3D Reconstruction of Environment using RGB-D Imaging

A project report submitted in partial fulfillment of the requirements for the degree of

Bachelor of Engineering

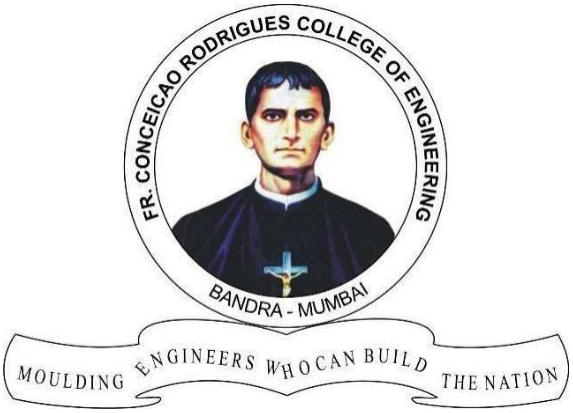
by

Christo Aluckal (Roll No. 7917)
Sumedh G. Deshpande (Roll No. 7927)
Karan Rao (Roll No. 7965)

Under the guidance of

Dr. Brijmohan Daga

Dr. Yogesh Agarwadkar (InfiCorridor Solutions Pvt. Ltd.)



DEPARTMENT OF COMPUTER ENGINEERING

Fr. Conceicao Rodrigues College of Engineering, Bandra (W), Mumbai – 400050

University of Mumbai

June 15, 2020

Abstract

Development in the field of mobile robots has increased exponentially in the last couple of years. They have become compact, robust, highly maneuverable and are able to carry a wide variety of sensors and equipment. However, most commercial implementations of mobile robots occur either outdoors with the help of signals such as GPS or require a priori knowledge of the indoor environment. In this project, we aim to develop a methodology that will assist in exploration and navigation of indoor environments using RGB-D images. The proposed method will generate 2D maps for the exploration and navigation as well as generate 3D models of the mapped environment.

Table of Contents

Table of Figures

Table of Tables

Chapter 1 : Introduction

Navigation has become an important part of our lives and although much research has been done in using the navigation on the outside, considerably less research has been done in terms of navigating indoors. Most of the technologies used before require prior setup to be made and hence require knowledge of the environment beforehand. These include WiFi and Bluetooth beacons, GPS sensors etc. all of which prove to be less viable of an option in terms of real-time implementation. Unmanned Autonomous Vehicles especially aerial vehicles have seen large scale development and improvements over the past few years.

Modern UAVs are very compact and customizable. They work very well in outdoor environments due to their usage of GPS, however indoor navigation is still extremely complicated due to the existence of a number of complexities which include dynamic obstacles, clutter and a lack of any meaningful GPS signals. The project aims to make 2D and 3D maps and models using RGBD imaging which can be reconstructed statically or dynamically. The project is divided into two groups. One for navigation and reconstruction. The navigation team works on algorithms to explore an unknown environment while the reconstruction team works on algorithms to provide the 2D and 3D maps and models required to navigate in these unexplored areas. We are performing a comparative analysis of the various methods to reconstruct environments.

Chapter 2: Literature Review

2.1 RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments

2.1.1 Background

In this paper, the authors introduce RGB-D Mapping. It is a framework for using RGB-D cameras to generate dense 3D models of indoor environments. RGB-D Mapping exploits the integration of shape and appearance information provided by these systems. Alignment between frames is computed by jointly optimizing over both appearance and shape matching. The approach detects loop closures by matching data frames against a subset of previously collected frames. To generate globally consistent alignments TORO is applied which is an optimization tool developed for SLAM. The overall system can accurately align and map large indoor environments in near-real time and is capable of handling situations such as featureless corridors and completely dark rooms.

2.1.2 Literature Review

The authors implement the following methods to generate a 3D model

- RGB+Depth: This provides a stream of colour and depth images that will be used in the algorithm
- Feature Matching: Sparse visual features are extracted from two frames and associate them to depth values to generate feature points in 3D
- RANSAC: RANSAC provides the transformation between the feature sets. It is an iterative optimization method
- ICP: After RANSAC, ICP determines the transformation between two point clouds that provide the least error

- Loop Closure Detection: LCD is used to reduce the cumulative error in frame alignment. It uses multiple frames to determine if the current path taken has been taken before
- Surfel Map Generation: A surfel consists of a location, a surface orientation, a patch size and color. Surfels store a measure of confidence, which is increased through being seen from multiple angles over time. Surfels with low confidence are removed from the representation

2.1.3 Summary

YEAR	AUTHOR	COUNTRY	OBJECTIVE	CONTRIBUTION	METHOD	CONCLUSION
APRIL 2012	Henry, Peter & Krainin, Michael & Herbst, Evan & Ren, Xiaofeng & Fox, Dieter.	U.S.A.	In this paper the authors introduce RGB-D Mapping. A framework for using RGB-D cameras to generate dense 3D models of indoor environments.	The authors investigate how potentially inexpensive depth cameras developed mainly for gaming and entertainment applications can be used for building dense 3D maps of indoor environments.	RGB + Depth Feature Matching RANSAC ICP Loop Closure Detection Surfel Map Generation	The authors present RGB-D Mapping, a full 3D mapping system that utilizes a novel joint optimization algorithm combining visual features and shape-based alignment.

Table 2.1 Summary for Paper I

2.1.4 Research Gap

- Per frame, our current implementation extracts features in 150 ms, runs RANSAC in 80 ms, and runs dense ICP in an average of 500 ms. Surfel generation takes roughly 6 seconds per frame
- RANSAC is the faster and more reliable alignment component when considered individually. However, there are situations where it fails and the joint optimization is required
- For some frames, many detected visual features are out of range of the depth sensor, so those features have no associated 3D points and do not participate in the RANSAC procedure
- When the majority of the features lie in a small region of the image, they do not provide very strong constraints on the motion
- The current implementation of RGB-D Mapping is not real-time. The global alignment process of RGB-D Mapping is still limited. Instead of optimizing over camera poses, a joint optimization over camera poses and 3D points could result in even more consistent reconstructions

2.2 PROBLEMS IN INDOOR MAPPING AND MODELLING

2.2.1 Background

Research is being actively conducted in the field of Indoor Mapping and Modelling(IMM) for more than thirty years. This research has come in the form of As-Built surveys, Data structuring, Visualisation techniques, Navigation models and so forth. Much of this research is founded on advancements in photogrammetry, computer vision and image analysis, computer graphics, robotics, laser scanning and many others. This paper considers both existing and emerging problems in IMM that are relevant to commercial enterprises and the general public, groups this paper expects will emerge as the greatest users IMM and also, this paper discusses the various possible applications of IMM. Existing problems are those that are currently being researched. These will hopefully provide a framework for assessing progress and advances in indoor modelling. The framework will be detailing existing and emerging problems, their solutions and present best practices.

2.2.2 Literature Review

The main aim of this paper is to highlight the major problems and difficulties encountered in the process of indoor mapping and modelling

Some of the major issues mentioned by the authors are:-

1. Variable lighting conditions: Variable lighting conditions affect the process of indoor mapping when using vision based systems by inducing errors and discontinuities due to the variation of light in indoor areas. The presence of surfaces that reflect/refract light have a similar effect
2. Variable occupancy of indoor spaces: Many measurement systems are designed for environments that contain as few visual obstructions as possible. Indoor environments are

often busy and cluttered. Measurement under these conditions is difficult, particularly for vision based systems

3. Real-time acquisition of dynamic environments: Modelling of dynamic environments in which there are many moving objects is presently done off-line. The challenge is to do it on the fly
4. Diversity of Indoor Environments: Overcoming the above problem will first require a study of the different types of indoor environments with the objective of cataloguing and categorising them
5. Software tools: There are very few dedicated tools for processing indoor data and where they exist they are usually found in research labs
6. PoI and landmarks strategies: Various strategies of Points of Interest and landmarks have been investigated to facilitate outdoor navigation (Peters et al 2010). However these cannot be readily applied for indoor environments. Different indoor objects might be identified as landmarks or points of interest for the purpose of distinct applications. These objects might be even not that characteristic as the traditional outdoor landmarks
7. Complexity visualisation: The amounts of information that are available to the user will certainly increase. The challenge is to manage the display of this information without reducing its complexity
8. Real-time change visualisation: Visualising change in 2D is fairly straightforward. Visualising change in 3D is a little more difficult. The challenge is how to visualise change in 3D in real-time. This is particularly important for work with mobile devices

Some of the possible/growing applications of IMM are:

1. Indoor modelling for crisis response: Evacuating an area during a time of crisis has for decades been an active area of research. Applications for this purpose have to be able to determine safe routes and guide users through safe routes. This requires sensors to capture the state of the indoor environment, relate sensory information to the indoor model, determine viable escape routes and finally convey the escape route to the user,

either visually or aurally. Additional to this, 3D indoor models can be used in crisis simulations

2. Augmented systems: Augmented systems superimpose CGI over real world imagery with the purpose of offering the viewer an enhanced or more informative image of the real world
3. Gaming: Traditionally 3D games have relied on the design of virtual indoor environments. However, as 3D indoor models become common, it is likely that there will be greater use of 'real' indoor models. Eventually as augmented systems mature, it can be expected that a new genre of games will emerge in which games are played in real indoor environments with the benefit of augmented reality
4. Industrial applications: Measuring and discovering the composition of space, i.e. determining the content of indoor space will allow greater interactivity with the space. This applies in particular to industrial and manufacturing environments where users have to operate/navigate machinery in confined spaces
5. Natural description of indoor environments (semantics): If the context of an environment can be discerned then it stands to reason that with further work we can describe the environment in natural language. This then allows space to be described to users in ways that are familiar to them
6. Real-time decision support: A user interacts, navigates or uses an indoor environment with a purpose. If an indoor environment can be described in natural language, and provided that an AI is available, then a user's decisions in the indoor environment can be supported by the AI

2.2.3 Summary

YEAR	AUTHOR	COUNTRY	OBJECTIVE	CONTRIBUTION	DATA	CONCLUSION
DEC 2013	Sisi Zlatanovaa, George Sitholeb, Masafumi Nakagawac, Qing Zhud	South Africa	This paper discusses the various problems encountered during IMM and provides solutions to the discussed problems while simultaneously providing the real world applications of IMM.	The paper provides valuable insight about the kind of constraints and limitations one might face in the process of indoor mapping. It also discusses the various possible fields in which IMM can be used.	Acquisition here refers to the measurement techniques, sensors, media, and platforms used to acquire raw data describing the geometry and radiometry of indoor environments.	Problems encountered during IMM are discussed in detail along with proposed solutions for the same and possible applications of mapping and modelling for commercial and academic purposes have been outlined.

Table 2.2 Summary for Paper II

2.2.4 Research Gap

The problems discussed above are only a means to an end. Ultimately the goal of IMM is to allow users to interact with indoor environments in ways that enhance their use of space. Many of the problems are interlinked, i.e. and enhancement in one problem will provide improvements in the other. This is particularly the case with acquisition and modelling problems. This paper has considered existing and emerging problems in indoor mapping. The problems are not presented in any order of priority, partly because many of the problems are interlinked. The purpose of the paper is to set up a framework for discussing problems in indoor mapping and modelling. It is hoped that this framework can then be used as a platform for describing indoor

mapping and modelling research and developing benchmarks to test solutions for the problems posed here and later.

Chapter 3: Proposed System

3.1 Problem Statement

To create 2D and 3D models for addressing the issue of accurate indoor navigation.

3.2 Motivation

1. **Previous technologies** - A lot of indoor navigation was carried out with the help of ultrasonic beacons, WPS (Wi-Fi Position System), Bluetooth based approaches and so on. All of these required specialised hardware to be setup in the indoor environment and hence are not dynamic in nature which is why they cannot be used in unexplored areas. Due to this very reason there is a need for a robust system that can function without the need of any prior setup to be installed.
2. **Accuracy issue** - Localisation in indoor environments is much more challenging because of the high complexity of obstacles and clutter. Lack of line-of-sight prevents GPS to provide reliable information for the vehicle to localise itself, GPS also fails to provide any details of possible obstacles in the environment.
3. **Analysis** - Making of 2D and 3D maps makes it easier to perform analysis of this environment. It helps in identifying potential objects as well as determining optimal paths once these maps are generated.

3.3 Scope

The project aims to make 2D and 3D models using RGB-D imaging. It focuses on developing 2D and 3D maps which would be both static and dynamic.

3.4 Constraints

1. Room/interior must be sparsely populated i.e. there cannot be too much clutter
2. Since the depth camera works on the principle of light, reflective and refractive surfaces generate some error or discontinuity in the 2-D map

3. Objects in the field of view of the camera must be stationary i.e. there should be no moving object during the construction of the map
4. Suitable lighting conditions must be maintained for optimum construction of the map

3.5 Objectives

- Completed Objectives

    1. 2D Map generation using our algorithm
    2. Dynamic 2D Map generation
    3. Static 3D Map generation using the revised system
    4. Dynamic 3D Map generation

- Future Work

    1. Performing obstacle avoidance using 2D models.
    2. Identification of objects in a reconstructed environment.

3.6 Designing effective input and outputs

The fundamental flaw in the data acquisition process is that the depth camera works on the principle of stereo IR and therefore is susceptible to deviations caused by fundamental properties of light such as reflection and refraction. These deviations cause erroneous data points which are reflected in the reconstructed plot. Hence, to generate minimum deviations, care must be taken to avoid potential pitfalls in the data acquisition process. The output of the program is a file which provides representations for the data points mapped to the world coordinate system. Hence, proper file representation must be generated to avoid any unnecessary issues.

3.7 Proposed Architecture

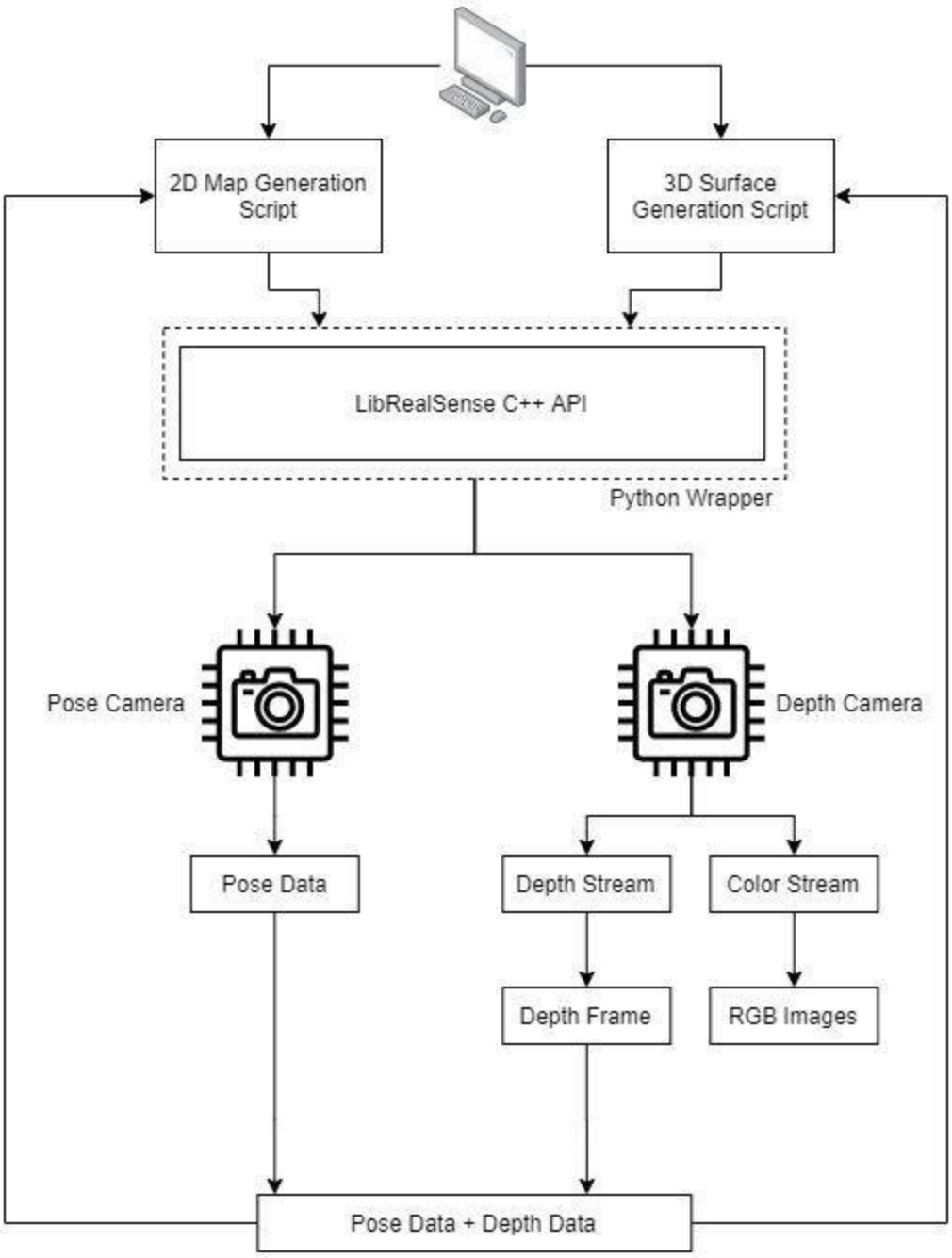


Figure 3.1 General Architecture

Architecture:

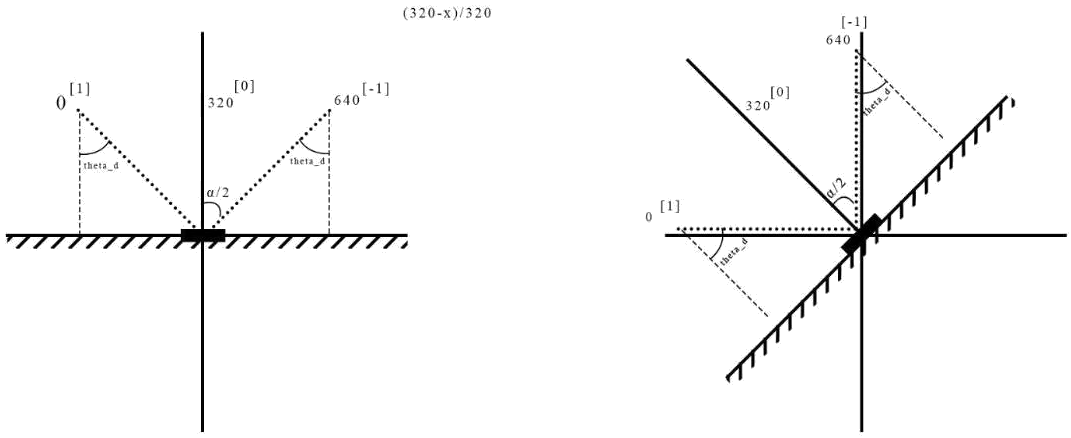
1. **Hardware:-** The hardware includes two cameras. One is a depth camera and the other is a pose camera. We obtain the depth data from the depth camera and the pose data from the pose camera to plot the 2D data. The depth camera also outputs a stream of colour JPEGs and depth PNGs sequentially.

2. **Software:-** The cameras are interfaced with the respective programs by using their API called the LibRealSenseSDK. We use the python wrapper for the API which is originally written in C++. Along with this, the Open3D and PyVista libraries provide the requisite functions for meshing point clouds and generating surfaces.

3.8 Past Work

3.8.1Static 2D Map Generation Algorithm

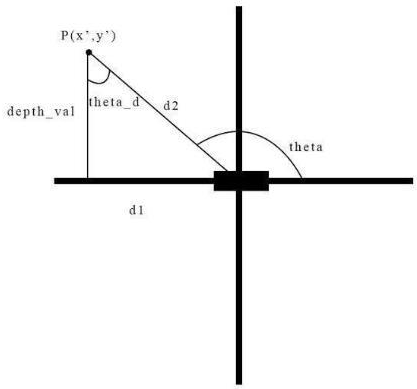
- Acquire the Depth and Colour Stream
- Input Angle of the camera
- Calculate the angle offset from positive X-axis
- Calculate angle made by pixel w.r.t. camera sensor
- Calculate the coordinates of the pixel location w.r.t. Fixed Reference
- Input deviations in x, y directions in meters
- Calculate final location w.r.t. Fixed Reference



Irrespective of the orientation of the camera, the angle made by the pixels (theta_d) will range from -α/2 to α/2 wrt camera normal using the formula

theta_d = ((hfov)/2)*((320-x)/320)

Figure 3.2 Pixel Angle Calculation



Data that we have:

- depth_val : Normal distance of the pixel location from sensor
- theta : Angle made by the pixel wrt +X axis
- theta_d : Angle made to the axis perpendicular to camera normal

Data to figure out:

- d1 : Cartesian distance of pixel from sensor
- d2 : Shortest distance of pixel location from sensor
- x',y' : Coordinates in the Global Cartesian System

Method: We know that,

- d1 = depth_val * tan(theta_d)
- d2 = (depth_val^2+d1^2)^(1/2)

- x' = d2*cos(theta)
- y' = d2*sin(theta)

Figure 3.3 Spherical Coordinate Calculation

3.8.2 Dynamic 2D Map Generation Algorithm

- Access the Tracking Camera to get the position and angular values
- Check if Thresholding criteria is met
- If criteria is met then plot the points and store the [position,points] in a dictionary
- If criteria is not met then skip current plotting and storing iteration
- Repeat until a satisfactory map is generated
- Store the final [position,points] dictionary in a file

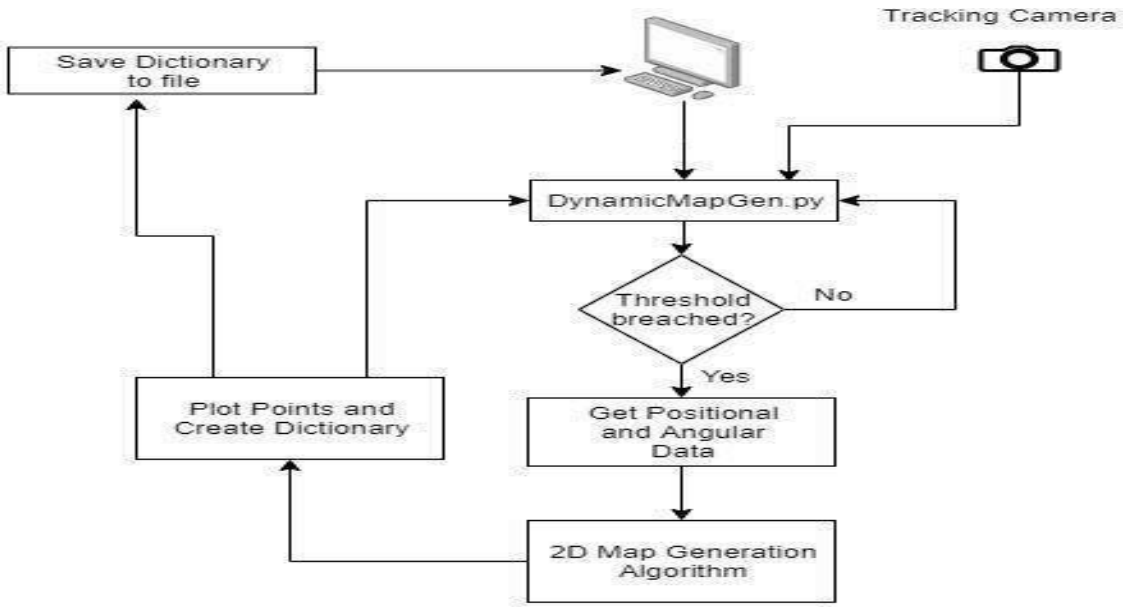


Figure 3.4 Dynamic 2D Map Flow Diagram

3.8.3 Static 3D Map Algorithm (Previous Approach)

- Make fragments: build local geometric surfaces (fragments) from short subsequences of the input RGB-D sequence. This uses RGBD odometry,Multiway registration,and RGB-D integration
- Register fragments: the fragments are aligned in a global space to detect loop closure.This part uses Global registration,ICP registration,and Multiway registration
- Refine registration: the rough alignments are aligned more tightly. This part uses ICP registration,and Multiway registration
- Integrate scene: integrate RGB-D images to generate a mesh model for the scene. This part uses RGB-D integration.

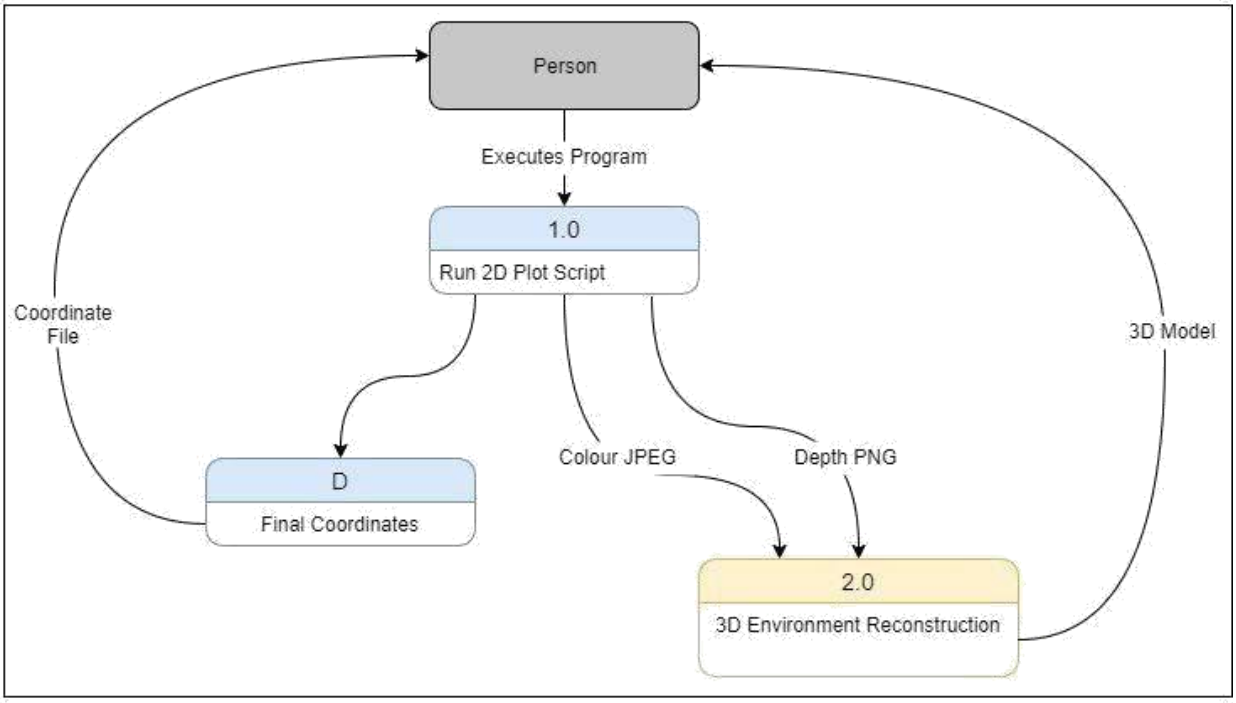


Figure 3.5 General Data Flow Diagram (Previous Approach

3.8.4 2D Data Flow Diagram (in reference to figure 3.6)

- The 2D plotting starts in **P1.0** which is the execution of the 2D plotting script
- The 2D plotting occurs dynamically and thus requires the current pose data of the camera relative to its starting position. This occurs in **P2.0** which interfaces the pose estimator camera methods and retrieves a tuple of current pose data i.e. (x,y,z,roll,pitch,yaw)
- Due to the refresh rate of both the depth and the pose estimator camera, we must only map data when a significant motion has taken place. This is done in **P3.0** where the current pose data is calculated with the previous pose data. If the difference between the x,y or yaw values is significant enough, we proceed with the plotting of the points. In **P3.1**,thresholding criteria is not met hence no updating takes place. In **P3.2** the pose data is updated if and only if the thresholding criteria is met, which in-turn acts as the previous pose data for the next iteration. This previous pose data is stored globally in a custom data structure, and can be referenced at any time
- After **P3.2**,**P4.0** occurs which accesses the depth camera methods and initializes and configures the depth and colour pipeline/stream. Both these streams run in parallel. For the 2D mapping, we are interested in acquiring the depth stream. For the 3D Reconstruction system, we require a series of Colour JPEG images and corresponding

Depth PNG images. **P5.0** and **P5.1** are responsible for converting the depth and colour frames received from **P4.1** and **P4.2** to the corresponding colour and depth images

- To create 2D maps, we first define the region between which depth data must be gathered. Theoretically, we can obtain the depth of every single pixel in the depth frame, however, most of these depth values are irrelevant. We only consider the central row of the depth frame as it most likely provides data about objects in the environment that the mobile robot is likely to encounter. We optimize even the central row value. In **P6.0**,we calculate the start and end of the row region based on the rotation of the camera. When the camera rotates, it most likely will rotate less than its horizontal FOV. Even though the thresholding criteria is met, overlapping points will still be plotted. Hence, we calculate the pixels that do not overlap with the previous iterations and map only those pixels of the central row
- Since we have all the data we need, **P6.1** converts all the points from the local coordinate system to the world/spherical coordinate system. The points generated in this iteration are stored temporarily in a custom data structure, which occurs in **P7.0**.The next iteration begins after this
- Once the mapping requirements are satisfied, the points from the temporary data structure are moved to a permanent data structure which can then be used for any further processing requirement

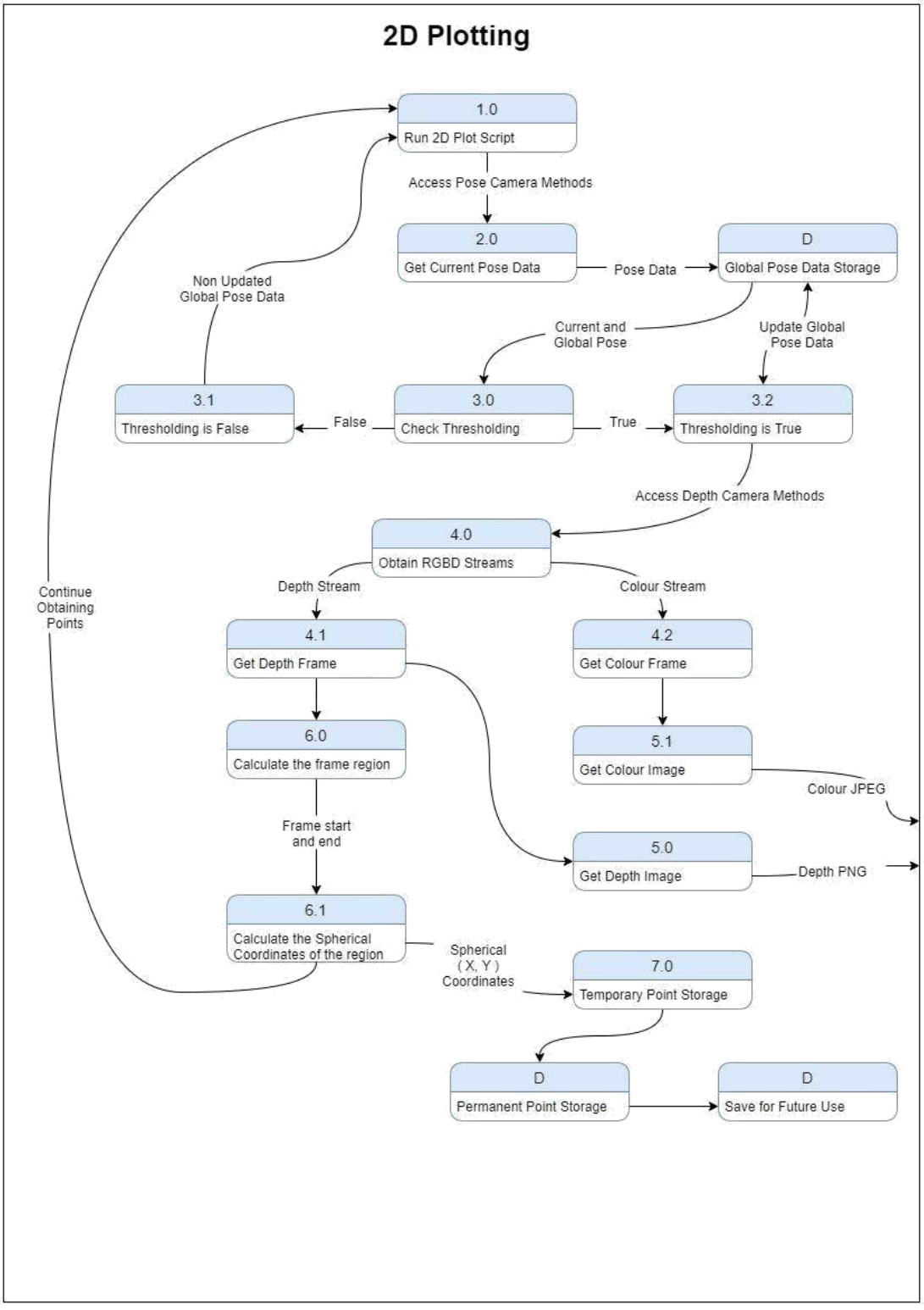


Figure 3.6 2D Data Flow Diagram

3.8.5 3D Data Flow Diagram (in reference to figure 3.7)

- For **P1.0** the 2D plotting algorithm provides a sequence of colour JPEGs and depth PNGs. RGB-D image pairs (Colour and depth images) are read from a source RGB-D image and registered to a target RGB-D image. For adjacent RGBD images, an identity matrix is used as initialization. For non-adjacent RGBD images, wide baseline matching is used as an initialization. This is called as frame matching and is **P2.0**
- A pose graph for multiway registration of all RGBD images is made. Each graph node represents an RGBD image and its pose which transforms the geometry to the global fragment space. For efficiency, only key frames are used
- Once a pose graph is created, multiway registration is performed and poses of RGB-D images are estimated. This leads to the formation of fragments. This occurs in **P3.0**
- Once the poses are estimated, RGB-D integration is used to reconstruct a colored fragment from each RGB-D sequence
- Downsampling of point clouds is done to make a point cloud sparser and regularly distributed. Normals and FPFH feature are precomputed
- We compute a rough alignment between two fragments. If the fragments are neighboring fragments, the rough alignment is determined by aggregating RGB-D odometry obtained from Make fragments.This leads to the registration of fragments
- In **P4.0,** we perform the refining of the fragments using optimization methods which might not always prove to be reliable
- Finally we read the alignment results from both Make fragments and Register fragments, then compute the pose of each RGB-D image in the global space. **P5.0** leads to the integration of RGB-D images using RGB-D integration.This leads to the formation of .ply files. The .ply files are viewable in any 3d rendering software

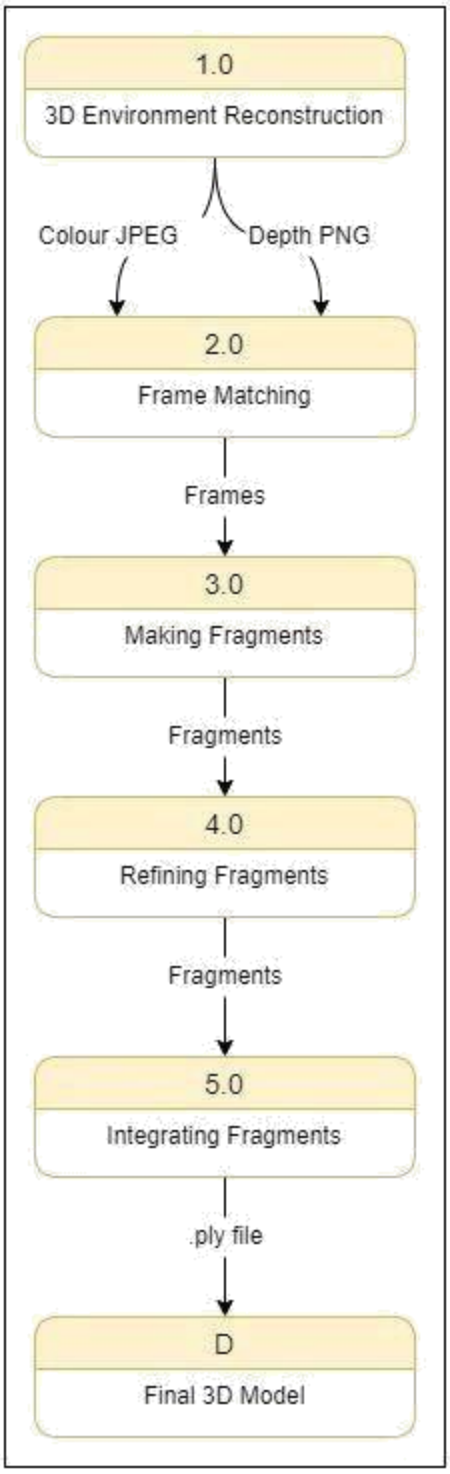


Figure 3.7 3D Data Flow Diagram

3.8.6 2D Static Map Results

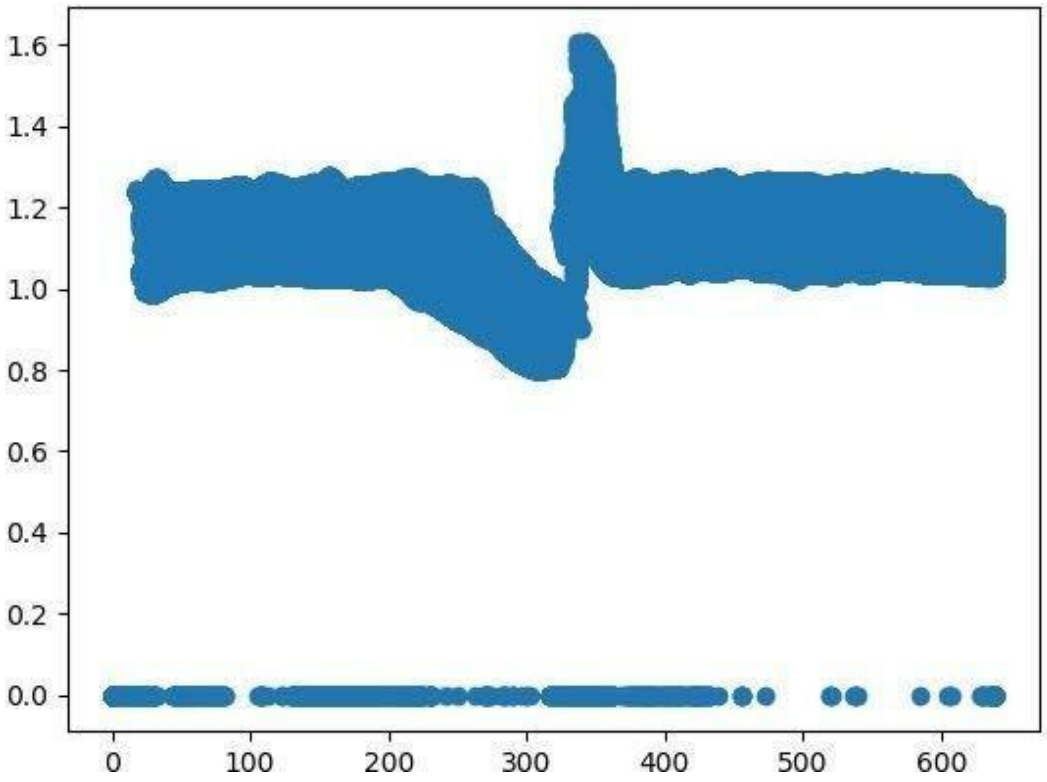


Figure 3.8 2D Static Map Results 1

The above output was the first test performed with the 2D algorithm. The mapping was done by taking a pixel coordinate (X,Y) and getting the respective depth values from each of them. Once we have a depth value associated with each pixel we create a map where the pixel X coordinate was the X coordinate of the map and the pixel depth values were the Y coordinate of the map. By doing this, we essentially take the cubic representation and slice a plane from it i.e. Top Plane. Here, the algorithm was run for a cupboard which was slightly open to stimulate variations. Because of the use of the camera at a resolution of 640x480, we have 640x480=307,200 points mapped.

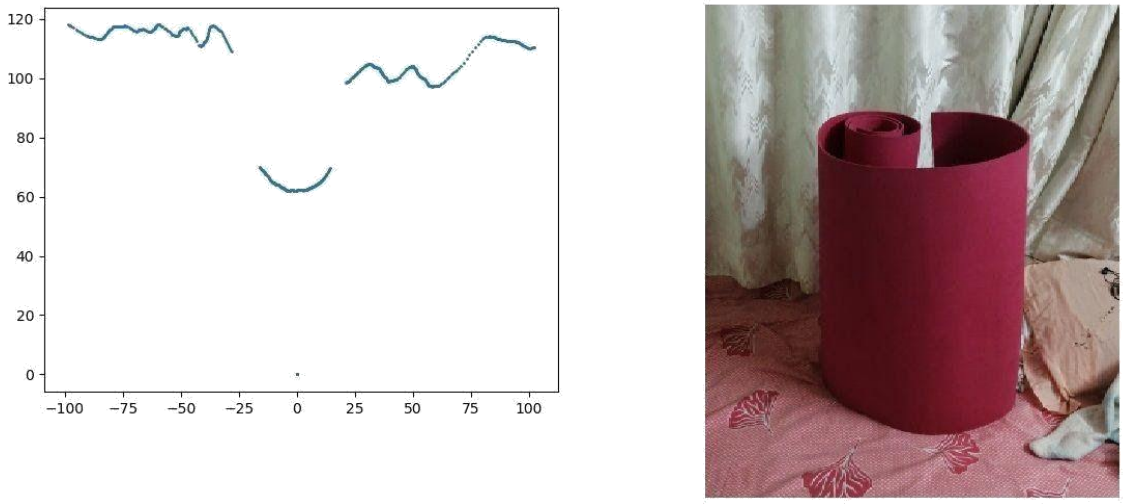


Figure 3.9 2D Static Map Results 2

Due to the need of mapping the current coordinate system to a world coordinate system, we devised a method to convert this spatial coordinate system to the world coordinate system. The previous result contained 307,200 points which may cause issues in the overall processing of the data. Hence, we consider only the central row in the depth measurement. Obstacles and/or objects which lie above or below the central row are less likely to cause any collision. These measurements of the extreme rows are hence irrelevant.

3.8.7 2D Dynamic map results

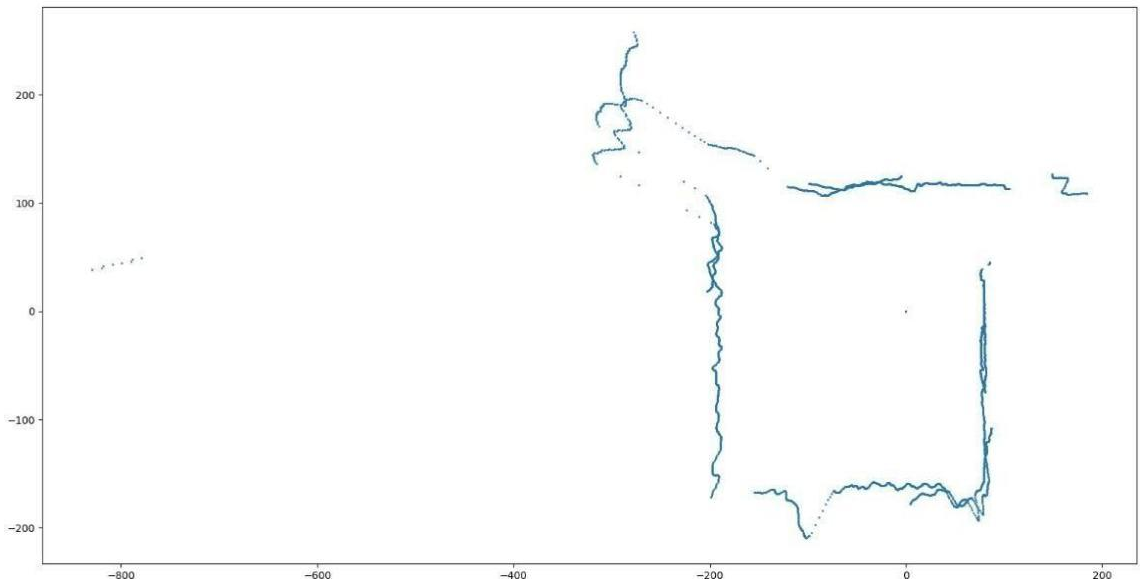


Figure 3.10 Dynamic Map Results 1

After obtaining a result for a single row value, we needed to add the condition that the mobile robot will be in motion and as such, the motion can be translator or rotatory. To account for this, when the user captures one row data during the depth acquisition, they also input the amount the move in the X and Y axis and also input the rotation performed. These parameters were then fit into the algorithm and a corrected output was observed. However, this method only integrates rotation and translation as inputted by the user and as such cannot be performed dynamically.

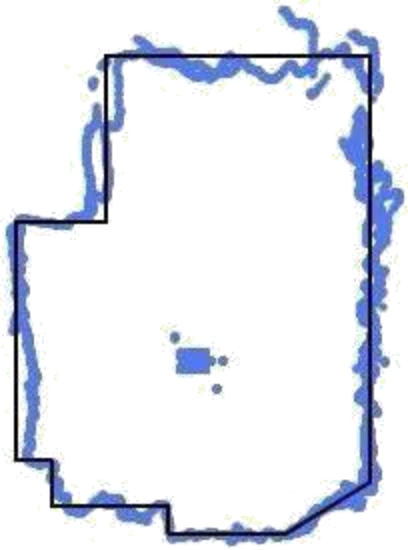


Figure 3.11 Dynamic Map Results 2

After integrating the rotation and translation compensation we needed to scale the system to accommodate for dynamic changes in the position, i.e. the system must know its rotation and translation without any human input. These parameters are obtained using a Pose Estimating Camera which proves values such as (x,y,z,roll,pitch,yaw). Using the x,y and yaw values, we can calculate the offset of the camera which then helps us dynamically plot the map.A thresholding function was also implemented which reduced the number of points plotted by only plotting when significant motion had taken place. The solid black lines indicate an approximation of the room this test was conducted in.

3.8.8 3D Manipulation Results

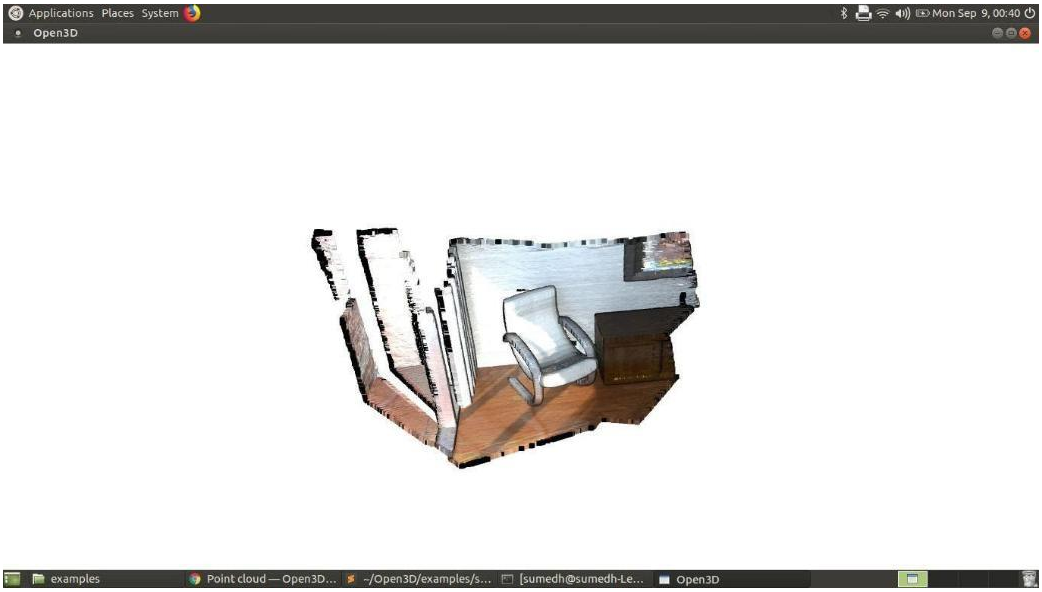


Figure 3.12 Point cloud

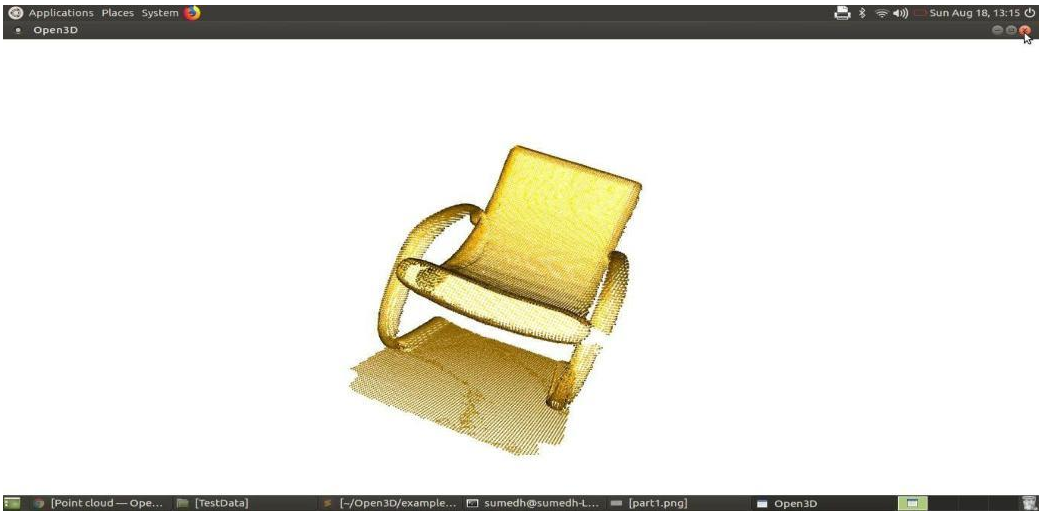


Figure 3.13 Isolated chair

3.8.9 Comparative study

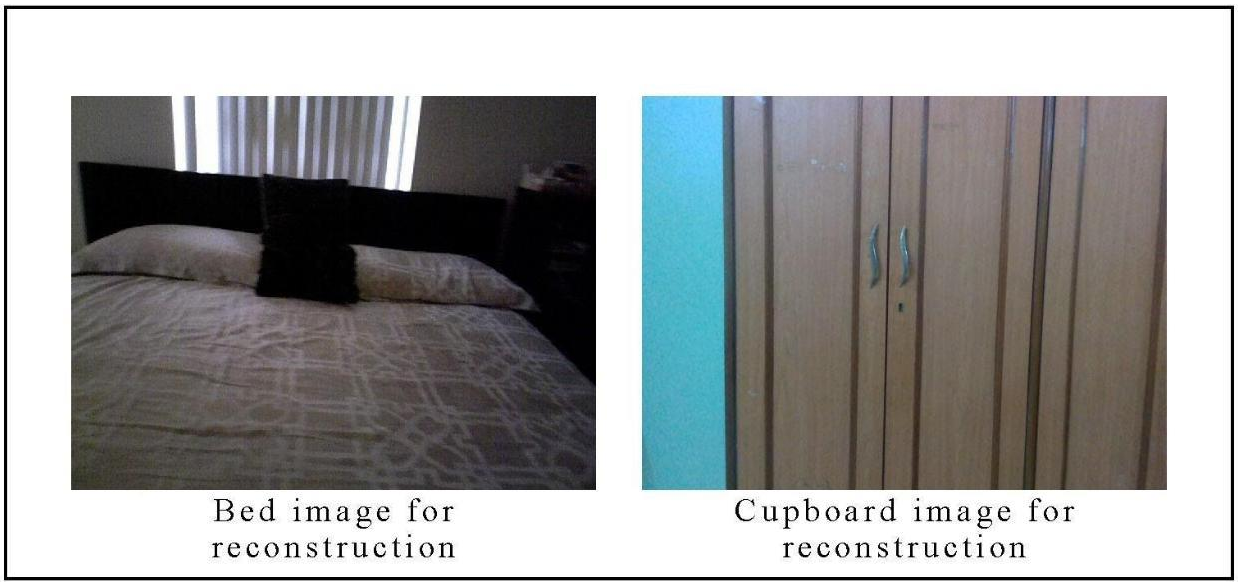


Figure 3.14 Images for reconstruction

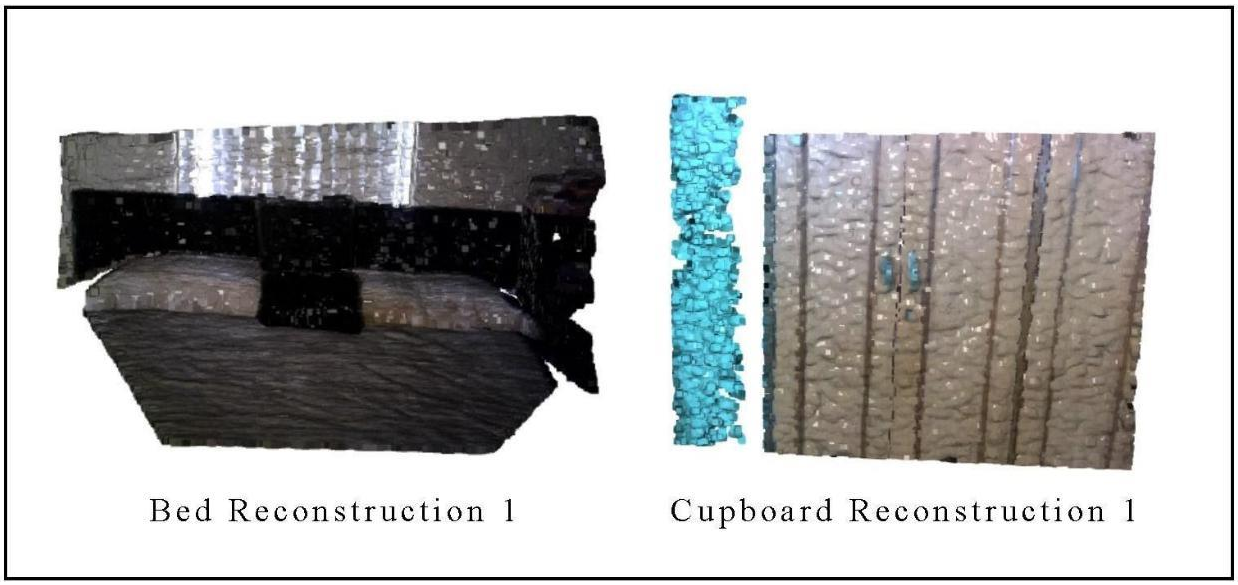


Figure 3.15 Reconstruction comparison 1

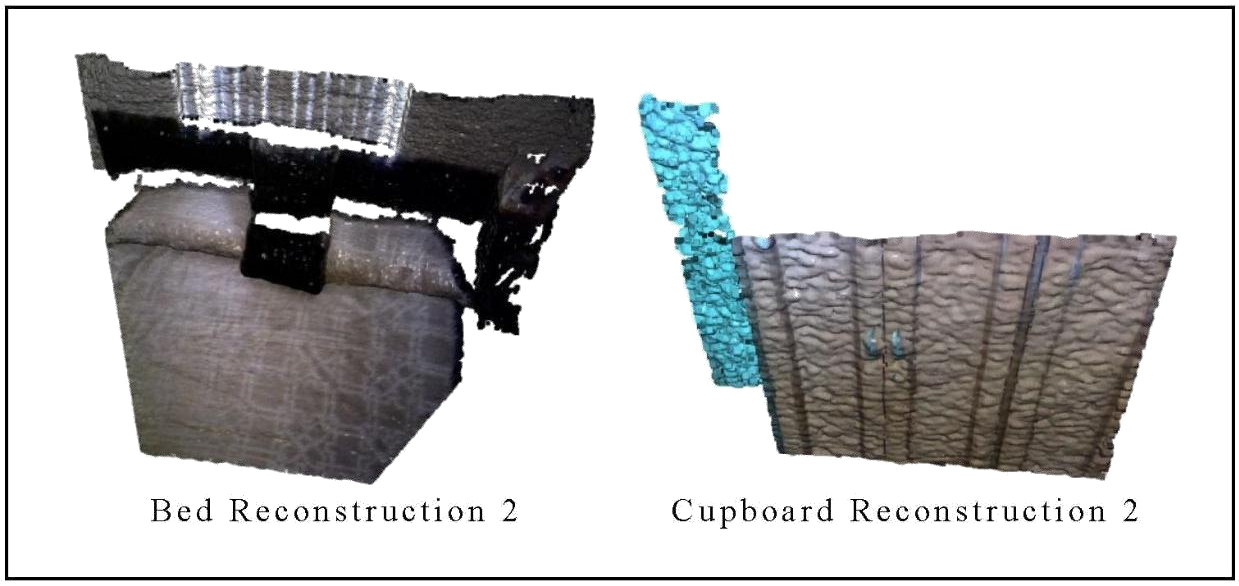


Figure 3.16 Reconstruction comparison 2

3.9 Updated System

The following section details the updated methodology for the 3D Reconstruction System only. The 2D Map Generation System detailed in section 3.8 remains unchanged.

3.9.1 Revision from previous system

As it can be seen from fig 3.15 and 3.16, the previous system relied entirely on image processing and hence would be a computationally expensive process in terms of both space and time to give an accurate result. As an example, the previous system required processing of roughly 1000-1500 depth and color images which would require special hardware. The newly proposed system does not use images but rather uses point clouds generated by a series of rather inexpensive mathematical calculations which makes the process of generating an accurate result far simpler. The output can also be verified in professional CAD software to measure the dimensional accuracy of the generated model which was not possible in the previous system due to the output being a simple visual reconstruction of the environment rather than a mathematically accurate model.

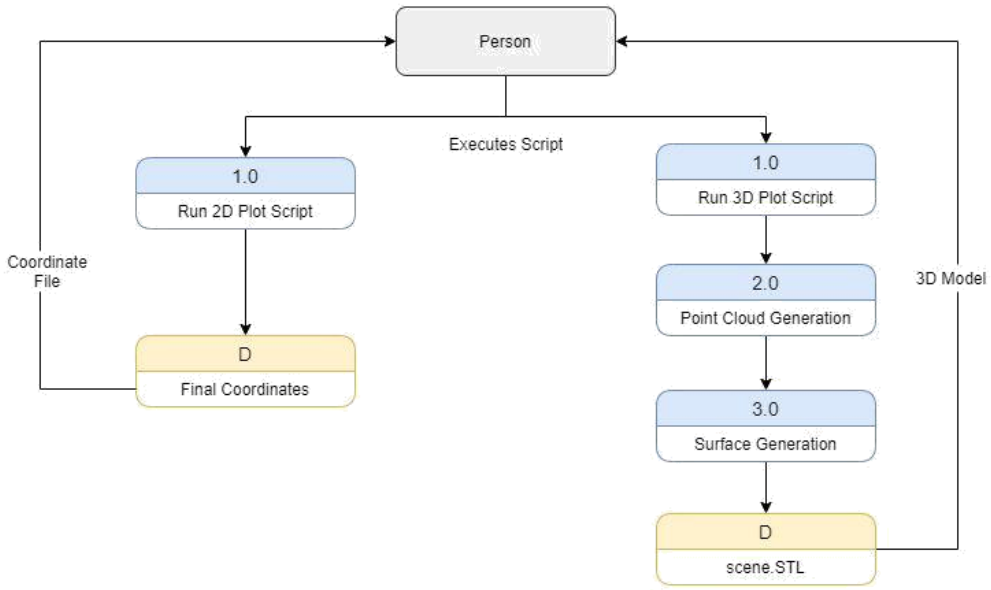


Fig 3.17 General DFD (New Approach)

3.9.2 Point Cloud Generation Algorithm

- Obtain yaw value from the pose camera and store it
- Retrieve the full pose data from the pose camera and pass it to the map generation function
- Get the entire depth frame
- Calculate the Region of Interest (ROI) for the frame using the current yaw and the stored yaw value
- Using the yaw value from the previous step we calculate the X,Y,Z coordinate of each pixel in the ROI
- Offset the X,Y,Z values with the current position of the camera
- After the requisite iterations, save the coordinates before moving to the next frame

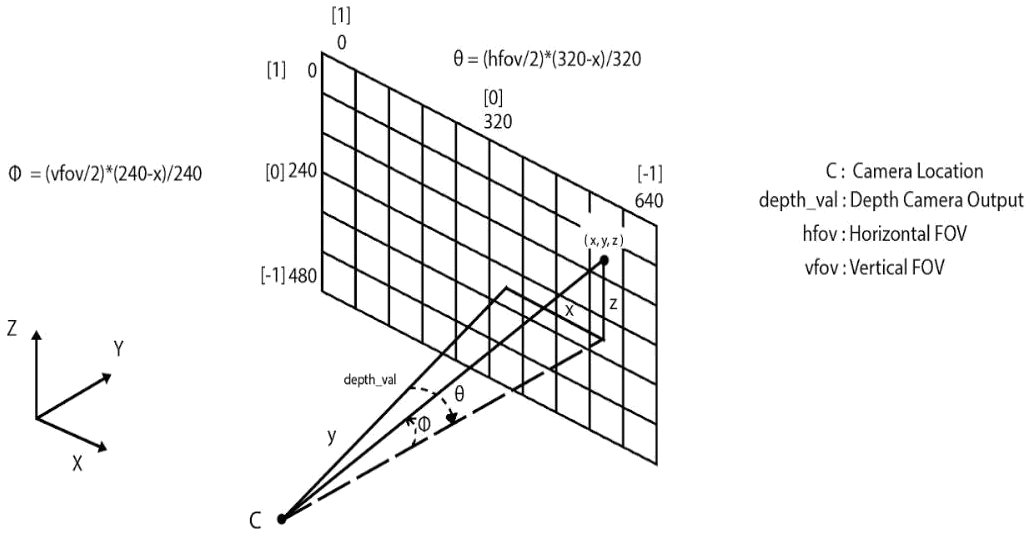


Fig 3.18 3D Spherical Coordinate Calculation

3.9.3 Surface Generation Algorithm

- Generate a point cloud from the xyz file obtained from 3.9.2 using Open3D
- Use the voxelization, downsampling operations from the Open3D library, preprocess the point cloud and estimate the normals
- Remove the camera origin positions from the point cloud data
- Apply Ruppert's algorithm to the preprocessed point cloud for the surface generation using triangulation
- Smoothen the surface
- After iterating over all the .xyz files, combine the surfaces to form a composite scene
- Save the scene to an STL file for further analysis

3.9.4 Point Cloud Generation

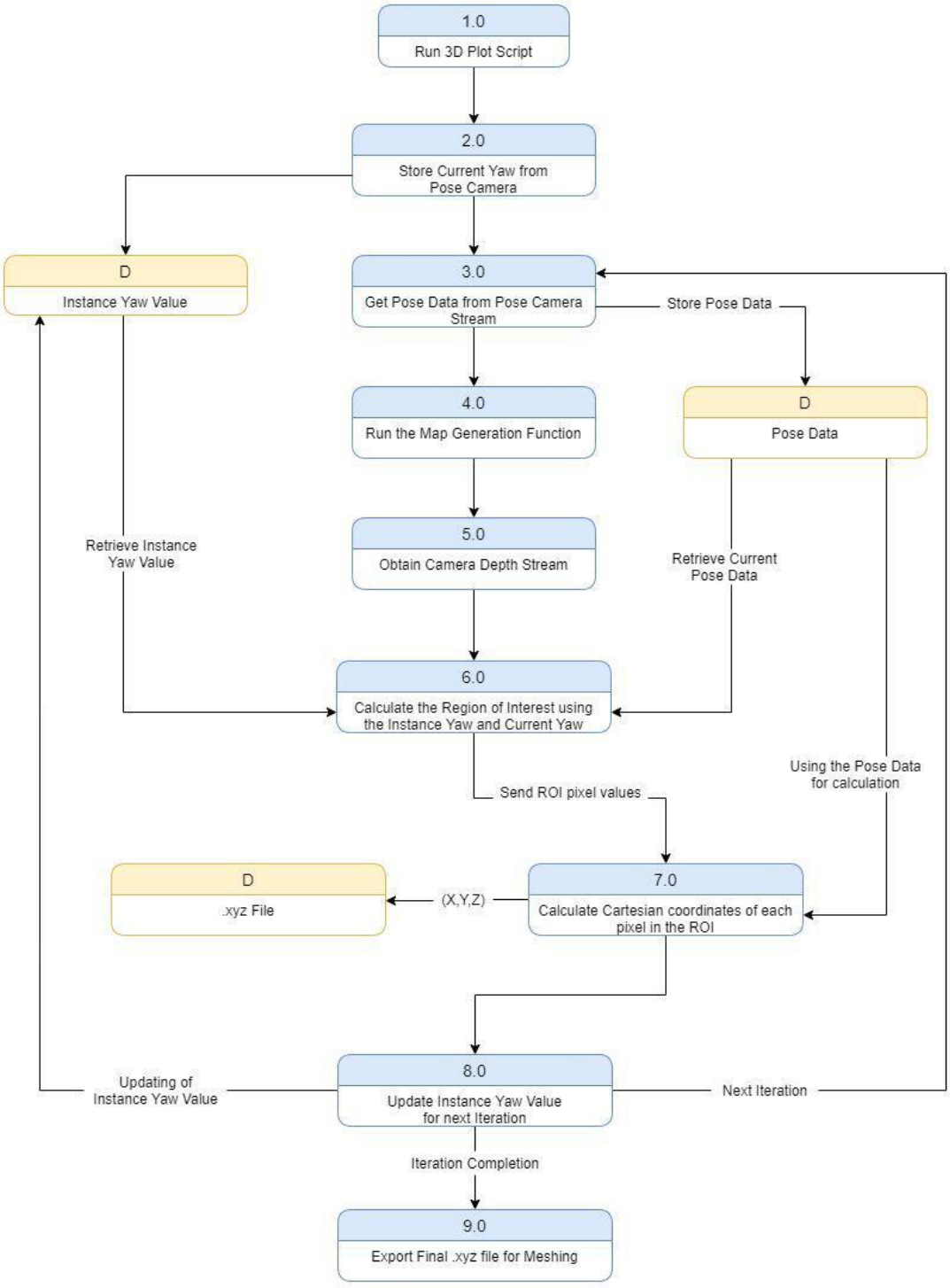


Fig 3.19Point Cloud Generation DFD

Explanation for fig 3.19

- The 3D point generation algorithm starts at **P1.0** where the script is executed
- We interface the pose camera which provides us the (x, y ,z) of the camera as well as its roll, pitch and yaw values. In **P2.0** we store the first yaw value in a data structure called Instance Yaw Value. This value will be referenced and updated in all successive iterations
- Once we have stored the yaw value, we pass the full pose data that will be retrieved in **P3.0** and store it in a data structure called Pose Data. This Pose Data is sent as an argument for the map generation function
- We run the map generation function in **P4.0**
- To calculate the coordinates of the points in the real world, we require the depth frame of the depth camera, i.e. the depth values of all the pixels. We use inbuilt functions to retrieve this depth frame in **P5.0**
- Because the amount of information has a new dimension as compared to the previous 2D implementation, we need a way to filter out the unnecessary data points. Thus we calculate the Region of Interest i.e. the start and end point for each iteration of the map generation. We use the Instance Yaw Value and Pose Data values to determine the ROI for the iteration. In case it is the first iteration, we consider the ROI to be the entire frame. Subsequent iterations will only consider a part of the whole frame. This calculation of the ROI is done in **P6.0**
- The start and end point of the frame is sent to **P7.0** where the real (x, y, z) values are calculated using Cartesian Methods. We converted the pixel coordinates to real coordinates. These values are temporarily stored in lists and once the iteration is complete, these values are appended sequentially to a permanent file with the .xyz file extension
- After a single frame is computed we can either move for the next iteration or end the iterations. We decide the next course of action in **P8.0**. If we want to conduct more iterations, the Instance Yaw Value is updated to the current yaw value i.e. the yaw value in Pose Data. The iteration again repeats from **P3.0** but with the newly updated Instance Yaw Value
- In case the iteration is ended, the .xyz file is exported in **P9.0** which can be further used for meshing algorithms

3.9.5 Surface Generation

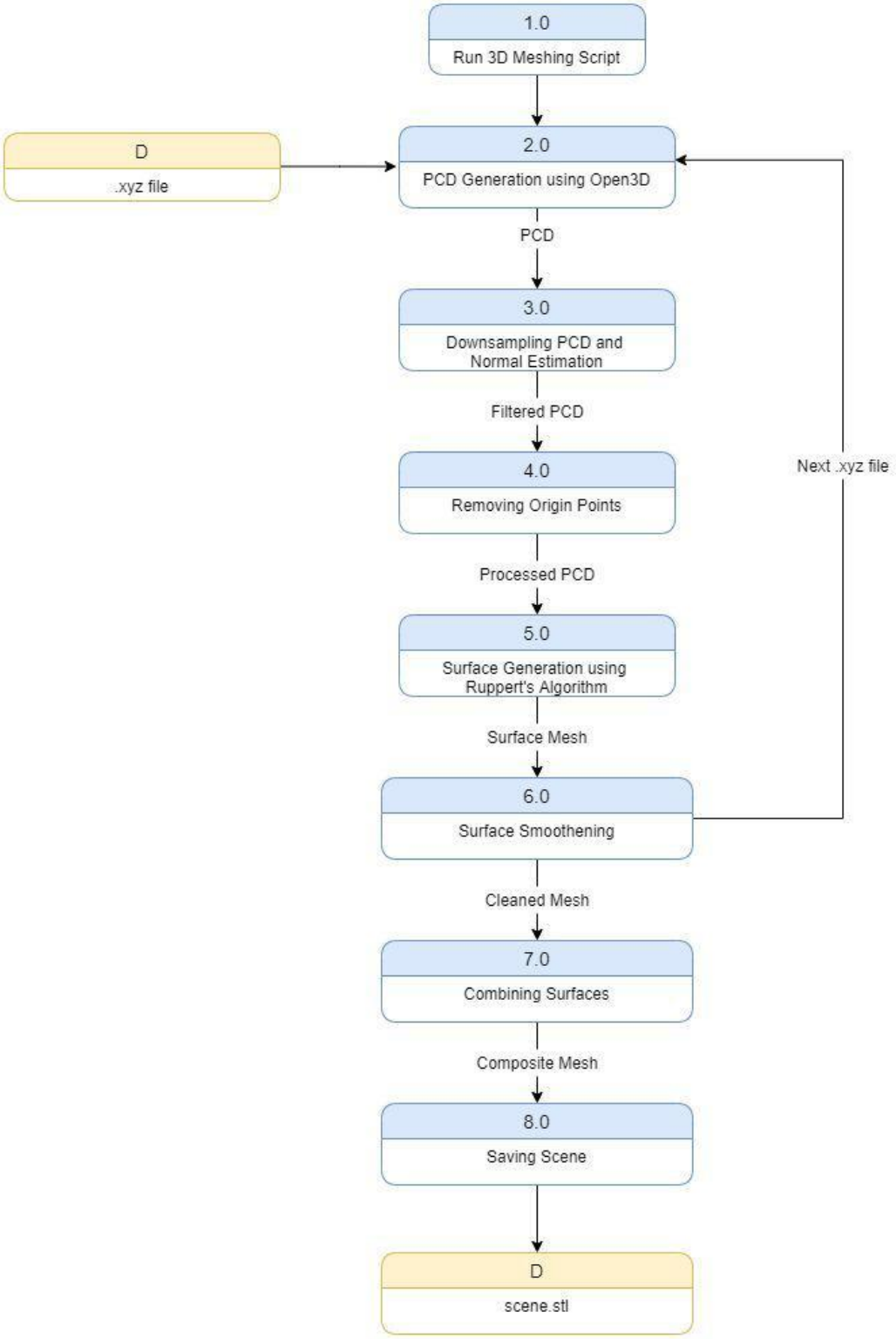


Fig 3.20 Surface Generation DFD

Explanation for Fig 3.20

- The 3D surface generation algorithm starts at **P1.0** where the script is executed
- The .xyz files generated in 3.9.5 are opened in Open3D which converts the list of points with (x,y,z) coordinates into a point cloud. Each file is converted into its own point cloud in **P2.0**which is then sent for preprocessing and surface generation
- Voxel downsampling uses a regular voxel grid to create a uniformly downsampled point cloud from an input point cloud from which the normals are estimated in **P3.0**.It is often used as a pre-processing step for many point cloud processing tasks. Downsampling helps reduce the number of points and hence reduces computational complexity. The algorithm operates in two steps:
    - Points are bucketed into voxels.
    - Each occupied voxel generates exactly one point by averaging all points inside
- All origin combinations aka the camera position combinations eg-(0,0,0) are removed in **P4.0** to further simplify the process of triangulation or mesh generation. Points are converted into PyVista's polydata type and then into a numpy array from which the combinations are removed using linear algebra and then the simplified pointcloud is converted back into the polydata format
- **P5.0** is the main step of the entire process in which Ruppert-Delaunay Triangulation is applied to the preprocessed pointcloud which gives the surface/mesh of that corresponding pointcloud
- **Delaunay Triangulation** - In mathematics and computational geometry, a Delaunay triangulation (also known as a Delone triangulation) for a given set P of discrete points in a plane is a triangulation DT(P) such that no point in P is inside the circumcircle of any triangle in DT(P). Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation; they tend to avoid sliver triangles. The triangulation is named after Boris Delaunay for his work on this topic from 1934
- For a set of points on the same line there is no Delaunay triangulation (the notion of triangulation is degenerate for this case). For four or more points on the same circle (e.g., the vertices of a rectangle) the Delaunay triangulation is not unique: each of the two possible triangulations that split the quadrangle into two triangles satisfies the "Delaunay

condition", i.e., the requirement that the circumcircles of all triangles have empty interiors

- By considering circumscribed spheres, the notion of Delaunay triangulation extends to three and higher dimensions. Generalizations are possible to metrics other than Euclidean distance. However, in these cases a Delaunay triangulation is not guaranteed to exist or be unique
- **Ruppert's Algorithm** - In mesh generation, Ruppert's algorithm, also known as Delaunay refinement, is an algorithm for creating quality Delaunay triangulations. The algorithm takes a planar straight-line graph (or in dimension higher than two a piecewise linear system) and returns a conforming Delaunay triangulation of only quality triangles. A triangle is considered poor-quality if it has a circumradius to shortest edge ratio larger than some prescribed threshold
- The algorithm begins with a Delaunay triangulation of the input vertices and then consists of two main operations.
    - The midpoint of a segment with non-empty diametral circles is inserted into the triangulation.
    - The circumcenter of a poor-quality triangle is inserted into the triangulation, unless this circumcenter lies in the diametral circle of some segment. In this case, the encroached segment is split instead.
- These operations are repeated until no poor-quality triangles exist and all segments are not encroached
- After the surface is generated by applying triangulation on the pointcloud, the surface is smoothened to prevent any jaggedness, rough edges or astray triangles in **P6.0**
- Once all the surfaces are generated, they are simply added up with each other using boolean operations from the PyVista library in **P7.0**.This added mesh or map is then displayed with the appropriate lighting parameters
- Once a proper scene or map has been generated, it is exported into an STL file for further use in **P8.0**such as taking measurements or post processing

Chapter 4 : Updated System Results

4.1 Reconstruction results

Various reconstruction algorithm results have been detailed below for the following reference image

Fig 4.1 Static Reconstruction Reference Image

1. Ball Pivoting - The Ball-Pivoting Algorithm (BPA) computes a triangle mesh interpolating a given point cloud. Typically, the points are surface samples acquired with multiple range scans of an object. The principle of the BPA is very simple: Three points form a triangle if a ball of a user-specified radius p touches them without containing any other point. Starting with a seed triangle, the ball pivots around an edge (i.e., it revolves around the edge while keeping in contact with the edge's endpoints) until it touches another point, forming another triangle. The process continues until all reachable edges have been tried, and then starts from another seed triangle, until all points have been considered.

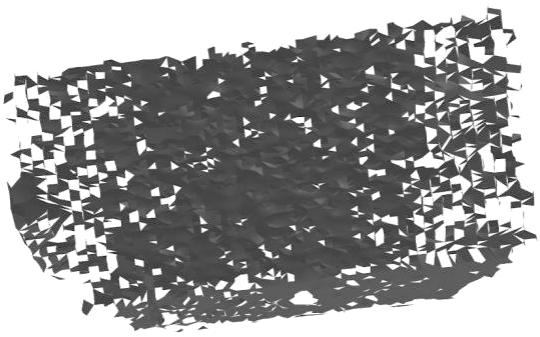


Fig 4.2 Ball Pivoting

2. Tetrahedral - The algorithm requires the minimal surface information. The solid modelling of the 3D objects needs not to be given in full explicit connectivity definitions of the surface triangles. The generated tetrahedrons have empty circum spheres which are the indication of the Delaunay property. A new automatic node replacement scheme reflecting the initial surface nodal spacings is developed. The successive refinement scheme results in such a point distribution that the algorithm does not require any surface conforming checks to avoid penetrated surface boundaries and overlapped tetrahedrons. The surface triangles become a direct consequence of interior tetrahedralization

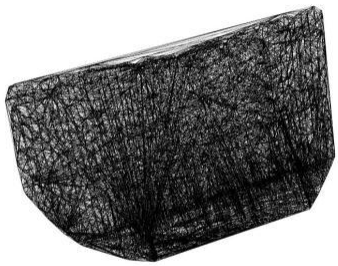


Fig 4.3 Tetrahedral Mesh

3. Alpha (alpha=0.1358) - Alpha shapes are closely related to alpha complexes, subcomplexes of the Delaunay triangulation of the point set. Each edge or triangle of the Delaunay triangulation may be associated with a characteristic radius, the radius of the smallest empty circle containing the edge or triangle. For each real number α, the α-complex of the given set of points is the simplicial complex formed by the set of edges and triangles whose radii are at most 1/α. The union of the edges and triangles in the α-complex forms a shape closely resembling the α-shape; however it differs in that it has polygonal edges rather than edges formed from arcs of circles. More specifically, Edelsbrunner (1995) showed that the two shapes are homotopy equivalent. (In this later work, Edelsbrunner used the name "α-shape" to refer to the union of the cells in the α-complex, and instead called the related curvilinear shape an α-body.)

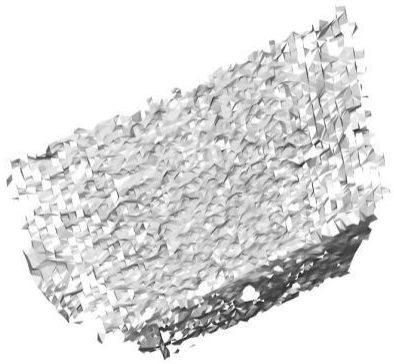


Fig 4.4 Alpha Mesh

4. Delaunay Triangulation with Rupperts - In mathematics and computational geometry, a Delaunay triangulation (also known as a Delone triangulation) for a given set P of discrete points in a plane is a triangulation DT(P) such that no point in P is inside the circumcircle of any triangle in DT(P). Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation; they tend to avoid sliver triangles. The triangulation is named after Boris Delaunay for his work on this topic from 1934.

For a set of points on the same line there is no Delaunay triangulation (the notion of triangulation is degenerate for this case). For four or more points on the same circle (e.g., the vertices of a rectangle) the Delaunay triangulation is not unique: each of the two possible triangulations that split the quadrangle into two triangles satisfies the "Delaunay condition", i.e., the requirement that the circumcircles of all triangles have empty interiors.

By considering circumscribed spheres, the notion of Delaunay triangulation extends to three and higher dimensions. Generalizations are possible to metrics other than Euclidean distance. However, in these cases a Delaunay triangulation is not guaranteed to exist or be unique.

Ruppert's Algorithm - In mesh generation, Ruppert's algorithm, also known as Delaunay refinement, is an algorithm for creating quality Delaunay triangulations. The algorithm takes a planar straight-line graph (or in dimension higher than two a piecewise linear system) and returns a conforming Delaunay triangulation of only quality triangles. A triangle is considered poor-quality if it has a circumradius to shortest edge ratio larger than some prescribed threshold.

The algorithm begins with a Delaunay triangulation of the input vertices and then consists of two main operations.

- The midpoint of a segment with non-empty diametral circles is inserted into the triangulation.
- The circumcenter of a poor-quality triangle is inserted into the triangulation, unless this circumcenter lies in the diametral circle of some segment. In this case, the encroached segment is split instead.

These operations are repeated until no poor-quality triangles exist and all segments are not encroached

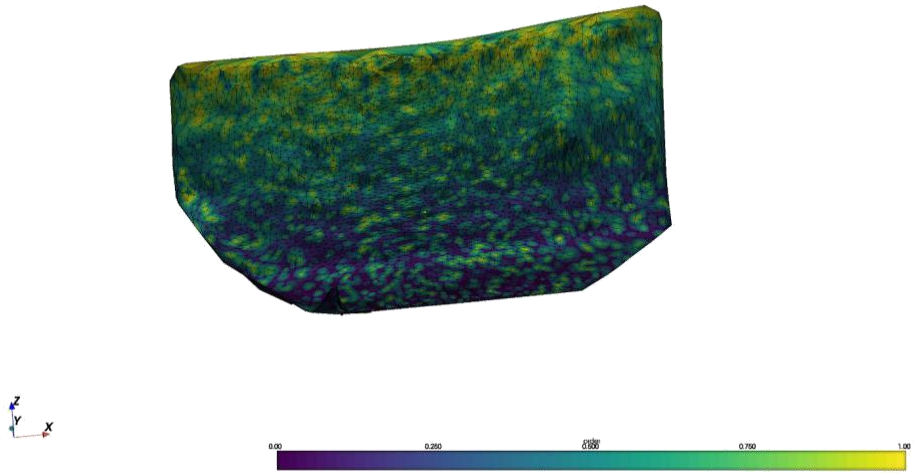


Fig 4.5 Delaunay Triangulation

5. Poisson Reconstruction - The Poisson formulation considers all the points at once, without resorting to heuristic spatial partitioning or blending, and is therefore highly resilient to data noise. Unlike radial basis function schemes, our Poisson approach allows a hierarchy of locally supported basis functions, and therefore the solution reduces to a well conditioned sparse linear system. We describe a spatially adaptive multiscale algorithm whose time and space complexities are proportional to the size of the reconstructed model

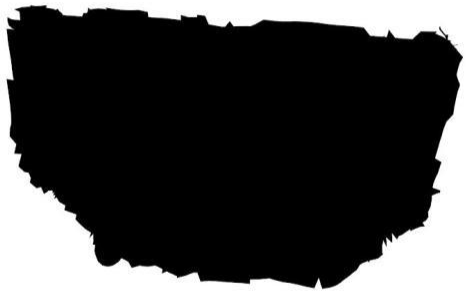


Fig 4.6 Screened Poisson

6. Marching Cubes - The algorithm proceeds through the scalar field, taking eight neighbor locations at a time (thus forming an imaginary cube), then determining the polygon(s) needed to represent the part of the isosurface that passes through this cube. The individual polygons are then fused into the desired surface. This is done by creating an index to a precalculated array of 256 possible polygon configurations (28=256) within the cube, by treating each of the 8 scalar values as a bit in an 8-bit integer. If the scalar's value is higher than the iso-value (i.e., it is inside the surface) then the appropriate bit is set to one, while if it is lower (outside), it is set to zero. The final value, after all eight scalars are checked, is the actual index to the polygon indices array. Finally each vertex of the generated polygons is placed on the appropriate position along the cube's edge by linearly interpolating the two scalar values that are connected by that edge. The gradient of the scalar field at each grid point is also the normal vector of a hypothetical isosurface passing from that point. Therefore, these normals may be interpolated along the edges of each cube to find the normals of the generated vertices which are essential for shading the resulting mesh with some illumination model.

a. Using PyMCubes

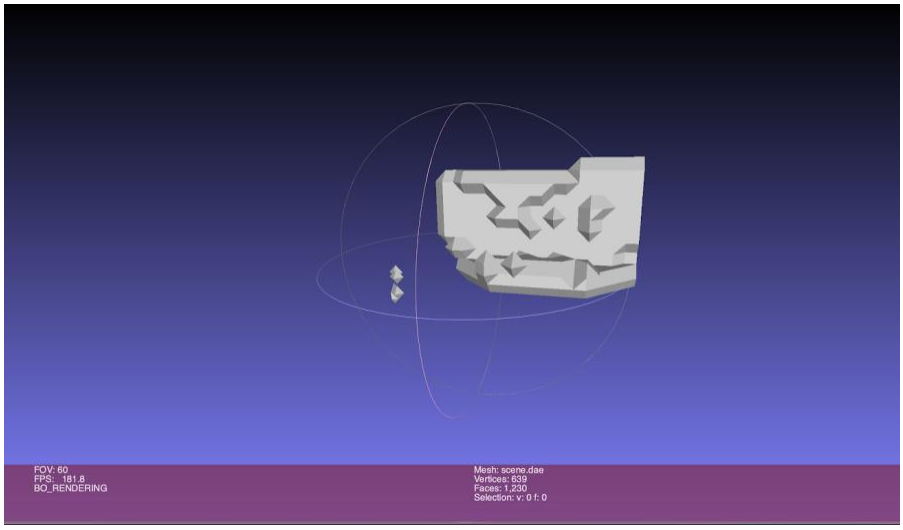


Fig 4.7 Marching Cubes (PyMCubes)

b. Using Skimage

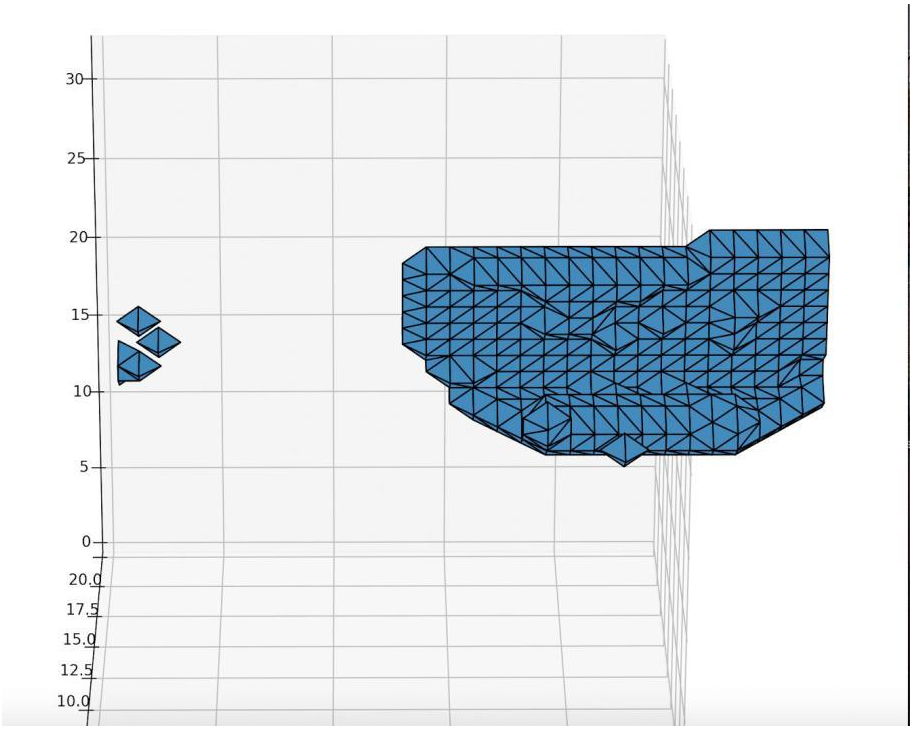


Fig 4.8 Marching Cubes (Skimage)

4.2 Algorithm Comparison

Time Graph

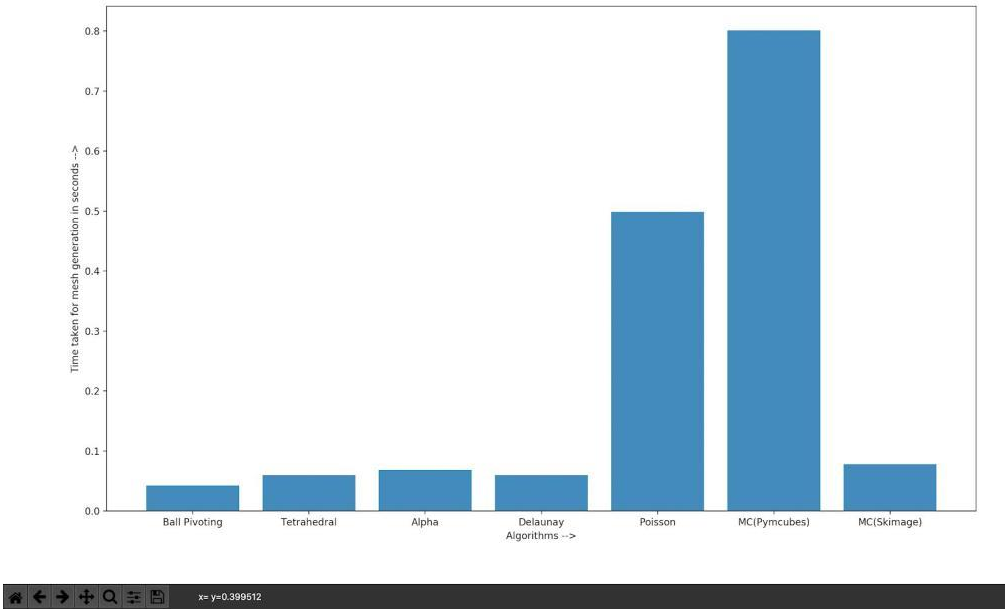


Fig 4.9 Time taken for Various Algorithms

Fig 4.9 highlights the comparison between different algorithms mentioned in detail and the time taken for mesh generation by each of them in seconds. As per the outputs in section 4.1 we can visually interpret that the fig 4.5 (our algorithm) is in close resemblance with the original image in fig 4.1. When we compare the visual accuracy with respect to the time taken to generate the output by different algorithms, we can infer that the most optimum result is obtained by Delaunay with Ruppert's algorithm with respect to time and accuracy.

4.3 Mapping Results

Continuous (Continuously generated scene files)

The continuous system consists of getting the depth frame with a single iteration of the code. We calculate the Region Of Interest (ROI) for each frame and consider only the relevant part.

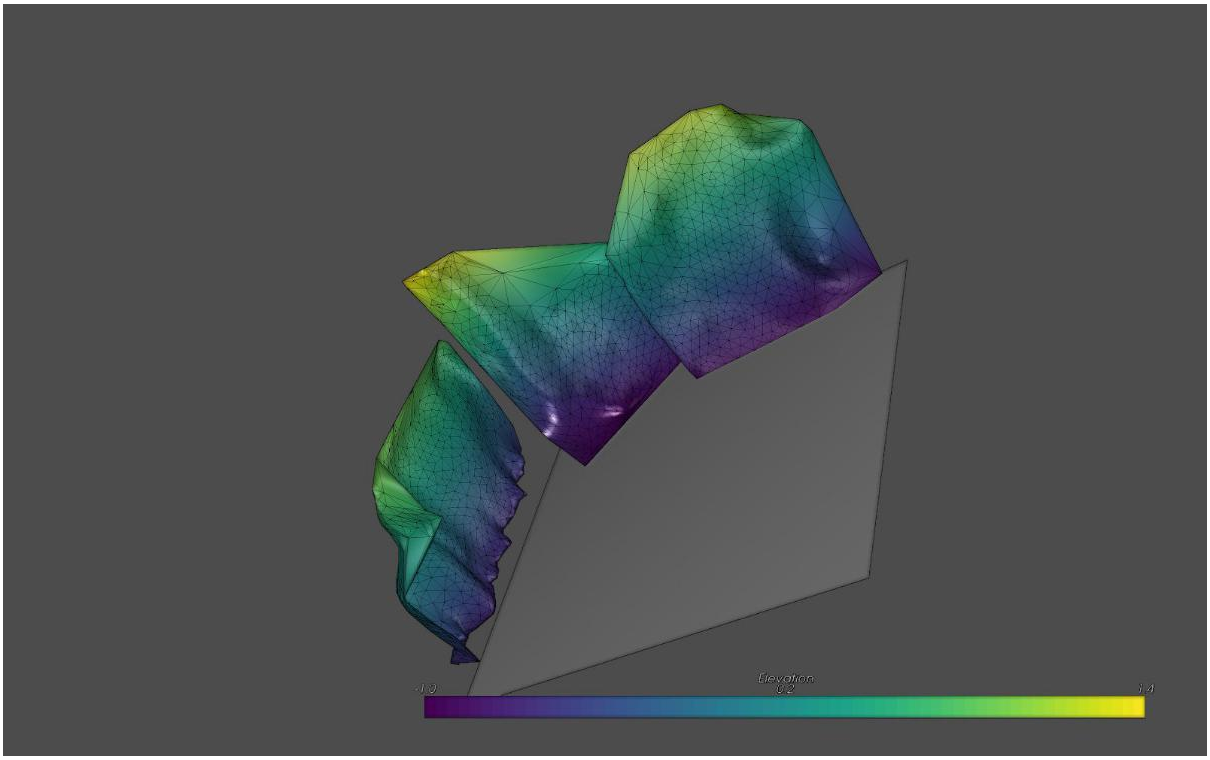


Fig 4.10 Continuous Map

Discrete (Separate wall and corner files)

The difference between continuous and the discrete iterations is that in discrete, the full frame is considered and each new frame is stored in a separate xyz file. Hence there are overlapping surfaces in this section.

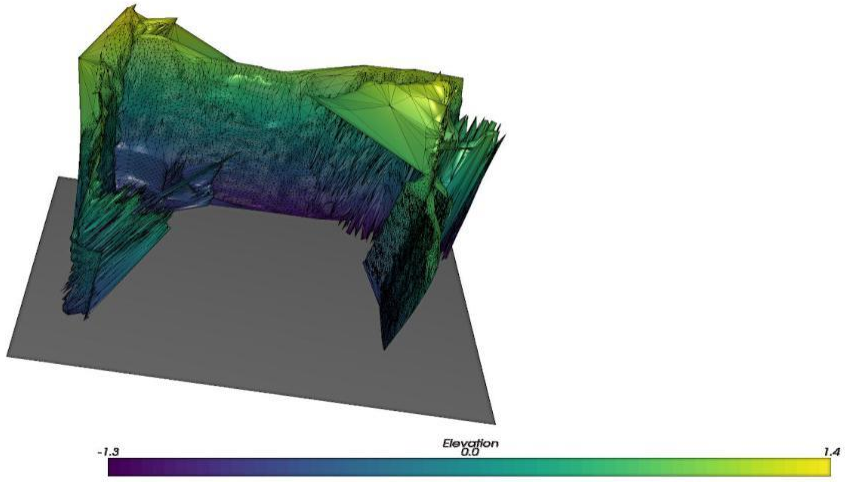


Fig 4.11 Discrete Map

4.4 Static 3D Mathematical Results

4.4.1 Set I

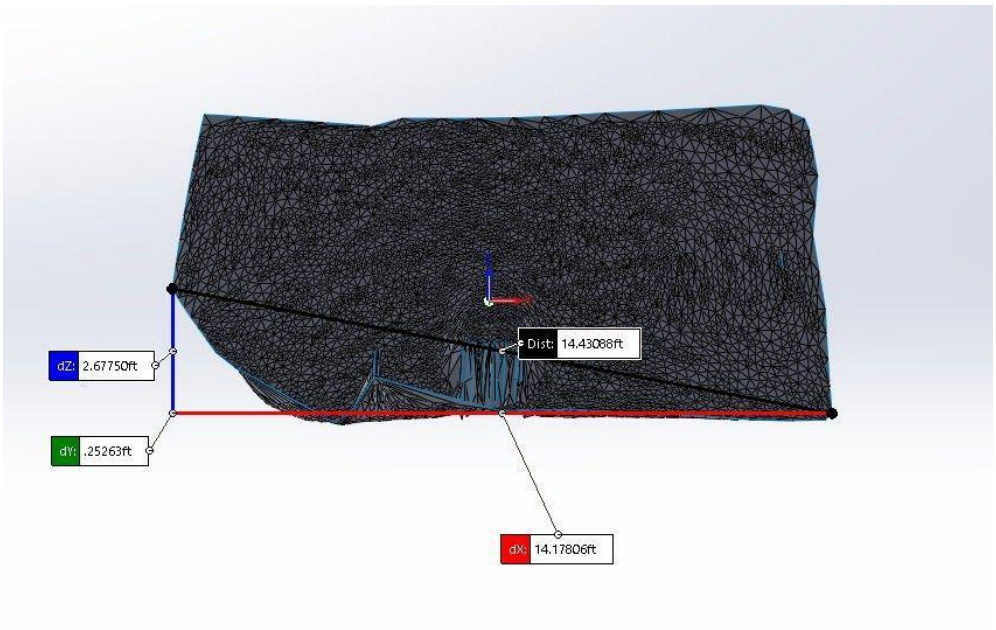


Fig 4.12 Length of wall with chair (Red Line)

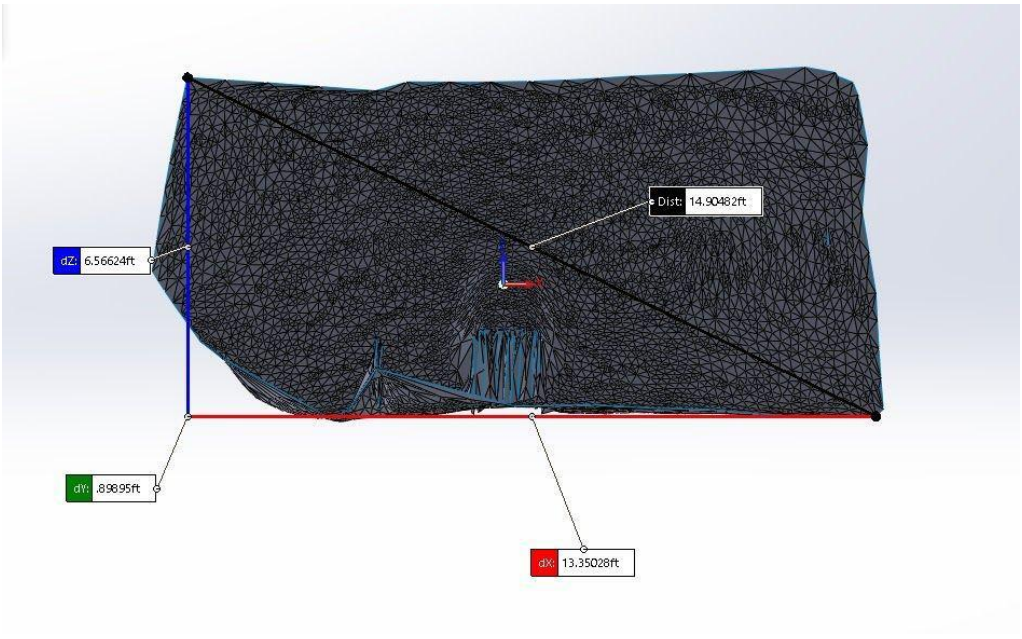


Fig 4.13 Height of wall with chair (Blue Line)

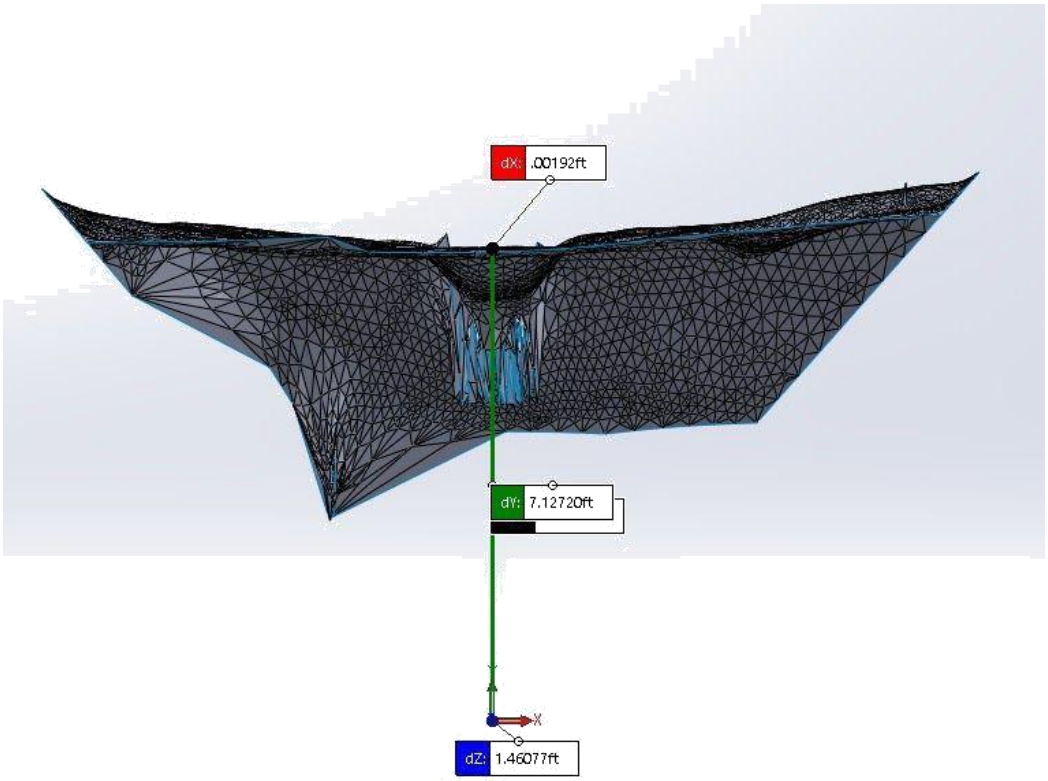


Fig 4.14 Distance of the wall from the camera (Green Line)

4.4.2 Set II

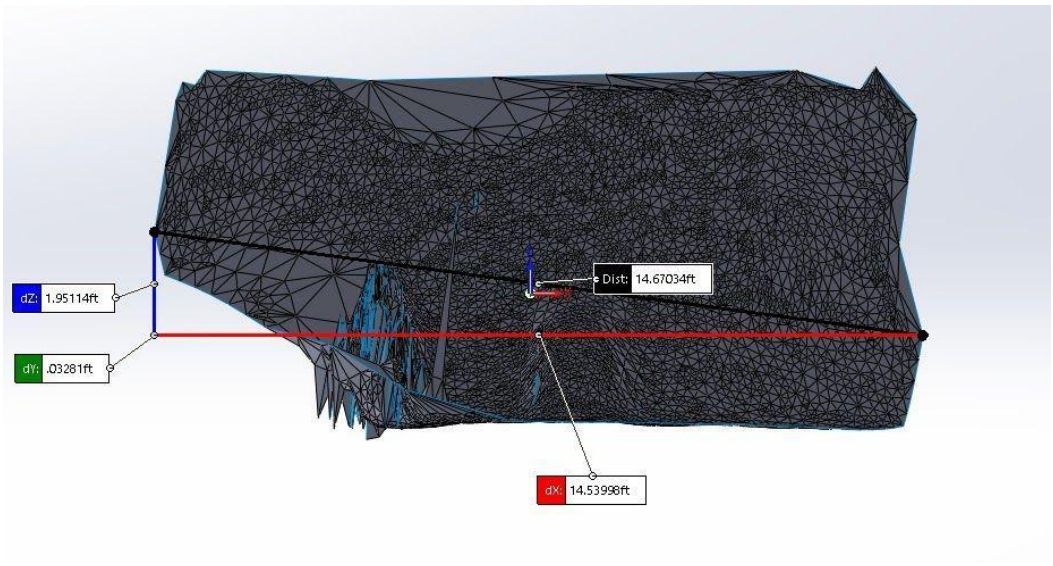


Fig 4.15 Length of wall with suitcase (Red Line)

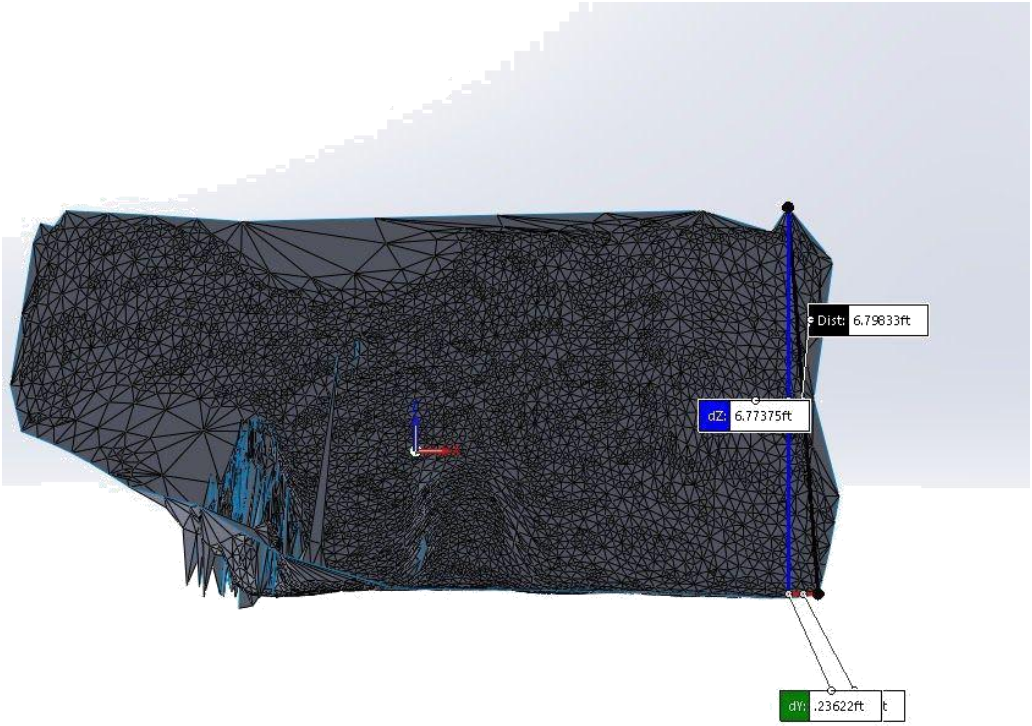


Fig 4.16 Height of wall with suitcase (Blue Line)

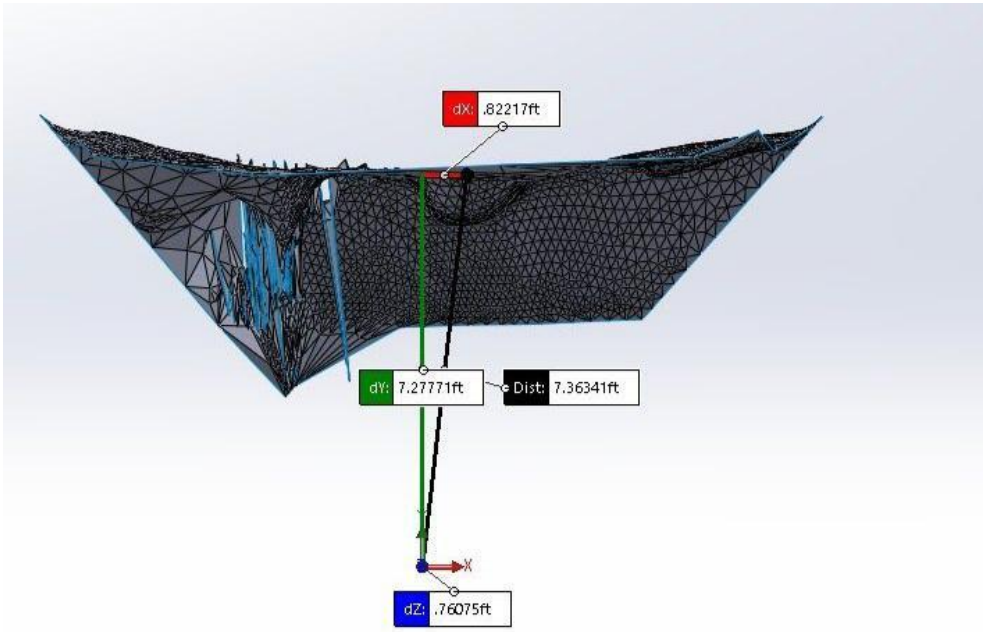


Fig 4.17 Distance of wall from camera (Green Line)

4.4.2 Set III

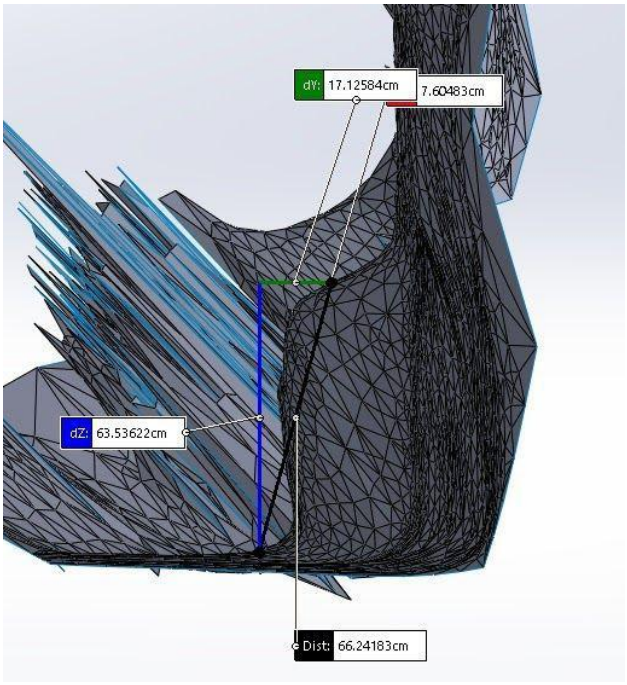


Fig 4.18 Suitcase Height (Blue Line)

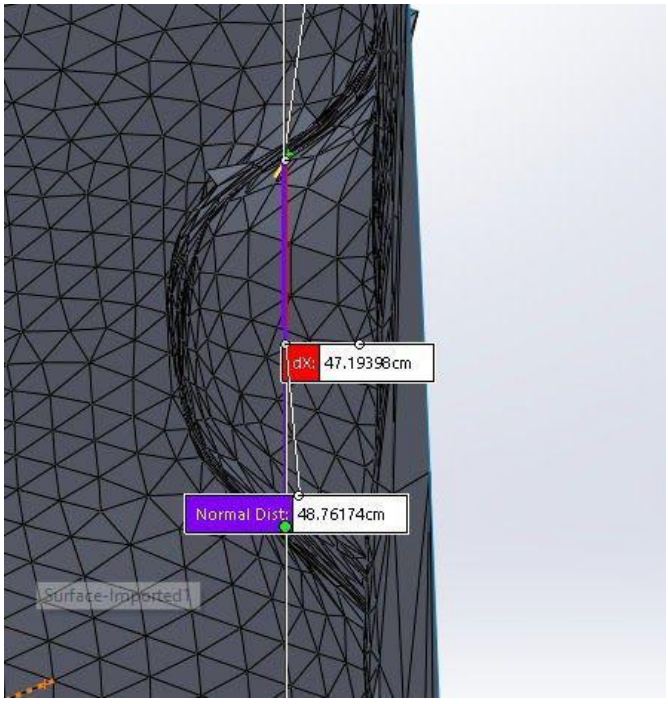


Fig 4.19 Suitcase Width (Red Line)

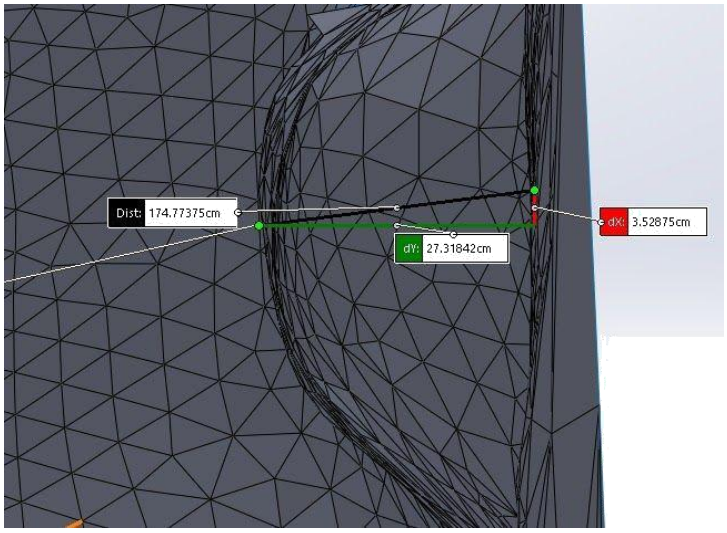


Fig 4.20 Suitcase Length (Green Line)

4.5 Dimensional Error Calculation

4.5.1 Wall Error Calculation

No.	**Measurement 1 in ft. (Chair)**	**Measurement 2 in ft. (Suitcase)**	**Avg**	**Real Approx.**	**Error %**
Length	14.18	14.54	14.36	12.02	19.47
Height	6.57	6.77	6.67	7.17	6.97
Distance from Camera	7.13	7.28	7.21	7.53	4.25

Table 4.1 Error Calculation of Dimensions of the wall

4.5.2 Suitcase Calculation

No.	Measurement in cm. (Suitcase)	Real Approx.	Error %
Length	27.3	26.4	3.4
Height	63.5	66.6	4.65
Width	47.2	41.2	14.56

Table 4.2 Error Calculation of the Suitcase

Explanation for Tables 4.1 and 4.2

- Fig 4.12, 4.13 and 4.14 depict a mesh where a chair is placed in front of a wall. In fig 4.12, the red line indicates the length of the wall. In fig 4.13, the blue line indicates the height of the wall and in fig 4.14, the green line indicates the distance of the wall from the camera location. Fig 4.15, 4.16 and 4.17 shows the mesh with a suitcase in front of the same wall. These 6 figures (4.12 to 4.17) are controlled outputs that are used to measure the accuracy of the meshing algorithm. Fig 4.18, 4.19 and 4.20 depict just the suitcase from the previous figures. We measure the accuracy of the dimensions of the wall and the suitcase created by the meshing algorithm wrt the real values
- In table 4.1 we consider the dimensions of the wall, i.e. its length, height and distance from camera. We take the dimensions highlighted in fig (4.12-4.17) and consider the average of these values. We then measure the percentage of error using the formula : |(real-measured)*100/real|. The real value was measured using a standard measuring tape. While considering the length of the wall, we determined the measured value to be 14.18ft from fig 4.12 and 14.54ft from fig 4.15. Averaging these values we get 14.36ft. The real value was determined to be approximately 12.02ft. The error calculation becomes |-2.34*100/12.02| = ~19.47% error. The same calculation is performed for the height of the wall and the distance from the camera
- In table 4.2, we measure the dimensions of the suitcase i.e. its length, height and width. Fig 4.18, 4.19 and 4.20 highlight the dimensions taken from the output generated by the meshing algorithm. We then determine the error by comparing it with its real dimensions that were acquired using a measuring tape

Chapter 5 : Applications

- **Indoor mapping and modelling -** Real-time acquisition of dynamic environments: The modelling of dynamic environments, i.e. environments in which there are many moving objects (e.g., crowded rooms) or environments in which space reforms (e.g., industrial plants) is presently done off-line. The challenge is to do this in real-time and on mobile devices
- **Indoor navigation with obstacle avoidance -** Before navigational information can be extracted, the geometric model of the indoor environment has to be attributed/semantically enriched and structured in a geometric form ideal for navigational. These navigational models differ in their treatment of the continuity of indoor space and by extension
- **Augmented systems -** Augmented reality (AR) is an interactive experience of a real-world environment where the objects that reside in the real world are enhanced by computer-generated perceptual information, sometimes across multiple sensory modalities, including visual, auditory, haptic, somatosensory and olfactory. AR can be defined as a system that fulfills three basic features: a combination of real and virtual worlds, real-time interaction, and accurate 3D registration of virtual and real objects
- **Gaming -** Traditionally 3D games have relied on the design of virtual indoor environments. However, as 3D indoor models become common, it is likely that there will be greater use of ‘real’ indoor models. Eventually as augmented systems mature, it can be expected that a new genre of games will emerge in which games are played in real indoor environments with the benefit of augmented reality
- **Finding the optimum or shortest path -** By creating a mathematically accurate map or model of the indoor environment, a robot can navigate around the entire scene effectively by calculating the shortest path between the source and destination points. This allows a multitude of different tasks such as indoor warehouse delivery and pipeline inspection that the robot can perform efficiently

- **Obstacle classification** -The RGB system present onboard combined with image processing techniques such as COCO or Tensorflow can be used to determine the obstacles that are present in the indoor environment. We can further classify these obstacles based on the amount of threat that they pose to the robot
- **Unification of outdoor and indoor models -** The development of modelling standards for 3D city models is well developed, largely because outdoor data acquisition and modelling has been active for many years. Indoor modelling is a more recent activity and the models are not as mature as those of outdoor models. Ultimately a seamless transition between outdoor and indoor models is desired, and this is an active area of research
- **Indoor modelling for crisis response -** Evacuating an area during a time of crisis has for decades been an active area of research. Applications for this purpose have to be able to determine safe routes and guide users through safe routes. This requires sensors to capture the state of the indoor environment, relate sensory information to the indoor model, determine viable escape routes and finally convey the escape route to the user, either visually or aurally

Chapter 6 : Hardware and Software Requirements

6.1 Hardware requirements

1. Intel Realsense Depth Camera
2. Intel Realsense Tracking Camera
3. USB 3.0
4. Core i5 processor or above

6.2 Supported OS environments

1. Linux
2. Ubuntu 16.04 or above
3. Windows 10
4. Mac OSX

6.3 Software requirements

1. Python3
2. Open3D
3. PyVista
4. LibRealSense SDK
5. Pyrealsense library
6. Meshlab
7. CAD Software

Chapter 7 : Summary & Conclusions

7.1 Summary

- The first hurdle of solving the navigation problem was interfacing the cameras.
- The retrieved depth data needed to be plotted to visualize the acquired values. We were able to take the x, y, depth cube and slice it along the y axis to retrieve x, depth values to obtain the top view of the environment. This approach however lacked any meaningful data relating to the actual environment.
- After multiple tests, we found that the previous system generated excess redundant points. Hence, we optimized the depth gathering data to retrieve only the central row of the depth frame.
- Previous tests were conducted using only a single depth frame which held the coordinates of only one frame which would be lost due to the need of restarting the program. We developed a method where multiple depth frames could be stitched together. Hence n depth frames could be plotted as opposed to a singular frame. The limitation however was the need for human input.
- The problem of human input was solved by using a pose estimation camera which provided all the necessary parameters to plot the map dynamically i.e. without the need for any human intervention. Further optimizations were conducted but one of the essential goals of the project had been realized by this stage.
- After the 2D plot was completed, the 3D environment reconstruction system was developed. The same logic i.e. Cartesian coordinate calculation and redundant point removal that was used in the 2D plotting algorithm was extended to account for the new dimension in the 3D plotting algorithm. The output of this algorithm resulted in the generation of a point cloud dataset. This dataset was then utilized by the surface generation algorithm.
- In the next step, a refined method of Delaunay Triangulation called Ruppert's Algorithm was used to generate a mesh or surface as described in the surface generation algorithm. Thereafter, each surface was smoothened and all the individual surfaces were added up to generate a mathematically accurate model of the environment.

7.2 Conclusion

- One of the goals of our project was to create a 2D map dynamically without any human intervention which has been successfully achieved. Although a few results were found which deviated from the original distance with the maximum being 28cm, this is a data acquisition issue. The core algorithm for converting the spatial coordinates to a world coordinate does not cause any problem.
- The main task of the project was to develop an algorithm that converted depth data to a working and mathematically accurate 3D model. We obtained the results of this algorithm and measured the percentage of error with respect to the real world values and found an error percentage within reasonable bounds

7.3 Future Work

- Although the dynamic generation algorithm works as intended, certain issues in the data acquisition can cause discontinuities and/or deviations in the desired output. We plan on addressing this issue by either omitting data points that result in this discontinuity, or normalize the data acquisition so that it does not cause significant changes.
- The dynamic 3D generation algorithm output requires post processing to get a completely accurate result without discontinuities and holes generated by light sources which is currently achieved using a separate software. We plan on incorporating a solution for the same in our proposed system.
- Object detection technologies are a separate system and can be combined with the proposed system to achieve object detection and mapping simultaneously

7.4 References

- Y. Cui, S. Schuon, D. Chan, S. Thrun and C. Theobalt, "3D shape scanning with a time-of-flight camera," 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Francisco, CA, 2010, pp. 1173-1180.
- Henry, Peter et al. “RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments.” ISER (2010).
- Izadi, Shahram & A. Newcombe, Richard & Kim, David & Hilliges, Otmar & Molyneaux, David & Hodges, Steve & Kohli, Pushmeet & Shotton, Jamie & J. Davison, Andrew & Fitzgibbon, Andrew. (2011). KinectFusion: Real-time dynamic 3D surface reconstruction and interaction. ACM SIGGRAPH 2011 Talks. 23
- Zlatanova, Sisi & Sithole, George & Nakagawa, Masafumi & Gist, A. (2013). Problems In Indoor Mapping and Modelling. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. XL-4/W4. 10.5194/isprsarchives-XL-4-W4-63-2013.
- Romanoni, Andrea & Delaunoy, Amaël & Pollefeys, Marc & Matteucci, Matteo. (2016). Automatic 3D Reconstruction of Manifold Meshes via Delaunay Triangulation and Mesh Sweeping. 10.1109/WACV.2016.7477650.
- C. Aluckal et al., "Dynamic real-time indoor environment mapping for Unmanned Autonomous Vehicle navigation," 2019 International Conference on Advances in Computing, Communication and Control (ICAC3), Mumbai, India, 2019, pp. 1-6, doi: 10.1109/ICAC347590.2019.9036813.