SOFTWARE ENGINEERING

PROJECT REPORT

3D Scanner

January 9, 2017

Hassan Saeed
Islam Mokhtari
Ivan Mikhailov
Katherine Sheran
Minh Vu
Saed Khwaldeh
Solene Guillaume
Tajwar Aleef
Usama Pervais
Yeman Hagos

Contents

Abstract **3**

List of Figures **6**

List of Codes **7**

1 Introduction **8**
1.1 Overview 8
1.2 Objectives 8
1.3 Structure of Report 9

2 Background **10**
2.1 Kinect Sensors 10
2.2 3D Reconstruction 11

3 Related Tools and Libraries **13**
3.1 Latex 13
3.2 Git 13
3.3 QT Framework and QT Creator 14
3.4 Microsoft SDK 14
3.5 Open NI 2.0 14
3.6 Point Cloud Library 15
3.7 Kinect Fusion 15
3.8 Arduino and Arduino Integrated Development Environment 15

4 Getting Started **17**
4.1 Project Management 17
4.1.1 Self-score Table 17
4.1.2 Weekly Meeting 18
4.1.3 Weekly Internal Report 18
4.1.4 Task of Requirements 19
4.2 Coding Standard 19
4.3 Platform 20
4.4 Seminars 21

5 Implementation **25**
5.1 Graphical User Interface . . . 25
5.2 Surface Measurement . . . 26
5.2.1 PassThrough Filter . . . 26
5.2.2 Statistical Outliers Removal Filter . . . 28
5.3 Sensor Pose Estimation . . . 29
5.3.1 Overview of Iterative Closest Point . . . 29
5.3.2 Literature review . . . 30
5.4 Smoothing . . . 33
5.4.1 Moving Least Squares . . . 33
5.4.2 Laplacian Smoothing . . . 34
5.5 Mesh reconstruction . . . 35
5.5.1 Greedy Triangulation . . . 36
5.5.2 Grid Projection . . . 37
5.5.3 Poisson . . . 39
5.6 Holes Filling . . . 40

6 User Guide **44**
6.1 Home Screen Window . . . 44
6.2 Calibration Frame Window . . . 45
6.3 Registration and Filtering Window . . . 45
6.4 Previously Scanned Files . . . 46

7 Results and Conclusion **50**
7.1 Results . . . 50
7.2 Conclusions . . . 52

Bibliography **55**

Abstract

In the following work, a 3D reconstruction software using Kinect and Point Cloud Library (PCL) for scanning, reconstructing and exporting a 3D model of a person into a 3D printer is explained in detail. First, a brief introduction to 3D reconstruction and an explanation of the depth measuring technology utilized by the Kinect v2.0 is given. Some fundamental concepts necessary to understand the procedure of how the Kinect can perform a digital reconstruction of an object is also given.

This project is based on open source libraries and frameworks. So next, a presentation of the tools and libraries utilized during the design of the software is provided. The main source used for this project is the PCL, being an open source project, free for commercial use and with wide contents of online documentation, PCL is one of the key core elements for the development of this project.

Additionally, an explanation of the inner management of the project, like division of time and tasks, defining deadlines and setting the coding standards, is also provided in detail. Regarding the GUI, some images are included in order to explain how the user can perform a new scan or open a recent project.

As for the implementation of the reconstruction, the process is divided in four main tasks: Surface Measurement, Sensor Pose Estimation, Smoothing and Surface Extraction. Surface Measurement consists on the scanning and obtaining several point clouds from the Kinect. For this scanning process a turntable was used in order to obtain around 37 point clouds with an approximate interval of 10 degrees per each one point cloud scan. This data is filtered by a pass through filter and some outliers removal procedure. For the Sensor Pose Estimation, the Iterative Closest Point (ICP) algorithm is performed in order to register the obtained point clouds from the Kinect into one. For removing noise and sharp edges from the surface, smoothing is performed by Moving Least Squares (MLS) and Laplacian Smoothing (LS). Finally, for reconstructing the mesh, Greedy Triangulation (GT) and Grid Projection (GP) is performed. The Poisson reconstruction method was also implemented but as the algorithm was the slowest and required more extra cleaning of the mesh, this method was discarded.

The report includes a detailed explanation of the mentioned techniques as well as some examples of images to demonstrate the final output obtained. The proposed software successfully scans and performs a 3D reconstruction of a person, however for obtaining more detailed features of the face, it

is recommended to collocate the Kinect closer to the person in order to avoid loss of data.

List of Figures

2.1 Main elements of the Kinect . . . 10
2.2 Sensor comparison . . . 11
2.3 3D reconstruction steps . . . 12

3.1 Arduino . . . 16

4.1 Self-score table . . . 18
4.2 Internal weekly report . . . 19
4.3 Task of Requirements . . . 23
4.4 Turntable . . . 24
4.5 Turntable . . . 24

5.1 PassThrough Filtering process . . . 27
5.2 Statistical Outliers Removal Filter (SOR) . . . 28
5.3 SOR effect . . . 29
5.4 ICP performed on the Stanford bunny . . . 30
5.5 Visual example of how the ICP iterates and brings point clouds into the same coordinate reference . . . 31
5.6 MLS effect . . . 34
5.7 Before and after applying LS . . . 35
5.8 Greedy Triangulation performed on the Stanford bunny . . . 37
5.9 Before and after apply GT . . . 38
5.10 Grid Projection performed on the Stanford bunny . . . 38
5.11 Before and after apply GT . . . 39
5.12 General Overview of the Poisson method . . . 39
5.13 Poisson reconstruction performed on the Stanford bunny . . . 40
5.14 Poisson reconstruction with depth=10 . . . 41
5.15 Holes Detection . . . 42
5.16 Holes Detection . . . 42
5.17 Holes Detection . . . 43

6.1 Home Window . . . 44
6.2 Calibration Frame Window . . . 47

6.3 Calibration Frame Window 48
6.4 4 Previously Scanned Files 49

7.1 Merging scans from the Kinect 51
7.2 Greedy Triangulation reconstruction with mu=5, D=50, R=0.025 51
7.3 Grid Projection recontruction with res=0.0025 52

Listings

5.1 Create PCL Visualition 25
5.2 Create PCL Visualition 25
5.3 Create PCL Visualition 25
5.4 PassThrough Filter 27
5.5 Statistical Outliers Removal Filter 28
5.6 Iterative Closest Point 31
5.7 Save point cloud and display to VTK 33
5.8 Moving Least Squares 33
5.9 Laplacian Smoothing 34
5.10 Greedy Triangulation 36
5.11 Grid Projection 37
5.12 Poisson 40

Chapter 1

Introduction

1.1 Overview

3D reconstruction is a technique used in computer vision which has a wide range of applications in areas like object recognition, city modeling, virtual reality, physical simulations, video games, special effects, just to name a few.

To perform a 3D reconstruction, specialized hardware is required. This equipment was often very expensive and available only for industrial or academic purposes. Nowadays, high quality 3D scanners are now available at a low cost, making research available for everyone at home.

3D printing technology has also made a breakthrough in recent years. By taking a digital concept and placing the physical results of that concept into the user's hand, 3D printing is without a doubt a promising technology that will only become more commonplace as it continues to be refined [8].

The combination of both technologies, 3D reconstruction and 3D printing, into a general 3D copy machine bears an enormous economical potential for the future [15] hence the urgent need to obtain affordable and meaningful 3D content.

1.2 Objectives

The objective of this work is to design a homemade acquisition and processing software to perform 3D reconstruction of a full sized person. Our final aim is to replace the Skanect software by our own, home-made acquisition and processing software so that we no longer depend on proprietary software.

To achieve this goal there are three main steps to perform. First, we need to acquire data of a person in less than ninety seconds using a inexpensive sensors. Second, convert the acquired point cloud data into a watertight mesh of good quality. Third, export the reconstructed model to a 3D printer to obtain proper prints of the model.

1.3 Structure of Report

After this introduction, in **Chapter 2** the Kinect scanner and the methodology for 3D reconstruction is reviewed. This chapter covers some fundamental concepts necessary to understand the procedure of how the Kinect can perform a digital reconstruction of an object.

Chapter 3 presents the different tools and libraries that were used. This project is based on open source libraries and frameworks. In this chapter these technologies are presented with more detailed information.

Next, **Chapter 4** will present the inner management of the project. From division of time and tasks, defining deadlines, setting the coding standard and the design of the weekly reports.

Following that, **Chapter 5** will discuss the general procedure of performing a 3D reconstruction, and will elaborate on the implementation of the selected method that was used to generate the current results.

In **Chapter 6** a user guide with detailed instructions and explanations on how to use the software is included.

The results and conclusions will be discussed in **Chapter 7**, which will include a variety of images of the different reconstruction methods applied, followed by the team conclusions regarding the design of this software.

Chapter 2

Background

2.1 Kinect Sensors

The Kinect (see Figure 2.1 [1]) is a light laser scanner that generates real time depth maps. It obtains a colored 3D point cloud, also known as RGB-D image, which provides information about the color and the depth of each scanned pixel [17].

The Kinect consists of three main elements:

- **IR light emitter:** beams IR rays light into the scene.
- **Monochrome CMOS sensor:** captures the reflected IR light.
- **RGB camera:** traditional color camera.

The Kinect converts the pattern obtained by the CMOS sensor into a depth map. The depth map is aligned to the color picture, obained by the RGB camera, and they are combined into a single RGB-D image that is sent towards the USB port. [3]

Figure 2.1: Main elements of the Kinect

Figure 2.2 below shows the comparison between existing sensors. Three sensors, including Kinect 1,2

[1] Figure 2.1 was adopted from `http://www.msdn.microsoft.com`

and Intel provided to be used for this project, are highlighted in red rectangular windows.

	Orbbec Persee	RealSense SR300	Orbbec Astra	RealSense R200	FED Stereo Camera	RealSense F200	Kinect for XBox One	UO mini Ix	Leap Motion	Kinect for XBox 360
Released	December 2016	March 2016	September 2015	September 2015	May 2015	January 2015	July 2014	May 2013	October 2012	June 2011
Price	$240	$150	$150	$99	$449	$99	$100	$695	$100	Unavailable
Tracking Method	IR	IR	IR	IR	Stereo RGB cameras	IR	IR	Passive IR	IR	IR
Range	0.4m – 8m	0.2m – 1.2m	0.4m – 8m	0.5m – 3.5m	1.5m – 20m	0.2m – 1.2m	0.5m – 4.5m	0.3m – 2.4m	0.025m – 0.6m	0.4m – 4.5m
RGB Image	1280×720, 30 FPS	1920×1080, 30 FPS	1280×960, 10 FPS	1920×1080, 30 FPS	configurable between 1280×480, 120 FPS and 4416×1242, 15 FPS	1920×1080, 30 FPS	1920×1080, 30 FPS	configurable between 320×120, 360 FPS and 752×480, 56 FPS	N/A	640×480, 30 FPS
Depth Image	640×480, 16 bit, 30 FPS	640×480, 60 FPS	640×480, 16 bit, 30 FPS	640×480, 60 FPS	configurable between 640×480, 120 FPS and 2208×1242, 15 FPS	640×480, 60 FPS	512×424, 30 FPS	configurable between 320×120, 360 FPS and 752×480, 56 FPS	20 to 200+ FPS	320×240, 30 FPS
Connectivity	Ethernet	USB 3.0	USB 2.0	USB 3.0	USB 3.0	USB 3.0	USB 3.0	USB 2.0	USB 2.0	USB 2.0
Physical Dimensions	172×63×56 mm	14×20×4 mm	160×30×40 mm	130×20×7 mm	175×30×33 mm	150×30×58 mm	250×66×67 mm	52×25×11 mm	76×30×17 mm	280×64×38 mm
Works outdoors?	✗	✗	✗	✗	✓	✗	✗	✓	✗	✗
Skeleton tracking?	✗	✗	✓ (only hand positions)	✗	✗	✗	✓ (six skeletons)	✗	✗	✓ (two skeletons)
Facial tracking?	soon	✓	✗	✓	✗	✓	✓	✓	✗	✓
3D scanning?	✗	✓	✗	✓	✓	✓	✓	✓	✗	✓
Simultaneous apps?	soon	✓	✗	✓	✗	✓	✓	✗	✗	✗
Gesture Training?	✗	✗	✗	✗	✗	✗	✓ (Visual Gesture Builder)	✗	✓	✗ (only via third-party tools)
Gesture Detection?	✗	✓	✗	✗	✗	✓	✓ (hand open, closed, lasso)	✗	✓	✓ (hand grip, release, press, scroll)
Toolkits	C++, Java, OpenNI, ROS	Java, JavaScript, Processing, Unity3D, Cinder	OpenNI	Java, JavaScript, Processing, Unity3D, Cinder		Java, JavaScript, Processing, Unity3D, Cinder	WPF, Cinder, OpenFrameworks, JavaScript, vvvv, Processing, Unity3D, more	Dense3D, OpenCV, Qt5	Javascript, Oculus Rift, Unity3D, Unreal	WPF, Cinder, OpenFrameworks, JavaScript, vvvv, Processing, Unity3D, more
Project Examples	Depth Data Viewer, RGB Data Viewer	Various face tracking examples, mostly with C++. Only one Unity3D sample.	HandViewer, Depth Data Viewer, RGB Data Viewer	Various face tracking examples, mostly with C++. Only one Unity3D sample.	Background subtraction, right image disparity, depth map	Many examples of face tracking, gesture tracking, speech detection on a variety of different platforms and frameworks	Many examples of skeleton tracking, face tracking, and speech detection on a variety of different platforms and frameworks	Very few samples in each of the supported languages, mostly to get raw image and depth data	There are a number of examples available for each of the lanugaes and platforms supported	Many examples of skeleton tracking, face tracking, and speech detection on a variety of different platforms and frameworks

Figure 2.2: Sensor comparison

2.2 3D Reconstruction

3D reconstruction is the process of creating 3 dimensional models based on a set of 2D images.

One of the requirements of this project was to use the Kinect sensor for doing the reconstruction. The Kinect sensor is a depth camera that can provide information of the distance (depth) between the camera and the object you want to reconstruct based on its time of flight (TOF).

There are 4 main steps to be followed for doing a 3D reconstruction using the Kinect:

- **Surface Measurement:** the Kinect provides us the depth of a set of 2D images.
- **Sensor Pose Estimation:** Using the Iterative Closest Point (ICP) algorithm we minimize the difference between the current and the previous frame. We then apply the transform matrix to the point cloud to obtain the global coordinate system.
- **Smoothing:** Sharpness and noise is removed to obtain a smoother mesh by applying Moving Least Squares and Laplacian smoothing.
- **Surface Extraction:** Extract the surface from the global model and display it on screen.

Figure 2.3: 3D reconstruction steps

Chapter 3

Related Tools and Libraries

3.1 LATEX

LateX is a system where you can prepare documents for professional use like scientific articles, books, presentations, to name a few [2]. In addition, LateX encourages its users to concentrate on having the right content in their documents instead of focusing on how their documents look like. We have decided to use LateX for this report as it provides us many advantages over using other traditional software like Microsoft Word, for instance, using LateX handles mathematical equations and expressions in very good way which makes it very comfortable for the user. Moreover, LateX helps us to fit any kind of content inside our report, by this we focus more on our project and stop worrying about how our report will look like at the end.

3.2 GIT

Git is an open source control system to manage and share cooperative work, especially for programming. Git is secure as it protects the code and the change history against both accidental and malicious change and ensures that the history is fully traceable. It is very optimized as it allows to commit changes, create branches, merge codes, and compare it to past versions of itself. It recognizes the attributes of real source code file trees, how they are modified over time and which are the access patterns. It is also flexible as it supports various kinds of nonlinear development work flows.

Github is a web based Git repository hosting service. It was used in this project as a free central repository to store all the data in a cloud. It allowed each developer to always have a full local copy of the repository with the development history. Even if a local hard drive fails or if anyway the data is lost, all the data can be recovered. In Github it is possible to store projects in the Git format and from

there extract the data to use it with the Git tools [5]. For this reasons, Git was the best option for the communication, speed and efficiency of the project.

3.3 QT FRAMEWORK AND QT CREATOR

Qt is a cross platform application framework for developing software that can be run on various software and hardware platforms with little to no change in the underlying code base. It can be run natively with GUI on Linux, OS X and Windows operating systems. And applications can be created for multiple desktop, embedded, and mobile device platforms such as Android and iOS.

Qt is based on C++ and it has simple and optimized tools that increase the speed of the developed applications. Moreover it has a GUI editor which is simple to use. Compared to other Integrated Development Environment (IDE), like Visual Studio, Qt has full C++ features, a better object support, a nice GUI editor and most importantly, it is compatible with most operating systems.

3.4 MICROSOFT SDK

A Software Development Kit (SDK) is a set of tools used to develop software for a determined operating system . Microsoft Windows SDK, is an SDK from Microsoft that provides tools, documentation, header files, libraries, code samples and compilers that developers can use to create applications that run on Microsoft Windows. Windows SDK can be used with the Kinect sensor to develop applications that support gesture and voice recognition, using Kinect sensor technology on computers using Windows.

3.5 OPEN NI 2.0

The OpenNI 2.0 Application Programming Interface (API) provides access to PrimeSense compatible depth sensors. It allows an application to initialize a sensor and receive depth, RGB, and IR video streams from the device. It provides a single unified interface to sensors and ONI recording created with depth sensors.

OpenNI also provides an uniform interface that third party middleware developers can use to interface with depth sensors. Applications are then able to make use of both the third party middleware , as well as underlying basic depth and video data provided directly by OpenNI.

3.6 Point Cloud Library

The PCL is an open source project for point cloud processing written in C++. It is aimed to be a cross platform library which is why it is successfully compiled and deployed on Linux, Windows, MacOS and Android [5].

The PCL framework contains numerous state of the art algorithms for point cloud processing, including filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation. These algorithms can be used, for example, to filter outliers from noisy data, stitch 3D point clouds together, segment relevant parts of a scene, extract key points and compute descriptors to recognize objects based on their geometric appearance, create surfaces from point clouds and visualize them [14].

Since PCL provides all required functionality from the data acquisition to the surface creation and because it is free for commercial and research use, it was the selected library for the implementation of this project.

3.7 Kinect Fusion

Microsoft Kinect Fusion reconstructs a 3D model of an object by using the data from the Kinect sensor. It allows the user to simultaneously see and interact with a detailed 3D model of the scene.

3.8 Arduino and Arduino Integrated Development Environment

Arduino is an open-source platform used for building electronics projects. Arduino consists of both a microcontroller and and software, or IDE that runs on user computer, used to write and upload computer code to the physical board.

Figure 3.1 shows an Arduino [1].

In this project, Arduino is used for the Turntable platform mentioned in section 4.3 at page 20.

[1] Figure 3.1 was adopted from `https://learn.sparkfun.com/tutorials/what-is-an-arduino`

Figure 3.1: Arduino

Chapter 4

Getting Started

4.1 Project Management

4.1.1 Self-score Table

At the first meeting, a group of 11 members was formed and each one of us completed the self-evaluation (see figure 4.1). After the meeting, roles in the team were determined:

- Note taker
- Representative: send emails, group post, meeting organization, Edmodo posts.
- Weekly Report Writer: compile 3 subgroups reports and send to Professor, with previous authorization of the project manager.
- Project Manager: make sure everything works smoothly.

Three major subgroups were formed:

- Group 1: GUI
- Group 2: Functionality
- Group 3: Implementation

Roles in subgroup were assigned:

- Researcher & Programmer: everyone
- Writer: write report of the progress of group every week
- Logistics: do miscellaneous work
- Group Leader: split the work, approve report from writer, and prepare tasks for next week.

Member	Group	Self-score 1-10								Roles
		Project Management	Programming	UI/UX Design	Research	Logic Algorithm	Writing	Presentation Slides	Logistics	
Yeman	1	4	5	1	7	9	8	10	7	
Sharon		7	5	3	10	7	10	10	8	Main Writer
Minh		9	7	7	9	10	7	9	7	Project Manager
Solene	2	6	6	0	5	6	6	6	6	
Saed		7	4	5	10	10	7	10	9	Representative
Sam		9	8	3	10	7	7	9	9	Note Taker
Ivan	3	8	7	6	8	7	9	9	8	
Aleef		4	6	4	8	8	9	10	8	
Hassan		4	7	3	7	7	6	5	6	
Andrey		7	10	5	8	9	5	7	0	
Islam		7	7	5	8	7	9	9	9	

Figure 4.1: Self-score table

4.1.2 Weekly Meeting

After weekly meeting, we discussed and finalized the tasks for next week. An example can be see below:

1. Group 1 (GUI)

    - How to design GUI in Qt?
    - What is Flat Design?
    - Research on well-designed C++ software for 3D reconstruction

2. Group 2 (Functionality)

    - Skanect software: capabilities, strengths, weaknesses, and algorithms.
    - What is Kinect? How does it work?
    - Can we use mobile phone for this project?

3. Group 3 (Implementation)

    - ReconstructMe: is it feasible? What kinect version is compatible with it?
    - Research on existing SDK, libraries and frameworks
    - Coding standard for C++

4.1.3 Weekly Internal Report

To keep track of the progress and contribution of every member, we generated a table and asked everyone to fill in the form (see figure 4.2). In general, there are three questions:

1. What have you done for last week?

2. Which problems have you faced?

3. What is your next step?

NAME	GROUP	WORK DONE	PROBLEMS	NEXT STEP
Minh	1			
Yeman				
Sharon				
Sam	2			
Saed				
Solene				
Islam	3			
Ivan				
Andrey				
Hassan				
Tajwar				

Figure 4.2: Internal weekly report

4.1.4 TASK OF REQUIREMENTS

In software development, task of requirements include those tasks that the assigned resources need to deliver at final stage. Understand the requirements of the project help us prepare the task requirements as figure 4.3 [4].

4.2 CODING STANDARD

In this project, the naming coding standard below is followed as much as possible:

1. English should be the only language used for naming any entity or for commenting.
2. Function names, variable names, and filenames should be as descriptive as possible.
3. Variable names and types should be nouns.
4. Function names usually should be phrased as an imperative
5. Variable names should begin with a lower case letter.
6. Words embedded in variable names begin with either an upper case letter or an underscore (_).
7. Constants names should be all upper case and words should be separated by underscore (_).
8. Functions should start with a capital letter and have a capital letter for each new word. No underscores.

9. Accessors and mutators (get and set functions) should match the name of the variable they are getting and setting.
10. Types (classes, structs, typedefs, and enums) begin with an upper case letter and have a capital letter for each new word. No underscores.
11. Filenames should be all lowercase and can include underscores. C++ files should end in .cc and header files should end in .h.
12. Variables representing GUI components should preferably be suffixed by the component type name. Examples : mainWindow, propertiesDialog.
13. The plural form should preferably be used for names representing a collection of objects. Examples : vector<Point> points; int values[];
14. The prefix "is" should preferably be used for boolean variables and methods.

4.3 PLATFORM

The reason for integration an automated turntable (see figure 4.4 and 4.5) along with our software is because it delivers a more efficient way of taking point cloud and provides clouds which are already aligned to some extent. This is because the turn table rotates around a pivot and basically this means the person or thing being scanned is stationary and is rotating around a fixed axis. Also, it allows us to have a simple click to initiate function and it lets the software and turntable to take care and of the rest.

For the mechanical part of building the turntable, the initial plan was to design a sample model before beginning. Sample sketches were first made for the design. The plan was to make a simple turntable that can support and perform flawlessly without much complications. This was because of the lack of resources and tools. The idea was to use a simple twistboard as the platform is already designed to support a person. A twistboard is basically a round platform which rotates all the way to 360 degree and it is used mainly for exercising purpose. As a twist board can already support the weight of average body size and also is free to rotate, we decided to hack it to be the final part of the turntable.

For powering this twistboard, we used a 12V high torque 8rpm dc gear motor to power the whole system. We did research and found that this motor provides enough power to move the turntable with a person on it. To control the motor we used a L298N dual motor controller module and it was powered by a power supply taken from the lab.

An industrial size belt was used to convey the power from the motor to the turntable. We did not have a complete loop of the belt so we opted for using manual method of gluing and duct taping the belt shut into a loop.

Since we wanted an automated system that is user friendly and can fully communicate with the software, we opted to use Arduino for controlling the motor for the turntable. Serial communication was used to talk back with the Arduino using our software.

For the Arduino coding part, a very simply control function was visualized. We also added a green and a red led to give a physical visual feedback. The serial input takes three values from the software.

1. When our software sends "1" through the serial port, the motor pin output is set to 'high' and also the pin for green led is set to 'high' and the pin for red led is set to 'low'. This means both the motor and green led turns on and red led turns off when a '1' is passed.
2. When our software sends "0" through the serial port, the motor pin output is set to 'low' and also the pin for red led is set to 'high' and the pin for green led is set to 'low'. This means both the motor and green ledturns off and the red led turns on when a '1' is passed.
3. For a measure of safety, an extra condition was set, that is when our software sends '2' to the Arduino, all pins are set to 'low'. Meaning everything in the turntable turns off.

The Arduino also had a reset switch which basically turns the table in reverse. Since the belt is manually joined, the part where the joint is made is more thicker so it cannot go around the motors commutator and needs to be moved back to the initial position. This reset switch allows a quick way of reseting the belt back to its initial position.

This Arduino and the Kinect grabber part of the code is integrated so they can work together hence meaning we were able to achieve full automation of grabbing point cloud data.

In the Kinect grabber we used 'timer' as a measure of control for the increments taken by the turn table. After every 1.8 seconds, '0' was sent to the Arduino to stop the turntable. A point cloud of that frame was taken and then again a '1' was passed to turn on the turntable for 1.8 seconds again. This way point clouds were taken at 1.8 second increments until a full 360 degree of point cloud was taken. Considering the speed of the motor, for ever 1.8 second we observed around 10 degree increment of the turntable. Hence our software saves around 37 points clouds. '2' is sent at the end of the 360 degree acquisition to stop the turn table.

4.4 SEMINARS

In order to get everyone of our team members along with our classmates on the same page regarding different topics related to our semester project like coding standards and using LateX, we decided to conduct some seminars to cover these topics.

Two of these seminars has been conducted to our classmates where 30+ students have attended and showed a lot of interest in the content of the two seminars we presented in our classroom in Codorcet.

The first seminar was on coding standards explaining why we should follow some standards when programming our C++ project. The seminar was mainly about how to write the different parts of the code while making it maintainable.

Furthermore, the second seminar was on the basics of LateX, in this seminar we explained to our classmates the basics of using LateX including how to include images, tables, matrices in LateX, also, we showed the students how to create their first project and how to make the bibliography using LateX. The LateX session included a presentation along with demonstration using ShareLateX.com platform so the students got to know how to handle dealing with LateX platforms. Apart of the two seminars we presented to our classmates, we conducted couple of other semi-seminars for our group members to ensure that they will all get their skills improved and learn from other group members.

	Task Name	Start Date	End Date	Duration	Status
2	Video Streaming	10/18/2016	10/25/2016	6d	Completed
3	**Configuring Kinect**	10/18/2016	10/25/2016	6d	Completed
4	Configure Kinect 1	10/18/2016	10/25/2016	6d	Completed
5	Configure Kinect 2	10/18/2016	10/25/2016	6d	Completed
6	Grabbing data + saving data in a PCD file	10/18/2016	10/25/2016	6d	Completed
7	**Streaming**	10/25/2016	10/25/2016	1d	Completed
8	Visualization of Point Cloud Data	10/25/2016	10/25/2016	1d	Completed
9					
10	Data for 3D Reconstruction	10/25/16	11/15/16	16d	Completed
11	**Raw Data + Integrating Data with PCL prgrams**	10/25/16	11/01/16	6d	Completed
12	Saving data as received from Kinect	10/25/16	11/01/16	6d	Completed
13	Using data in PCL program for different operation (triangulation...)	10/25/16	11/01/16	6d	Completed
14	**Filtering + sorting data ready to use**	11/02/16	11/15/16	10d	Completed
15	Filtering	11/02/16	11/15/16	10d	Completed
16	Storing data (as a file or in database)	11/02/16	11/15/16	10d	Completed
17					
18	3D Reconstruction + GUI Ready	12/15/16	12/29/16	33d	Completed
19	**Mesh reconstruction**	12/15/16	11/22/16	6d	Completed
20	Poisson, Grid Projection, Greedy Triangulation	12/15/16	11/22/16	6d	Completed
21	Smoothing	12/15/16	12/22/16	6d	Completed
22	**Integrating GUI with PCL code**	12/22/16	12/29/16	6d	Completed
23	Configure PCL visualizer with QT	12/22/16	12/29/16	6d	Completed
24	Configure the main functions from PCL code in the GUI	12/22/16	12/29/16	6d	Completed
25					
26	3D Complete Model	12/29/16	12/15/16	13d	Completed
27	**First protoype**	12/29/16	12/09/16	9d	Completed
28	First software prototype	12/29/16	12/09/16	9d	Completed
29	Last prototype and final software	12/29/16	12/09/16	9d	Completed
30	**First tests**	12/29/16	12/15/16	5d	Completed
31	Tests and debugging	12/29/16	12/15/16	5d	Completed
32					
33	Final Product	12/31/16	01/02/17	10d	Completed
34	**Database Management**	12/31/16	12/31/16	8d	Completed
35	Save final scan in a file or a database	12/31/16	12/31/16	8d	Completed
36	**Depuring code**	01/01/17	01/02/17	2d	Completed
37	Clean code	01/01/17	01/02/17	2d	Completed
38	Software ready for use	01/01/17	01/02/17	2d	Completed

Figure 4.3: Task of Requirements

Figure 4.4: Turntable

Figure 4.5: Turntable

Chapter 5

Implementation

5.1 Graphical User Interface

QT does not have a widget dedicated to display point clouds. Therefore, we interfaced Visualization ToolKit (VTK) to display point clouds and mesh files. The VTK included with the PCL1.8 was not responding properly as intended.Hence, we downloaded VTK-7.0.0 and added its release folder to the systems path. Finally, it is interfaced to QT Graphical User Interface (GUI) as follows:

Step 1: Create PCL Visualition object

Listing 5.1: Create PCL Visualition

```
boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer1;
```

Step 2: Setup the QVTK Widget

Listing 5.2: Create PCL Visualition

```
// Set up the QVTK window
viewer1.reset (new pcl::visualization::PCLVisualizer ("viewer1", false));

// setup the rendering window of qvtk widget the viewer created
ui->qvtkWidget->SetRenderWindow (viewer1->getRenderWindow ());

// set the interactor of the viewer the interactor of qvtk widget
viewer1->setupInteractor (ui->qvtkWidget->GetInteractor (), ui->qvtkWidget->GetRenderWindow
```

Step 3: Add point cloud or mesh file to the viewer:

Listing 5.3: Create PCL Visualition

```
viewer1->addPointCloud (pcdOpen, "pcdOpen");          // For point clouds
viewer1->addPolygonMesh (mesh, pathName);     // For polygon mesh file
```

The remaining elements of our GUI are the normal UI QT widgets.

5.2 Surface Measurement

Errors of depth data of the Kinect can make the reconstructed surface unsmooth if they are not reduced. These errors can be categorized in:

1. **Gross error when edges of object surface are scanned:** The infrared light emitted from Kinect sensor will not be well reflected when encountering some sharp object or the edge of surface. This will result in an unreliable data of the edge, e.g. a smooth linear edge appear to be unsmooth in the depth image obtained from Kinect. This kind of gross error will severely harm the result because we cannot assure whether the points near the edge is behind or in front of the surface.

2. **Gross error when objects that are not well reflective are scanned:** When encountering some specular objects such as metal or glass, e.g. the monitor of computer, Kinect senor can hardly detect the depth data of these surfaces because these types of material do not reflect the infrared light, so no structured-light depth reading was possible. As a result, the depth image contains numerous holes where the data are missing.

3. **Small Gaussian noise**: Obtained from depth map.

A pre-process to reduce the noises and eliminate the gross error is required every time a raw dept map is obtained. This pre-process consists of applying a Bilateral Filter, which reduces the noise and close the small holes while preserving the details of the image.

5.2.1 PassThrough Filter

This PassThrough Filter (PTF) removes the points of the cloud which lie outside a given range in the 3 directions X,Y and Z. In our project, we are applying this filter while acquiring our raw data (in real time), moreover, we are giving the user the choice to define the range which he wants to keep our software to consider for scanning. Below you can see the part of our code which does the PassThrough filter in real time.

The filtering process can be seen in 5.1 [1].

[1] Figure 5.1 was contributed by PCL from `http://pointclouds.org/documentation/tutorials/`

Figure 5.1: PassThrough Filtering process

PTF C++ codes can be implemented as below:

Listing 5.4: PassThrough Filter

```
// Create the filtering object
pcl::PassThrough<PointType> pass;

//Setting a range in X axis
pass.setFilterFieldName ("x");
pass.setFilterLimits (ui->xMinSlider->value()/10.0,
ui->xMaxSlider->value()/10.0);

//Passing only the points in the window set by the slider in X axis
pass.setInputCloud (cloud);
pass.filter (*cloud_filtered);

//Setting a range in Y axis
pass.setFilterFieldName ("y");
pass.setFilterLimits (ui->yMinSlider->value()/50.0,
ui->yMaxSlider->value()/50.0);

//Passing only the points in the window set by the slider in X axis
pass.setInputCloud (cloud_filtered);
pass.filter (*cloud_filtered1);

//Setting a range in Z axis
pass.setFilterFieldName ("z");
```

passthrough.php

```
pass.setFilterLimits (ui->yMinSlider->value()/10.0,
ui->zMaxSlider->value()/10.0);

//Passing only the points in the window set by the slider in Y axis
pass.setInputCloud (cloud_filtered1);
pass.filter (*cloud_filtered);
```

5.2.2 Statistical Outliers Removal Filter

Measurement errors cause sparsing outliers which destroy the results of scanning. This makes estimation of the PCL characteristics like surface normals, it is also leading to erroneous values, which might cause registration failures. These issue can be managed by applying a statistical analysis on each point's neighborhood, and removing those which do not meet certain conditions. The SOR (figure 5.2 [2]) we are applying is based on the computation of the distribution of point to neighbors distances in the input dataset. For each single point it computes the mean distance from it to all its neighbors. As 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.

Figure 5.2: SOR

The number of neighbors to analyze for each point is set to 50, and the standard deviation to 1. This means that all points who have a distance larger than one standard deviation of the mean distance to the query point will be marked as outliers and removed.

Listing 5.5: Statistical Outliers Removal Filter

```
// Statistical Outliers Removal Filter
pcl::StatisticalOutlierRemoval<pcl::PointXYZ> sor;
sor.setInputCloud (cloud_filtered);
sor.setMeanK (50); // k-mean value
```

[2] Figure 5.1 was contributed by user fferri from http://stackoverflow.com/questions/31076745/how-to-remove-noise-using-meanshift-clustering-technique

(a) Before SOR **(b)** After SOR

Figure 5.3: SOR effect

```
sor.setStddevMulThresh (1.5); // standard daviation value
sor.filter (*cloud_outliers);
```

5.3 Sensor Pose Estimation

The second main step for achieving our 3D reconstruction is the sensor pose estimation. Pose estimation consists on finding the transformation of an object in a 2D image so that the 3D object can be reconstructed from it.

After we finished scanning with the Kinect, a series of point clouds are generated. This point clouds need to be registered and merged into a 3D surface model. To achieve this, we need to estimate the 3D rigid transformation matrix of the sensor pose and transform the point cloud into the global system coordinate [16]. In this work, we have estimated the sensor pose using the ICP algorithm.

5.3.1 Overview of Iterative Closest Point

The ICP algorithm is used for geometric alignment of 3D models [3] when an initial estimate of the relative pose is known [13]. In the ICP algorithm, one point cloud, the reference, is kept fixed, while the other one, the source, is transformed to best match the reference. The algorithm iteratively revises the transformation needed to minimize the distance from the source to the reference point cloud.

[3] Figure 5.4 was taken from `doi.ieeecomputersociety.org`

Figure 5.4: ICP performed on the Stanford bunny

After scanning a person, the next task is to combine the consecutive scans into a single point cloud. To achieve that, the first frame is fixed as reference of the coordinate system to be used. All the other frames will be transformed into that coordinate system. This transformation can be calculated iteratively using the ICP algorithm which will give as a result the alignment of the successive individual scans of point clouds into a unified 3D global map [12].

5.3.2 Literature review

The basic principle of ICP algorithm is to find the rotation and translation parameters between two point clouds during iterations [17].The general steps of the algorithm are the following.

First, a reference point cloud must be chosen P_0.

Second, given a sequence of point clouds P_i, the ICP algorithm finds the transformations T_i such that

$$P_{i-1} \approx T_i P_i \tag{5.1}$$

where: $i = 1, .., N$

Applying transformations iteratively, the cumulative transformations TC_i can be found by

$$TC_i = T_i . TC_{i-1}, \tag{5.2}$$

where $i = 1, .., N$

Cumulative transformation matrices can be used to bring all point clouds into the same coordinate reference:

$$Q_i = TC_i * P_i \tag{5.3}$$

where $i = 0, ..., N$

Finally, the reconstructed point cloud Q is formed by the union of all aligned point clouds:

$$Q = \bigcup_{i=0}^{N} Q_i \tag{5.4}$$

Figure 5.5: Visual example of how the ICP iterates and brings point clouds into the same coordinate reference

C++ codes of ICP can be seen in

Listing 5.6: Iterative Closest Point

```
pcl::PointCloud<PointT>::Ptr cloud_downsampled (new pcl::PointCloud<PointT>);

result = data[0].cloud;

for (size_t i = 1; i < data.size () ; i++)
{
    cloud_previous = result ;
    cloud_current = data[i].cloud;

    int iterations = 100 ;
```

```

    // In this part of the code we applied ICP on the point coulds
    // grabbed by the kinect and we aligned them two by two while
    // setting the correspondance distance to 15cm and transformation
    // epsilon to 10^-8.
    icp.setInputCloud (cloud_previous);
    icp.setInputTarget (cloud_current);
    icp.setMaxCorrespondenceDistance(0.015);
    icp.setTransformationEpsilon (1e-8);
    icp.setMaximumIterations (iterations);
    icp.setRANSACOutlierRejectionThreshold(0.03);
    icp.align (*cloud_merged);

    // Here, we find the transformation matrix to go from frame2 to
    frame1
    // and we apply the transformation on the second point cloud using
    transformPointCloud method
    // and we finally add the two point clouds together once they are
    in the same frame.
    Eigen::Matrix4f transform = icp.getFinalTransformation();
    if (icp.hasConverged ())
    {
        pcl::transformPointCloud(*cloud_previous, *cloud_merged,
        transform);
        *cloud_merged += *cloud_current;
    }
    else
    {
        PCL_ERROR ("\n.Not Converged\n");
    }
    // In this part of the code we downsample the new point cloud that contains
    // both processed point clouds
    pcl::VoxelGrid<PointT> grid;
    grid.setInputCloud (cloud_merged);
    grid.setLeafSize (0.005f, 0.005f, 0.005f);
    grid.filter (*cloud_downsampled);

    // This is the part where we update the cloud of the first frame in "result"
    // so that we align the next point clouds with this new one.
    result = cloud_downsampled;

    std::cout << iterations << " is done." << std::endl;
```

```

}// end of for loop
std::cout << "ICP is done." << std::endl;
```

Listing 5.7: Save point cloud and display to VTK

```
// In this part of the code we save the point cloud that represent the
// whole body in a PLY file.
stringstream name;
name << ui->scanNameLineEdit->text().toStdString() <<"3D.ply";
string filename = name.str();
pcl::io::savePLYFile(filename, *cloud_downsampled);
std::cout<<"PLY file is saved" << std::endl;

// In this part of the code we add the point cloud to the vtk viewer
// in order to view it in the Qt application
viewer2->addPointCloud (cloud_downsampled, "cloud");
viewer2->resetCamera ();
```

5.4 Smoothing

5.4.1 Moving Least Squares

The second step after filtering is smoothing the point cloud using MLS method. The MLS is used to smooth and resample the noisy data. The filters applied before were not completely efficient in removing the irregularities in our point cloud as some of these irregularities are very hard to be removed using SOR. In addition to removing the irregularities, MLS can help in creating complete models by resampling the data which recreate the missing parts of the surface in our point cloud by using higher order polynomial interpolations between the surrounding data points.

C++ code of Moving Least Squares is listed below

Listing 5.8: Moving Least Squares

```
std :: cerr << ">> Smoothing with MLS . . . " ;
//Output has the PointNormal type in order to store the normals calculated by MLS
pcl :: PointCloud<pcl :: PointNormal >::
Ptr mlspoints (new pcl :: PointCloud<pcl :: PointNormal >() ) ;
// Init object. Second point type is for the normals
pcl :: MovingLeastSquares<pcl :: PointXYZ , pcl :: PointNormal> mls ;
mls.setInputCloud(cloud_outliers);
```

(a) Before MLS **(b)** After MLS

Figure 5.6: MLS effect

```
mls.setComputeNormals (true) ;
mls.setPolynomialFit (true) ;
mls.setPolynomialOrder(2);
mls.setSearchMethod (tree) ;
mls.setSearchRadius (0.03) ;
mls.setSqrGaussParam(0.0009);
mls.process(*mlspoints);
std :: cerr << "Done . n n" ;
```

As a conclusion, resampling using 'double walls' can fill the missing parts in our point cloud and then the artifacts caused by registering multiple scans together, so the resultant point cloud will be a smoothed one. The figure 5.6 shows a point cloud before and after MLS smoothing.

5.4.2 Laplacian Smoothing

In our project we apply two smoothing methods, the first one is MLS which we mentioned above to smooth and resample the 3D point cloud. The second smoothing method is the LS which we apply on the mesh created by greedy triangulation reconstruction. In LS, first, all the vertices are being searched and then all the vertices are connected to the searched ones are being found as well. These information are stored in an array which is being updated with time. For the corner vertices, normally no smoothing is applied, but for the edge vertices, they are smoothed only along the edge, in addition, they are smoothed if the angle of the edge is smaller than a specific threshold. For all the other vertices, their new locations are being set by taking the average of the location of the other vertices connected to them.

Listing 5.9: Laplacian Smoothing

```
std :: cerr << "Begin Laplacian Smoothing . . . " ;
```

```
pcl :: PolygonMesh output ;
pcl :: MeshSmoothingLaplacianVTK vtk ;
vtk.setInputMesh ( triangles ) ;
vtk.setNumIter (20000) ;
vtk.setConvergence ( 0.0002 ) ;
vtk.setRelaxationFactor ( 0.0002 ) ;
vtk.setFeatureEdgeSmoothing ( true ) ;
vtk.setFeatureAngle (M_PI/5) ;
vtk.setBoundarySmoothing ( true ) ;
vtk.process ( output_laplacian ) ;
std :: cerr << "Done . " << std :: endl ;
```

Figure 5.7 [4] shows a ball mesh before and after LS.

Figure 5.7: Before and after applying LS

5.5 Mesh reconstruction

After the smoothing by MLS, a series of algorithms were performed in order to convert the original input point cloud data into a constructed polygon mesh. The algorithms implemented in this work where three:

- Greedy Triangulation
- Grid Projection
- Poisson

[4] Figure 5.7 was contributed in https://www.researchgate.net/figure/220677870_fig9_Fig-9-Example-of-smart-Laplacian-smoothing-from-the-MESQUITE-toolkit-implemented-in

5.5.1 Greedy Triangulation

GT is an algorithm that receives a point of clouds and converts it into a triangle mesh by projecting the local neighborhoods of a point along the point's normal and connecting unconnected points. We perform GT to create a triangular mesh from the smoothed point of clouds.

The focus of GT is to keep a list of possible points that can be connected to create a mesh. GT is obtained by adding short compatible edges between points of the cloud, where these edges cannot cross previously formed edges between other points GT can, however, over-represent a mesh with triangles when these are represented by planar points. GT contains accurate and computationally efficient triangulations when the points are planar. However, a big difference can be noticed when the triangulations are non-planar, this can be fixed by increasing the number of boundary vertices or linearly interpolating between the boundary vertices [11].

An approach at computing the GT is to compute all distances, sort them and examine each pair of points in length and compatibility with edges created. The Greedy Triangulation algorithm is simple at its core but also not very reliable. With a point cloud, S, of n points, the algorithm looks for the closest point where a compatible edge can be created between the two. A compatible edge can be described as an edge between two points that does not intersect with any other edge [1].

In the Point Cloud Library, greedy projection triangulation works locally, however it allows the specification of the required features to be focused on. Such parameters are neighbourhood size of the search, the search radius and the angle between surfaces [5].

Listing 5.10: Greedy Triangulation

```
std :: cerr << "Grid Projection..." ;
pcl :: PolygonMesh grid;
pcl :: GridProjection <pcl :: PointNormal> gp;
gp.setInputCloud (mlspoints);
cout<<"Grid Projection";
gp.setSearchMethod(tree2);
gp.setResolution( 0.0025 );
gp.setPaddingSize (3);
gp.reconstruct(grid);
std :: cerr << "Grid Projection Done";
```

The output of GT of our program can be seen in figure 5.9.

Figure 5.8: Greedy Triangulation performed on the Stanford bunny

5.5.2 Grid Projection

The second mesh reconstruction algorithm implemented in this work was GP. GP tries to create seamless surfaces that lack boundaries or proper orientation. One of the defining points for GP is that a continuous surface is created, as opposed to Greedy Triangulation, where the surface is not continuous and can contain holes[10]. We perform Greedy Projection to create a rectangular mesh, without holes, from the smoothed point of clouds.

The GP algorithm is a grid based surface reconstruction algorithm. Points are first partitioned into voxels, and a vector field is constructed, where the vectors at any given point are directed at the nearest point. A surface is then determined by examining where vectors with opposite directions point towards. Edges in the voxels that this surface are reconstructed from are determined, and padding cells (cells neighboring the voxels containing the points) are also added. The center points of each voxels are then projected based on the edge intersections, and the surface is reconstructed by connecting these center points [9].

Listing 5.11: Grid Projection

```
boost :: shared_ptr<pcl :: PolygonMesh> triangles (new pcl :: PolygonMesh );
std :: cerr << "GreedyProjectionTriangulation..." ;
pcl :: GreedyProjectionTriangulation<pcl :: PointNormal> gt;
gt.setInputCloud(mlspoints);
gt.setSearchMethod ( tree2 ) ;
gt.setSearchRadius (0.025) ; //R
```

(a) Before GT

(b) After GT

Figure 5.9: Before and after apply GT

```
gt.setMu (5) ; //mu
gt.setMaximumNearestNeighbors (50) ;//D
gt.setMaximumSurfaceAngle (M_PI/4) ; // 45 degrees
gt.setMinimumAngle (M_PI/18) ; // 10 degrees
gt.setMaximumAngle(2*M_PI/3);//120d
gt.setNormalConsistency ( false ) ;
gt.reconstruct(*triangles);
std :: cerr << "Done" ;
```

Figure 5.10: Grid Projection performed on the Stanford bunny

The method requires only two parameters: (1) padding and (2) resolution.

The output of GT of our program can be seen in figure 5.11.

(a) Before GP **(b)** After GP

Figure 5.11: Before and after apply GT

5.5.3 Poisson

Poisson is a mesh reconstruction from oriented points which can be casted as a spatial Poisson problem [7]. We perform the Poisson algorithm to obtain a watertight mesh from a set of smoothed and filtered point of clouds.

The Poisson reconstruction algorithm reconstructs a watertight model by using the indicator function and extracting the iso-surface. The indicator function is a constant function, obtained from the sampled data points, whose computation would result in a vector with unbounded values.

The surface integral of the normal field is approximated using a summation of the point samples and then reconstructed the indicator function using its gradient field as a Poisson problem [6]. The gradient field is used to reconstruct the indicator function of the surface.

Figure 5.12: General Overview of the Poisson method

The Poisson reconstruction uses the indicator function that works best with noisy data points and can therefore recover fine details. [6].

Figure 5.13: Poisson reconstruction performed on the Stanford bunny

The biggest problem with Poisson and the reason why it is not usable for this project, is the fact that it requires a clean mesh, without noise to actually reconstruct the object of interest. Therefore, in order to use Poisson, a lot of manual clean up should be done before trying to actually apply the algorithm. Moreover, there is only one parameter that can be changed, which is the depth of the octree. Making it even more difficult to obtain a good reconstruction if the input point cloud is noisy. Figure 5.14 includes an example of Poisson reconstruction using our data with a depth of 10.

Listing 5.12: Poisson

```
pcl :: Poisson <pcl::PointNormal> poisson;
poisson.setDepth(10);
poisson.setInputCloud(mlspoints);
boost ::shared_ptr<pcl::PolygonMesh> triangles2(new pcl::PolygonMesh);
poisson.reconstruct(*triangles2);
```

5.6 Holes Filling

During the work on the project we faced a problem of Holes Filling in the final mesh. The origins of these holes ranged from early scanning issues, where some parts of the body can't be scanned properly, to later stages, where filtering and meshing algorithms can distort the surface of the model and create additional hole issues.

In case of usage of Grid Projection amount of holes drastically decreases, but at the same time

Figure 5.14: Poisson reconstruction with depth=10

orientation of the squares, which are created by the algorithm (i.e. angles and orientations) can create small holes between a tiny number of squares, ranging from one. This can be fixed by usage of the built-in VTK library – **vtkFillHolesFilter**, which can fill holes with following types of edges:

1. Boundary (used by one polygon) or a line cell;
2. Non-manifold (used by three or more polygons);
3. Feature edges (dihedral angle > than a threshold);
4. Manifold edges (edges used by exactly two polygons).

But, nevertheless, this library is rendered useless for big holes. Most of the time, hole issue occur nearby and inside the seams and joints, which can be clearly seen in the figures 5.15, 5.16 and 5.17 below.

Therefore, apart from the hole detection which can be seen on the pictures, we propose a pipeline approach, which sums up to extraction of the detected edges and filling them with created patches, which can be passed through vtkTriangleFilter in case of usage of the Grid Projection method. This pipeline approach potentially solves the problems of the vtkFillHolesFilter, closing the big holes on the feet and head. It can be a suitable direction for the development of our project.

Pipeline approach comprises four steps:

1. Extraction of the edges (vtkFeatureEdges, vtkExtractEdges)

Figure 5.15: Holes Detection

Figure 5.16: Holes Detection

2. Create patches (vtkStripper)
3. Use contours as an input and fill holes with created patches (vtkTriangleFilter)
4. Append changes to the model

Figure 5.17: Holes Detection

Chapter 6

User Guide

6.1 Home Screen Window

Figure 6.1: Home Window

- To View Recent Projects

  1. Click **Recent Projects**
  2. This will take you to the Recent project window

- To Start New Project

  1. Click **Start New Project**
  2. This will take you to the Calibration Window
  3. Visualizer will appear

6.2 Calibration Frame Window

- To Set Pass through Filter Parameters

  1. To adjust X minimum value, move slider from left to right
  2. To adjust X maximum value, move slider from left to right or vice versa
  3. To adjust Y minimum value, move slider from left to right
  4. To adjust Y maximum value, move slider from left to right or vice versa
  5. To adjust Z minimum value, move slider from left to right
  6. To adjust Z maximum value, move slider from left to right or vice versa

- To Start Recording the Frames

  1. Enter the Name of person in **Enter Name** field
  2. Click **Record** button
  3. After clicking Record button, **Status** will change to *scanning*

- To Proceed with ICP, 3D Reconstruction & Filtering

  1. Click **Next** button
  2. This will display Registration and Filtering Window

6.3 Registration and Filtering Window

- To initialize ICP/Registration

  1. Enter Name in **Name of File** field. Note that this name should match the name entered in the Calibration Frame Window
  2. Click **Registration** to start ICP

  - Status at bottom of the window changes to Waiting
  - Once the Registration is completed, **Status** changes to *Registration completed*
  - Results are displayed on the Viewer, located on left

- For Filtering and Surface Reconstruction

  1. Select either Grid Projection or Greedy Projection Triangulation
  2. Set the Parameter of Statistical Outliers Removal
     - Set the Value of Mean (By default this value is set at 50)
     - Set the Value of Standard Deviation STD (By default this value is set to 1.50)
  3. Click **3D Reconstruction and Filter** to start 3D Reconstruction and filtering
  4. **Status** will change to *Filtering Started*

- To Go back to Previous Window

  1. Click **Home** button on the top left
  2. This will display Home Screen Window

6.4 Previously Scanned Files

- To Open Mesh File

  1. Click **Open Mesh File** to select/view the previously saved mesh file from the pop up dialog.
     a. Result will be displayed on the viewer on left
  2. Click **Open PCD File** to select/view the previously saved PCD file from the pop up dialog. Result will be displayed on the viewer on left

- To Go back to Previous Window

  1. Click **Home** button on the top left
  2. This will display Home Screen Window

Figure 6.2: Calibration Frame Window

Figure 6.3: Calibration Frame Window

Figure 6.4: 4 Previously Scanned Files

Chapter 7

Results and Conclusion

7.1 Results

To perform our final scans from the Kinect, the turntable was set to turn every 1.8 seconds, with an increment of approximately 10 degrees. After a full 360 rotation is completed the turntable stops and the Kinect stops taking scans. In total around 37 point clouds are generated from this process.

An important remark to be made is that, although the software achieved its purpose of scanning and performing 3D reconstruction of a person, the final 3D model was lacking some fine details. This was due to the fact that during the scanning, the Kinect was positioned at least one meter away from the person, in order to perform a full body scan. By increasing the distance between the Kinect and the person, a lot of information was lost and when the time to performed the reconstruction came, only a low percentage of data was left to work with.

To remove noise and unwanted data a real time pass through filter is applied in order to delimit the region of scanning, ignoring the background and focusing only on the person to be scanned. After that, an outliers removal is performed with a standard deviation value (STD) of 0.5.

To merge all the scans into one single point cloud, the ICP algorithm is performed.

Figure 7.1: Merging scans from the Kinect

After obtaining the reconstructed 3D point cloud, the MLS algorithm is applied in order to make the surface smoother.

Once the surface is smoothed, the mesh is extracted. We applied two different methods: GT and GP. This is the result of applying GT with $mu = 5, D = 50$ and $R = 0.025$.

Figure 7.2: Greedy Triangulation reconstruction with mu=5, D=50, R=0.025

The advantages of GT is that is fast to compute, and it has multiple parameters that can be tuned in order to change the final output. The disadvantages are that it contains holes and thus, needs further smoothing. To close the holes and smooth the surface, Laplacian smoothing was performed after the GT.

The second reconstruction technique was GP. It is a method that requires more computational time than GT but gives as output a rectangular mesh without holes, requiring no smoothing after. This are the results of GP with a resolution (res) of 0.0025.

Figure 7.3: Grid Projection recontruction with res=0.0025

Another method we tried to implement was Poisson. It is a method that gives a watertight mesh with fine details but is slower than GP and GT and also amplifies the noise, requiring to do a lot of previous cleanup of the mesh before trying to implement it. This is the reason why the Poisson method was discarded for the final implementation of our software.

7.2 Conclusions

The designed software can successfully scan a person using a low cost sensor, reconstruct a 3D model from the scans and export it for 3D printing. During the implementation of this project, a thorough research was performed in order to narrow down and find the final materials and methodologies which gave us the most optimal results. First of all, we had to decide which library to utilize. For this project the PCL was chosen as the best option for achieving our goals. Second, as multiple research have been performed with Kinect and PCL, the Kinect v2.0 was selected as the ideal sensor for the project. Third, for cleaning and filtering the data, a real time pass through filter and an outliers removal were performed. Fourth, for combining the consecutive scans into a single point cloud, the ICP algorithm was performed. Fifth, for obtaining a better shaped and evenly distributed mesh, MLS and Laplacian were used as smoothing techniques. Finally, for reconstructing the 3D model two algorithms were implemented: Grid Projection and Greedy Triangulation.

An improvement that can be made for this project, is to perform the scan by positioning the Kinect

closer to the person. In this way, the reconstructed model could have more details and be more accurate to the real life.

Although we encountered some problems and technical difficulties during the implementation of this project, like the incompatibility of the Kinect with MacOS or the installation of PCL to make it work with Qt, overall this project allowed each of the team members to improve their weaknesses, utilize their strengths and share it with others so as to obtain the best result possible.

Bibliography

[1] Matthew T Dickerson, Robert L Scot Drysdale, Scott A McElfresh, and Emo Welzl. Fast greedy triangulation algorithms. In *Proceedings of the tenth annual symposium on Computational geometry*, pages 211–220. ACM, 1994.

[2] Peter Flynn. Formatting information: A beginner's guide to latex (5th online ed.). *Cork: Silmaril*, page 193, 2002.

[3] H Gonzalez-Jorge, B Riveiro, E Vazquez-Fernandez, J Martínez-Sánchez, and P Arias. Metrological evaluation of microsoft kinect and asus xtion sensors. *Measurement*, 46(6):1800–1806, 2013.

[4] David C Hay. *Requirements analysis: From business views to architecture*. Prentice Hall Professional, 2003.

[5] Juha Hyvärinen et al. Surface reconstruction of point clouds captured with microsoft kinect. 2012.

[6] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (TOG)*, 32(3):29, 2013.

[7] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006.

[8] David Kirk, Abigail Sellen, Stuart Taylor, Nicolas Villar, and Shahram Izadi. Putting the physical into the digital: issues in designing hybrid interactive surfaces. In *Proceedings of the 23rd British HCI Group Annual Conference on People and Computers: Celebrating People and Technology*, pages 35–44. British Computer Society, 2009.

[9] Thuong Le-Tien, Marie Luong, Thai Phu Ho, and Viet Dai Tran. 3d reconstruction using kinect sensor and parallel processing on 3d graphics processing unit. *REV Journal on Electronics and Communications*, 3(1-2), 2013.

[10] Ruosi Li, Lu Liu, Ly Phan, Sasakthi Abeysinghe, Cindy Grimm, and Tao Ju. Polygonizing extremal surfaces with manifold guarantees. In *Proceedings of the 14th ACM Symposium on Solid and Physical Modeling*, pages 189–194. ACM, 2010.

[11] Lingni Ma, Thomas Whelan, Egor Bondarev, Peter HN de With, and John McDonald. Planar

simplification and texturing of dense point cloud maps. In *Mobile Robots (ECMR), 2013 European Conference on*, pages 164–171. IEEE, 2013.

[12] Niloy J Mitra, Natasha Gelfand, Helmut Pottmann, and Leonidas Guibas. Registration of point cloud data from a geometric optimization perspective. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 22–31. ACM, 2004.

[13] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145–152. IEEE, 2001.

[14] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[15] Jürgen Sturm, Erik Bylow, Fredrik Kahl, and Daniel Cremers. Copyme3d: Scanning and printing persons in 3d. In *German Conference on Pattern Recognition*, pages 405–414. Springer, 2013.

[16] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[17] Song Tiangang, Lyu Zhou, Ding Xinyang, and Wan Yi. 3d surface reconstruction based on kinect sensor. *International Journal of Computer Theory and Engineering*, 5(3):567, 2013.