode is not available, the one that most closely matches this specification will be chosen.'

[26]

When the focus is on the viewer window, the following keyboard commands are available:

- t/T : increase or decrease depth data confidence threshold
- k : enable next temporal filtering method
- b : toggle bilateral filtering
- a/A : increase or decrease bilateral filter spatial sigma
- z/Z : increase or decrease bilateral filter range sigma
- s : save the last grabbed cloud to disk
- h : print the list of standard PCL viewer commands

2.4 Dataset

Our dataset consists of 100 potholes in and around the IIT campus. They have been chosen as to represent sufficient variation. For each pothole, there are 3 XDMs taken either at different angles and/or ambient conditions. Also, for each pothole, 1 point cloud has been captured. After filtering for 'bad' quality XDMs and/or Point Clouds, we identify a set of 50 potholes which have been used for generating the results part of this project.

2.5 Challenges

Being a technology in development, several issues continue to arise. Some of the major challenges faced in data collection are enlisted below

2.5.1 Platform dependence

Although librealSense provides many of the essential features part of the Intel RealSense SDK, it is not yet complete and plagued with many issues. Most of the vision algorithms are provided only as part of the SDK, which limits the use. Some significant challenges while working with RealSense on Ubuntu are :

- **Manual Refresh** : It is subject to crash after each capture, also crashes at startup when plugged in before booting the viewer.

- **RGB-D alignment** : For applications where we need to sync the results produced from RGB processing and Depth processing separately, support is not currently provided for OS other than Windows
- **Core Dumps** : Floating point exceptions are generated when using callbacks in Ubuntu
- **GLFW/UVC** : GLFW and UVCbackend are prerequisites for installing librealsense on Ubuntu. They come with their own share of issues like 'GLFW variables not found' which arises often especially while testing the measurement API
- Infrared callback is triggered while when only the IR/depth stream is enabled.

2.5.2 Effect of Ambient/Device Conditions

Ambient conditions hugely impact the quality of the resulting images, since the Infrared component of sunlight interferes with the device IR. This may be resolved by resorting to use in night light, since light from lamps has no IR component hence the results contain slightly better quality of both RGB and depth components. Some ambient factors which affect the output are :

- The environment's temperature (effect on sensors)
- The environment's ambient IR light
- Temporal (frame-to-frame) and Spatial (within a frame) variation
- Sensor to sensor alignment and each sensor's field of view (FOV)
- Any other physical effect on a sensor including physical impact or vibration
- The location and type of target, especially the object's IR reflectivity and texture (sheen, dimples, and curvature)[15]

The following example demonstrates 'good', 'fair' and 'bad' quality depth images for the similar scene but with differing ambient conditions.



Figure 2.7: 'GOOD' quality depth map, with discernible variation in depthmap



Figure 2.8: 'FAIR' quality depth map, with less discernible variation



Figure 2.9: 'BAD' quality depth map, with negligible variation in depthmap

Chapter 3

DEPTHMAP PROCESSING

The processing, as outlined in the first chapter, is done on the two classes of depthmaps separately. Firstly the implementation steps for the top-down view are described, which included depthmaps captured parallel to the plane of interest i.e. the ground. This is similar in nature to the steps taken by Moazzam et al. in that we know beforehand that the background region surrounding the pothole is expected to have consistent depth values without variation. Any variation, then, implies the presence of pothole. This is then further segmented and quantified as per the steps detailed below.

For the second, more general case, which includes depthmaps captured at an angle of approximately 22.5 degrees to the normal (this equals the tilt in the model prototype to reduce the effect of ambient conditions and maximize the ground plane area captured), two approaches are attempted namely, depthmap processing and point cloud processing. Both offer their advantages and disadvantages. However, due to critical challenges encountered and explained in the further sections, the desired end result has not yet been attained. Several heuristic strategies have been attempted and their implementation details and results are presented, along with directions for further research.

3.1 Top Down view

We begin by extracting the depthmaps from the XDMs captured and then process them through a series of steps as outlined below.

3.1.1 XDM to DepthMap

This is done using XDM depth parser which is described in section 2.1.2

```
input : XDM file
java -getdepthmap XDMfile
output : XMP data , 16-bit grayscale depthmap
```

However, the produced depthmap has very close contrast values since they occupy a narrow range of depth values.

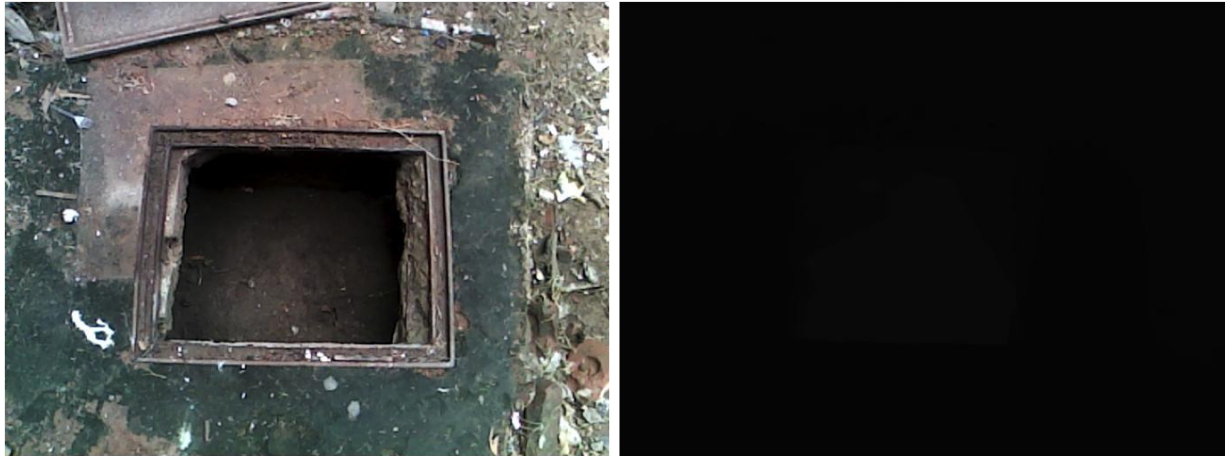


Figure 3.1: XDM to depthmap

3.1.2 Histogram Equalization

Since the difference in contrast values of the obtained depthmap is imperceptible, the histogram of the resulting depthmap is equalized in order to be able to visualise the effects of the various steps.

It works as follows[12] :

- Equalization implies mapping one distribution (the given histogram) to another distribution (a wider and more uniform distribution of intensity values) so the intensity values are spreaded over the whole range.
- To accomplish the equalization effect, the remapping should be the cumulative distribution function (cdf)
- Finally, we use a simple remapping procedure to obtain the intensity values of the equalized image.

OpenCV provides inbuilt method to accomplish this. The main method is :

```
void equalizeHist(InputArray src, OutputArray dst);
```

src - Source 8-bit single channel image.

dst - Destination image of the same size and type as src.

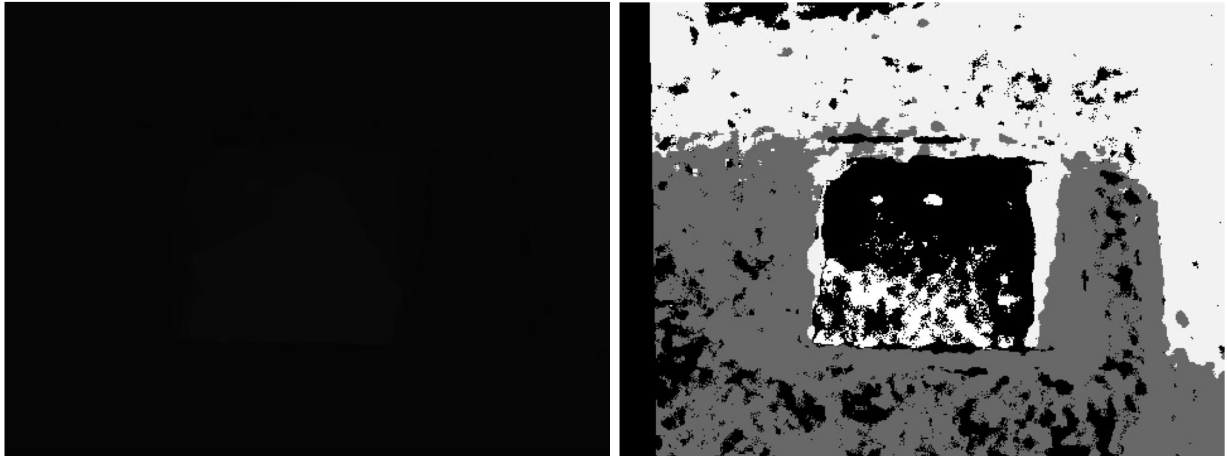


Figure 3.2: XDM to depthmap

This is further smoothed using OpenCV's Gaussian Filter which uses Gaussian Blur to smooth the source image.

```
void GaussianBlur(InputArray src, OutputArray dst, Size ksize, double sigmaX,
double sigmaY=0, int borderType=BORDER_DEFAULT );
```

PARAMETERS :

src - input image; the image can have any number of channels

dst - output image of the same size and type as src

ksize - Gaussian kernel size

sigmaX - Gaussian kernel standard deviation in X direction

sigmaY - Gaussian kernel standard deviation in Y direction

borderType - pixel extrapolation method

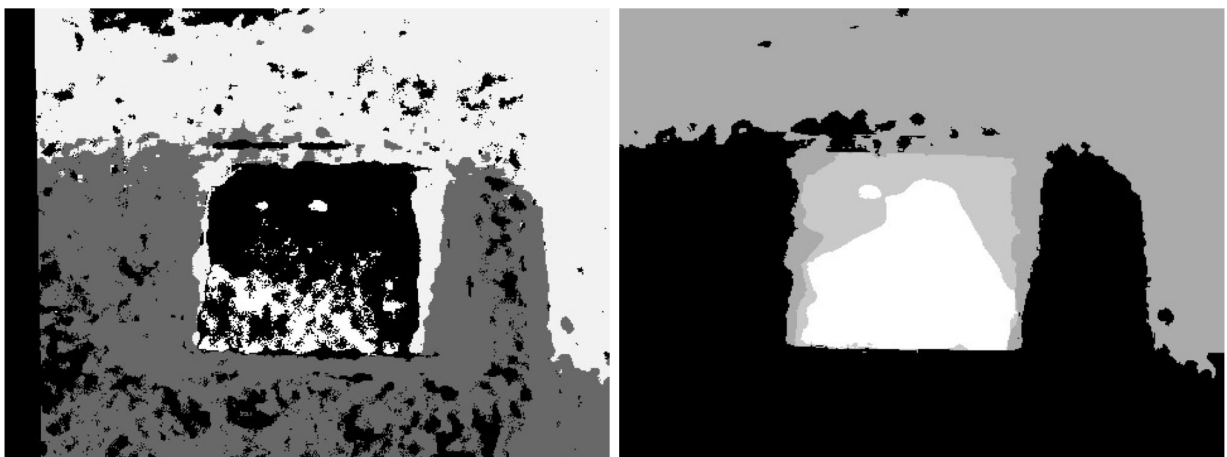


Figure 3.3: Gaussian Blur smoothing

3.1.3 Canny Edge Detection

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. Its algorithm involves the 5 steps in order : Gaussian filter to remove noise, find intensity gradients, apply non-maximum suppression to get rid of spurious response to edge detection, apply double threshold to determine potential edges and finally track edges by hysteresis.[6] OpenCV allows this with the function:

```
void cvCanny(const CvArr* image, CvArr* edges, double threshold1,  
double threshold2, int aperture_size=3 )
```

PARAMETERS:

image - single-channel 8-bit input image

edges - output edge map

threshold1 - first threshold for the hysteresis

threshold2 - second threshold for the hysteresis procedure

apertureSize - aperture size for the Sobel() operator

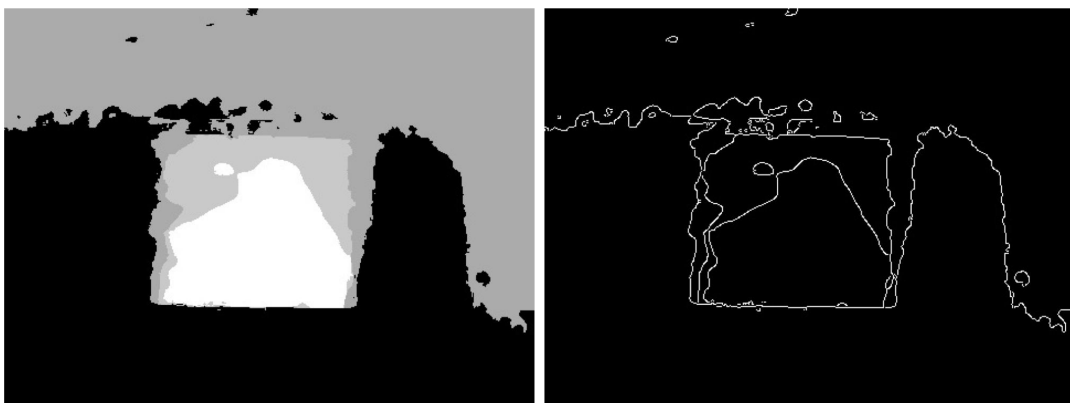


Figure 3.4: Canny Edge detection

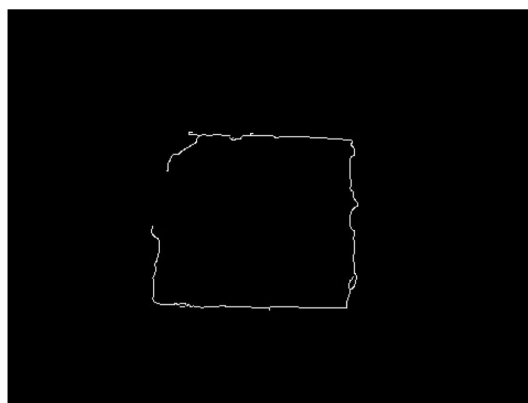


Figure 3.5: Canny with increased threshold

3.1.4 Hough Transform

For feature detection in case of potholes having approximately rudimentary shapes of circle or rectangular, we use Hough Transform to find imperfect cases of objects of the fore said two shape classes using a voting procedure carried out within a parameter space, from which object candidates are transferred (the local maximas) into the accumulator space constructed explicitly for computing Hough Transform. Current implementation works only for circles and rectangles.

To detect circles (which we expect a large class of potholes to be shaped approximately like) we use OpenCV Hough Circles method[13].

```
void HoughCircles(InputArray image, OutputArray circles, int method,
double dp, double minDist, double param1=100, double param2=100,
int minRadius=0, int maxRadius=0 )
```

PARAMETERS:

circles - Output vector of found circles
dp - Inverse ratio of the accumulator resolution to the image resolution
minDist - Minimum distance between the centers of the detected circles
minRadius - Minimum circle radius
maxRadius - Maximum circle radius

HoughLines method[14] is used to detect rectangles based on the presumption that the sides of the rectangle are the most prominent lines in the image. 4 biggest peaks in hough space then become the sides of the detected rectangle.

However, this can be done more efficiently by the built in methods.

```
void HoughLines(InputArray image, OutputArray lines, double rho,
double theta, int threshold, double srn=0, double stn=0 )
```

PARAMETERS:

lines - Output vector of lines
rho - Distance resolution of the accumulator in pixels
theta - Angle resolution of the accumulator in radians
threshold - Accumulator threshold parameter
method - CV_HOUGH_STANDARD / CV_HOUGH_PROBABILISTIC

3.1.5 MeasureDistance

Finally, once the circle/rectangle has been fitted, we pass the output lines vectors/ circle vector (in both cases, the lowest and uppermost points are passed. In trivial case for rectangle, the mid points of opposite lines are passed) to the MeasureDistance method of Intel RealSense SDK.[8] It computes and returns the distance between the two points along with confidence value.

PXCEnhancedPhoto::PhotoUtils::EnhancedDepth::MeasureDistance



Figure 3.6: Initial image and Final output

output :

Distance between (305,146), (301,361) = 45.05 cm

Confidence = 0.95

Precision = 62258.25 mm

For convenience, the series of steps for two sample images including the one in consideration are shown together :

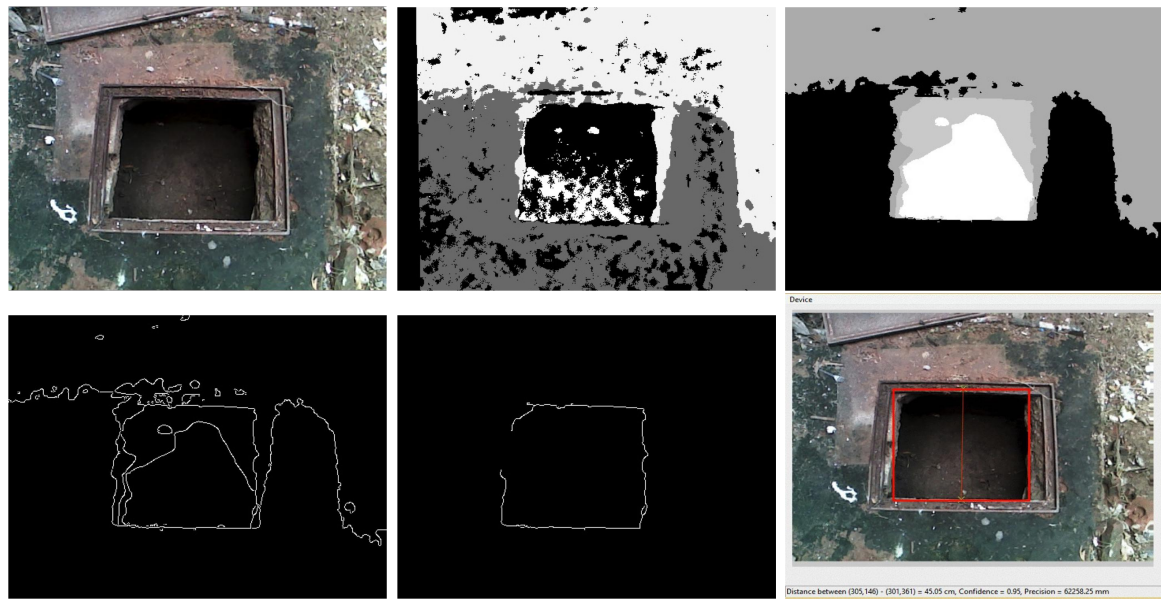


Figure 3.7: XDM to final result - Rectangle

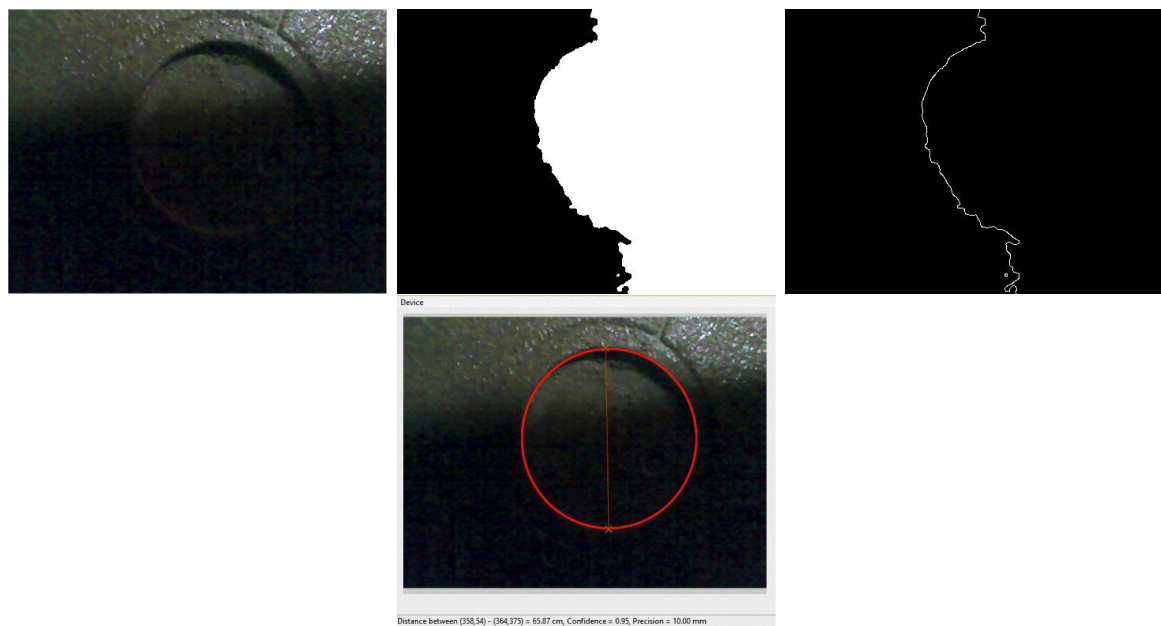


Figure 3.8: XDM to final result - Circular

output for example #2 :

Distance between (358,54),(364,375) = 65.87 cm

Confidence = 0.95

Precision = 10.00 mm

3.2 Inclined View

We first attempt to address the challenges faced when applying the previous approach to inclined case. Then another approach of processing point clouds is discussed in depth.

3.2.1 Banding Effect

Banding is a problem of inaccurate colour presentation in computer graphics. In 24-bit colour modes, 8 bits per channel is usually considered sufficient to render images. For our purpose, RealSense generates 16 bits per channel (grayscale channel).

Colour banding is more noticeable with fewer bits per pixel (BPP), where every shade can not be rendered because there are insufficient bits. Hence the first approach is to create a colormap of the corresponding images to analyze whether the other 8 bits can be represented in a different channel thereby eliminating the banding effect.

This effect is especially visible in inclined shots, where the depth gradient progressively increases. The effect is generated by certain methods of OpenCV and can be seen on the images below.

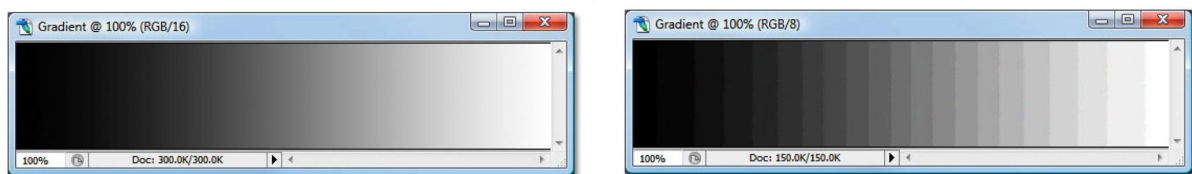


Figure 3.9: Banding effect w.r.t. gradient



Figure 3.10: Original XDM images

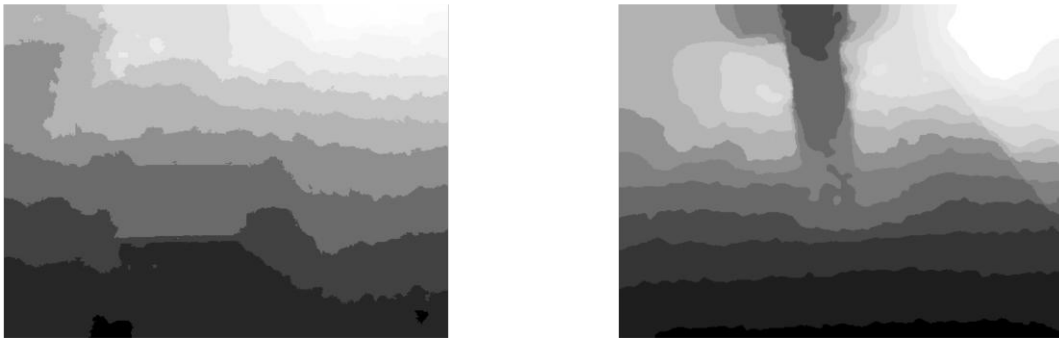


Figure 3.11: Downsized to 8 bit depthmap - banding effect

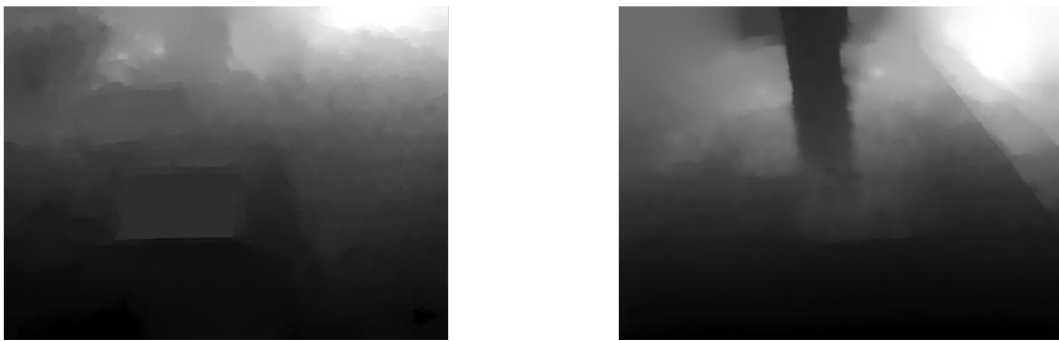


Figure 3.12: 16 bit depthmap

Image		Image	
Dimensions	480 x 360	Dimensions	480 x 360
Width	480 pixels	Width	480 pixels
Height	360 pixels	Height	360 pixels
Bit depth	16	Bit depth	8
File		File	
Name	2.png	Name	2_1.png
Item type	PNG image	Item type	PNG image
Folder path	C:\Users\SachinYadav1993\Documen...	Folder path	C:\Users\SachinYadav1993\Documen...

Figure 3.13: Downsizing by OpenCV

Colormap and OpenCV downsizing

This effect is apparently produced by certain methods of OpenCV. With the reason unclear, it was tested whether the effect is there on manually downsizing the images and also checking whether generating the colormap will produce extra detail (in case only 8 bits are allowed per channel, the other 8 bits could be channeled to a different color component in case of colormap.)

However, as shown below, there is no perceptible difference between the 16 bit, 8 bit and colormap components indicating that this effect is generated exclusively by some OpenCV method. For this reason and due to lack of 3 dimensional processing algorithms in OpenCV, a radically different approach involving point cloud library was used to test the various heuristics. This is explained in further sections.



Figure 3.14: Original Image XDM



Figure 3.15: Corresponding 8 bit, 16 bit and colormap

3.2.2 Point Cloud approach

Our approach to detect potholes and other depth variations once we have captured the point clouds is outlined below :

- **Outlier removal** : Remove outliers for better visualization as well as preparing the cloud for next operations which may not work with outliers
- **Normals Estimation** : The input cloud has calibration and point of capture data but does not contain the normal information itself. This is required for many point cloud library methods

- **Plane Segmentation** : This step segments out the plane of interest , independent of the angle of the image, once the normals have been estimated.
- **Cloud concatenation / Cluster extraction / Region growing segmentation** : Various heuristic approaches are attempted after the plane has been segmented. The simplest and most intuitive is the subtract the plane from the original cloud with outliers removed. Ideally, it should act as a filter for only the points below/above a certain threshold from the plane to pass. However, various format irregularities and dataset challenges are encountered which have been subsequently discussed.

3.2.3 Point Cloud outlier removal

'The point cloud grabber typically generate point cloud datasets of varying point densities. Additionally, measurement errors lead to sparse outliers which corrupt the results even more. This complicates the estimation of local point cloud characteristics such as surface normals or curvature changes, leading to erroneous values, which in turn might cause point cloud registration failures. Some of these irregularities can be solved by performing a statistical analysis on each point's neighborhood, and trimming those which do not meet a certain criteria. PCL's sparse outlier removal is based on the computation of the distribution of point to neighbors distances in the input dataset. For each point, we compute the mean distance from it to all its neighbors. By assuming that the resulted distribution is Gaussian with a mean and a standard deviation, all points whose mean distances are outside an interval defined by the global distances mean and standard deviation can be considered as outliers and trimmed from the dataset.'

[25]


Figure 3.16: Original Image XDM



Figure 3.17: Outlier removal

Cloud before filtering:

header:

seq: 0

stamp: 0.000000000

frame_id:

points[]: 1420910

width: 1420910

height: 1

is_dense: 0

Cloud after filtering:

header:

seq: 0

stamp: 0.000000000

frame_id:

points[]: 1239398

width: 1239398

height: 1

is_dense: 0

3.2.4 Normals Estimation

Given a geometric surface, it's usually trivial to infer the direction of the normal at a certain point on the surface as the vector perpendicular to the surface in that point. However, since the point cloud datasets that we acquire represent a set of point samples on the real surface, there are two possibilities:

- obtain the underlying surface from the acquired point cloud dataset, using surface meshing techniques, and then compute the surface normals from the mesh;
- use approximations to infer the surface normals from the point cloud dataset directly.

For our purpose we choose to do the latter, simpler approach. The problem of determining the normal to a point on the surface is approximated by the problem of estimating the normal of a plane tangent to the surface, which in turn becomes a least-square plane fitting estimation problem. The solution for estimating the surface normal is therefore reduced to an analysis of the eigenvectors and eigenvalues (or PCA – Principal Component Analysis) of a covariance matrix created from the nearest neighbors of the query point.[9]

Without going into the implementation details, which are taken care of by the inbuilt methods of PCL, we proceed to the next step of Plane Segmentation.

3.2.5 RANSAC Plane Segmentation

The RANSAC algorithm is a learning technique to estimate parameters of a model by random sampling of observed data. Given a dataset whose data elements contain both inliers and outliers, RANSAC uses the voting scheme to find the optimal fitting result. Data elements in the dataset are used to vote for one or multiple models. The implementation of this voting scheme is based on two assumptions: that the noisy features will not vote consistently for any single model (few outliers) and there are enough features to agree on a good model (few missing data). The RANSAC algorithm is essentially composed of two steps that are iteratively repeated:[21]

- In the first step, a sample subset containing minimal data items is randomly selected from the input dataset. A fitting model and the corresponding model parameters are computed using only the elements of this sample subset. The cardinality of the sample subset is the smallest sufficient to determine the model parameters.
- In the second step, the algorithm checks which elements of the entire dataset are consistent with the model instantiated by the estimated model parameters obtained from the first step. A data element will be considered as an outlier if it does not fit the fitting model instantiated by the set of estimated model parameters within some error threshold that defines the maximum deviation attributable to the effect of noise.[22]

```
$ ./planar_segmentation
```

```
Point cloud data: 154189 points
```

```
[pcl::SACSegmentation::initSAC] Setting the maximum number of iterations to 50
```

```
Model coefficients: 0 0 1 -1
```

```
Model inliers: 112891
```




Figure 3.18: Before plane segmentation (eg. 1)

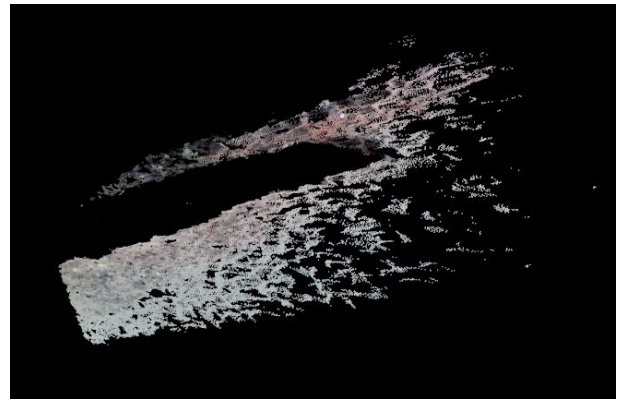
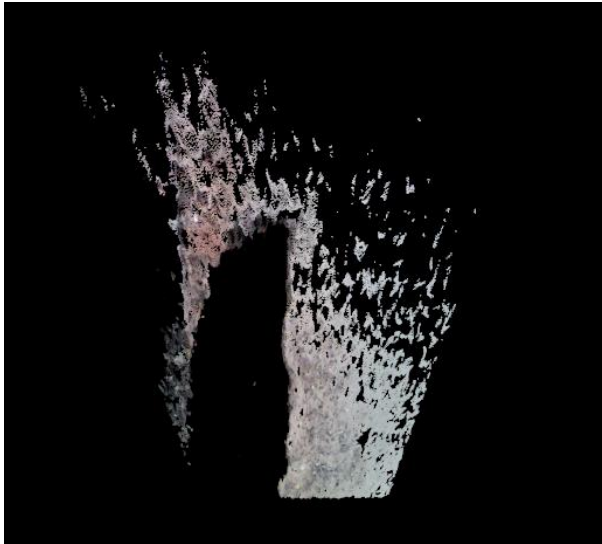


Figure 3.19: Segmented Plane (eg. 1)



Figure 3.20: Before plane segmentation (eg. 2)



Figure 3.21: Segmented Plane (eg. 2)

3.2.6 Region Growing Segmentation based Cluster Extraction

Out of many options to extract the potholes namely plane subtraction or region growing segmentation, the latter was chosen because of its fast implementation available via point cloud library. [5]

'Firstly the points are sorted by their curvature value as the region begins from a minimum curvature point as it will be located in a flat area.(growth from the flattest area allows to reduce the total number of segments). Once we have the sorted cloud, algorithm picks up the point with minimum curvature value and starts the growth of the region as follows'[24]:

- The picked point is added to the set called seeds.
- For every seed point algorithm finds neighbouring points.
 - Every neighbour is tested for the angle between its normal and normal of the current seed point. If the angle is less than threshold value then current point is added to the current region.
 - After that every neighbour is tested for the curvature value. If the curvature is less than threshold value then this point is added to the seeds.
 - Current seed is removed from the seeds.

It runs well on organised point clouds of the publicly available pcl point cloud dataset (table.pcd and wall.pcd). However due to format incompatibility and NaN issues, the point clouds obtained from the pcl realsense grabber remain unorganised and unsuitable for this process. These challenges are explained and demonstrated in the next section.

3.2.7 Challenges

NaN(Not a Number) and Inf values

Intel RealSense produces NaN and Inf values for a couple of reasons. Mostly it is if the confidence threshold for a particular depth value is not reached, either because of camera movement, ambient conditions or for many other intrinsic causes like range-related issues. This problem becomes particularly acute when the shots are taken inclined. For about every degree of tilt in the incline, the number of NaN values seem to increase exponentially.

Such point clouds produce error and/or exceptions in most of the methods pertaining to the point cloud library, since there are hardly any high confidence points when the nearest neighbor calculations are done, both for region growing segmentation and cluster extraction.

The figure below demonstrates the point cloud for a shot taken in broad daylight with 11 degrees of tilt from the vertical, causing the horizon line to appear midway in the XDM image. About 71 percent points in this image are taken to be NaN values. Even though this is an extreme example, the average NaN values for a general image stand around 50 percent, severely limiting the methods that can be deployed to process the point clouds. The RealSense SDK produces XDM images after pre processing the depthmaps for such 'holes'. However, this smoothing technique is at best approximate and hides the depth field variations which we are looking for. This happens because for our use case, there is no clear background-foreground separation making the segmentation process subject to even little approximations and error margins.



Figure 3.22: Many outliers, 70 percent NaN inliers

Format Incompatibilities

Different methods of the point cloud library demand the input point clouds in different formats and with different requirements. The cloud generated by realsenseviewer is neither normal estimated nor organised, moreover methods like ransac plane segmentation degenerate even an organised point cloud to an unorganised one.

3.3 Summary Of Results

3.3.1 Results for Hough Transform

	Images	Potholes	Misc	Identified	Success	Measurement error	False Positives
HoughCircles	23	26	7	17	65 %	18%	2

Table 3.1: HoughCircle depthmap processing

	Images	Potholes	Misc	Identified	Success	Measurement error	False Positives
HoughRectangle	14	12	2	7	58 %	21%	0

Table 3.2: HoughRectangle depthmap processing

3.3.2 Results for PCD processing

	Point Clouds	Segmented	Average time	Success rate
ransac	21	20	6.35 s	95.24 %

Table 3.3: Ransac results : 'FAIR' quality pcd

	Point Clouds	NaN Values	Segmented	Success rate
ransac	29	57%	11	37.93%

Table 3.4: Ransac results : 'BAD' quality pcd

Chapter 4

CONCLUSION AND FUTURE WORK

4.1 Conclusion

Intel RealSense is ideally suited for indoor applications where the separation between foreground and background is significant, for example applications such as Gesture recognition.

For the purpose of pothole detection in top down images, it works exceedingly well for rudimentary shapes of that of circles and rectangles. However, as potholes and obstacles are expected to be haphazard in shape, and the prototype can not be mounted top-down, it fails to deliver in practical outdoor circumstances with the camera mounted on the chest at an angle.

The official documentation assumes the range to be 1-10 metres. However, the experiments carried out demonstrate the destructive effect of ambient conditions especially as the range is extended by increasing the inclination. The number of invalid values and 'holes' seem to increase exponentially with the change in the inclination of the mount. This combined with the destructive IR interference and the constant camera movement, renders the output clouds unfit for processing.

Finally, on the basis of the experimentation carried out, I conclude that using Intel RealSense R200 for the purpose of stand alone pothole detection in the MAVI system is **currently unfeasible** for practical real-time purpose. Although it is cost effective as compared to industrial cameras and lasers, the use of infrared technology based on Kinect sensor for measurement is still a novel idea and further research is necessary for improvement in error rate. However, more research should be carried out since this technology and its supporting technologies are fast developing.

4.2 Future Work

Future work in this domain may be directed at the following issues:

- Assessing the camera performance in night light conditions , where there is no destructive interference, although RGB quality is compromised
- The feasibility of using Intel RealSense R200 in conjunction with a good quality RGB camera with an objective to reduce the number of false positives

- Feasibility study for using IntelRealSense R200 as an indoor navigation tool
- Detection of animals, stairs and such obstacles where either the foreground-background separation is vivid or the obstacles are patterned (stairs)
- Comparison of Kinect and Intel RealSense technologies in order to provide a baseline to adapt Kinect based studies to this newer technology.

Bibliography

- [1] Jahanshahi et al. “Unsupervised Approach for Autonomous Pavement-Defect Detection and Quantification Using an Inexpensive Depth Sensor”. In: *Journal of Computing in Civil Engineering* (2013).
- [2] Moazzam et al. “Metrology and Visualization of Potholes using the Microsoft Kinect Sensor”. In: *IEEE Annual Conference on Intelligent Transportation Systems* (2013).
- [3] Taehyeong Kim et al. “Review and Analysis of Pothole Detection Methods”. In: *Journal of Emerging Trends in Computing and Information Sciences* (2014).
- [4] Lee Bamber. “Dipping into the Intel® RealSense™ Raw Data Stream”. In: (2014). URL: <https://software.intel.com/en-us/articles/dipping-into-the-intel-realsense-raw-data-stream?language=es>.
- [5] Lee Bamber. “Intel® RealSense™ Technology and the Point Cloud”. In: (2015). URL: <https://software.intel.com/en-us/articles/intel-realsense-technology-and-the-point-cloud>.
- [6] *Canny Edge Detection OpenCV*. URL: http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html.
- [7] “Datasheet for the Intel® RealSense™ Camera R200”. In: (2016). URL: <http://www.intel.com/content/www/us/en/support/emerging-technologies/intel-realsense-technology/000023534.html>.
- [8] “Documentation for Intel RealSense SDK”. In: (2016). URL: <https://software.intel.com/en-us/intel-realsense-sdk/documentation>.
- [9] *Estimating Surface Normals in a PointCloud*. URL: http://pointclouds.org/documentation/tutorials/normal_estimation.php.
- [10] *Euclidean Cluster Extraction*. URL: http://www.pointclouds.org/documentation/tutorials/cluster_extraction.php.
- [11] Muneeb Fazal. “Modelling and Implementation of Face Detection and Recognition for Mobility Assistant for Visually Impaired System (MAVI)”. In: *IIT Delhi CSE* (2016).
- [12] *Histogram Equalization OpenCV*. URL: http://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/histogram_equalization/histogram_equalization.html.
- [13] *Hough Circle Transform OpenCV*. URL: http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html.

- [14] *Hough Line Transform OpenCV*. URL: http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html.
- [15] *Intel RealSense official page*. URL: <http://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html>.
- [16] *Intel RealSense Wiki*. URL: https://en.wikipedia.org/wiki/Intel_RealSense.
- [17] Houman A. (Intel). "The eXtensible Device Metadata (XDM) Specification - Version 1.0". In: (2015). URL: <https://software.intel.com/en-us/articles/the-extensible-device-metadata-xdm-specification-version-10>.
- [18] *librealsense github repository*. URL: <https://github.com/IntelRealSense/librealsense>.
- [19] kamal et al. Moazzam. "Performance assessment of Kinect as a sensor for pothole imaging and metrology". In: *International Journal of Pavement Engineering* (2016).
- [20] *Mobility Assistant for Visually Impaired*. URL: <http://www.cse.iitd.ac.in/mavi/>.
- [21] *Plane model segmentation*. URL: http://pointclouds.org/documentation/tutorials/planar_segmentation.php.
- [22] *Random Sample Consensus model*. URL: http://www.pointclouds.org/documentation/tutorials/random_sample_consensus.php.
- [23] *RealSense technology Intel page*. URL: <https://software.intel.com/en-us/articles/realsense-r200-camera>.
- [24] *Region growing segmentation*. URL: http://pointclouds.org/documentation/tutorials/region_growing_segmentation.php.
- [25] *Removing outliers using a StatisticalOutlierRemoval filter*. URL: http://pointclouds.org/documentation/tutorials/statistical_outlier.php.
- [26] *taketwo. real sense pcl grabber*. URL: <https://github.com/taketwo/rs>.