3D Profile Digitizer

By

Abdul Rehman Anwer	2011-NUST-SEECS-BE-EE-004
Fahad Mubeen	2011-NUST-SEECS-BE-EE-033
Rizwan Ahmad Bhatti	2011-NUST-SEECS-BE-EE-100
Sohaib Ishtiaq Malik	2011-NUST-SEECS-BE-EE-111

A Project report submitted in partial fulfillment

Of the requirement for the degree of

Bachelors in Electrical (Electronics) Engineering

Department of Electrical Engineering

School of Electrical Engineering & Computer Science

National University of Sciences & Technology

Islamabad, Pakistan

2015

CERTIFICATE

It is certified that the contents and form of thesis entitled **"3D Profile Digitizer"** submitted by *Abdul Rehman Anwer* **(2011-NUST-SEECS-BE-EE-004),** *Fahad Mubeen* **(2011-NUST-SEECS-BE-EE-033),** *Rizwan Ahmad Bhatti* **(2011-NUST-SEECS-BE-EE-100)**and *Sohaib Ishtiaq Malik* **(2011-NUST-SEECS-BE-EE-111)** have been found satisfactory for the requirement of the degree.

Advisor: _____

(Dr. Rehan Hafiz)

Co-Advisor: _____

(Dr. Khawar Khurshid)

Co-Advisor: _____

(Dr. Hammad Cheema)

ABSTRACT

Quality Control is one of the most important stages of a manufacturing process in which the manufactured goods are typically compared to a reference for matching its dimensions, texture, color and other features. Traditional quality inspection techniques involve use of a Coordinate Measuring Machine (CMM).This requires contact with the object under inspection. As it is compulsory for the object to come in direct contact with CMM, it is therefore unsuitable for delicate, complex and freeform shaped objects. Moreover techniques involving CMM are slow (frequency is few hundred Hz) and may modify or damage the object being tested. These techniques also do not provide information about color and texture of object which may be an important aspect in quality control.

We have developed a quality inspection tool which is much faster (frequency greater than 1 kHz) and contactless. Using 3D laser scanning, laser triangulation and image processing a Color mapped 3D model of an object can be created. Accompanied with it is a cross platform application with an intuitive GUI, which will enable 3D model extraction, with accuracy approaching 99%, for fast computerized analysis without harming the object.

3D laser scanning can also be used in production of custom braces, prosthetics, game development, conservation of historical artifacts etc.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

DEDICATION

To Almighty Allah who gave us the opportunity

&

To our parents for the selfless love and motivation

&

To the advancement of technology in Pakistan

ACKNOWLEDGEMENTS

We would like to thank the people of VISPRO lab, who helped us throughout our project. We would like to thank Advisor Dr. Rehan Hafiz for the guidance and motivation in the project. We would also like to thank our Co-Advisors, Dr. Khawar Khurshid and Dr. Hammad Cheema for their support.

We are extremely thankful to the administration of SEECS and NUST for encouraging such projects. Without such encouragement we would not have been able to accomplish this project.

Chapter 1

INTRODUCTION

1.1 WHAT ARE 3D SCANNERS/DIGITIZERS

3D scanners or digitizers are used to convert real life objects into digital 3D models. These 3D models can then be used in many applications. Some applications of 3D scanners are –

i. *Rapid Prototyping* – 3D Scanners can be used to scan hand created models, which can then be modified digitally. Such models can also be copied by printing using a 3D printer. Scanned models can also be used to mill dies and cavities for mass production of parts.

ii. *Orthotics and Prosthetics* – 3D scanners can scan human body parts, which require braces or prosthetics. These scanned parts can then be used to design custom prosthetics for the patient.

iii. *Quality Control* – 3D Scanners can scan manufactured parts, and the manufactured part can be compared with the actual design for quality control purposes.

iv. *Game and Movie Character Design* – 3D scanners can scan clay models made by artists and convert them directly into 3D models, which can be modified to be used in games and movies.

Figure 1 - 3D Scan Of A Toy[1]

1.2 TYPES OF 3D SCANNERS

3D scanners are of contact and contactless type. There are many types of 3D scanners employing different technologies. The main types of technologies that 3D scanners use are –

i. *Time of Flight* – Time of Flight (TOF) based 3D scannersare contactless scanners that use complex laser sources and sensors to calculate the time taken by light from a laser source to the sensor in order to calculate depth. Time of flight based 3D scanners are very expensive due to the equipment involved.

ii. *Triangulation* – Triangulation based 3D scanners use a laser and a camera. In this technology, the laser line being shined on the object gets distorted in the view of the camera with respect to the distance of the object from the camera, the depth is then calculated using triangulation. Triangulation based scanners tend to be the lowest cost of all 3D scanners.

iii. *Structured Lighting* – Structured lighting based scanners use a projector to project vertical lines of black and white patterns which are acquired using a camera. These are then processed by image processing in order to obtain the 3D model. Structured lighting based scanners tend to be expensive due to the projector involved.

iv. *Light Modulation techniques* – In this form of 3D scanning, the intensity of light being shined on an object is modulated by sine function. Depending on the depth of the object, a phase change occurs. In the acquisition process, the phase is translated into distance, which is then used to calculate the 3D model. Light modulation based scanners are also very expensive.

1.3 COMPARISON OF DIFFERENT TYPES OF 3D SCANNING METHODS

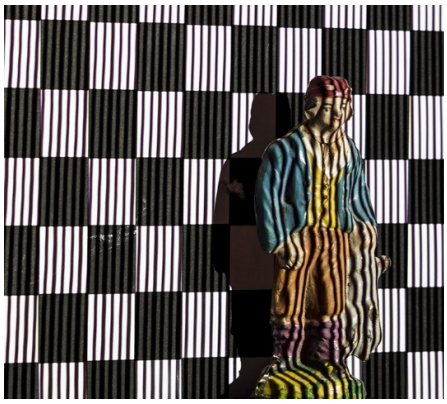


Figure 2 - Structured Lighting Technique[2]

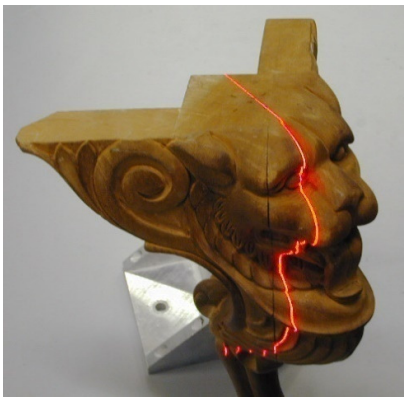


Figure 3 - Laser Scanning Technique[3]

The above mentioned types of 3D scanning methods were considered for the purpose of our Final Year Project. Of these techniques, TOF and modulation based equipment were very expensive, starting from few thousands of dollars for even the most basic components. Moreover, they were neither locally available, nor were they easy to import, thus these techniqueswere not pursued further. The remaining 2 techniques were then analyzed for their strengths, weaknesses, opportunities and threats.

	Strengths	Weaknesses	Opportunities	Threats
Laser Triangulation	Captures Very Fine detail Lower Part Cost Parts availability Small size of parts	Takes longer time for acquisition	Low Cost 3D Scanners are in high demand Future product development opportunity	High dependence on camera being used.
Structured Lighting	Fast projector response results in faster acquisition	Requires very bright DLP (Digital Light Processing) projector which is both expensive and hard to find. Projector can be damaged easily Huge Size of Projector	Possible innovation on technique involved	Unavailability of DLP projectors that are bright enough for scanning.

Table 1 - SWOT analysis of 3D scanning techniques

After the SWOT analysis was done it was decided that we would use the Laser Triangulation method of 3D scanning, due to parts availability, low cost and high demand.

1.4 PROBLEM STATEMENT

"Design and Implement a 3D scanner that uses a laser line and camera. It should be able to scan objects having a diameter between 50mm and 120mm, and a height between 50mm and 120mm. The scanner should be accurate within 5mm. It should be able to complete a scan in under 30 minutes"

The project consists of 4 main areas –

i. *Hardware*– The design and prototyping of the overall structure of the scanner, its design and prototyping.

ii. *Embedded*– The circuit design, as well as interfacing with electronics such as motor, laser, light source etc. using a microcontroller.

iii. *Image Processing* – Image Processing is the backbone of this project, and is used to convert the raw images acquired from a camera into the final 3D model.

iv. *Software* – Covers all programming that is done on the PC side in order to interface with the hardware, provide a GUI and enable seamless integration.

1.5 MOTIVATION

Our group wanted to do a project that would utilize the skills and knowledge we have learnt over the past 4 years. This project covers all basic aspects of Electrical Engineering. It utilizes key areas such as hardware design, software design, embedded electronics, image processing and engineering mathematics. We wanted to pursue a project that would be innovative. We felt that 3D scanning was an area that had lots left to be discovered and looked at it as a challenge. We believe that after the boom of 3D printers, 3D scanning is working towards becoming a mainstream tool and will be used in all types of industries in the future.

1.6 APPROACH METHODOLOGY

We employed the product development cycle in our project, as follows –

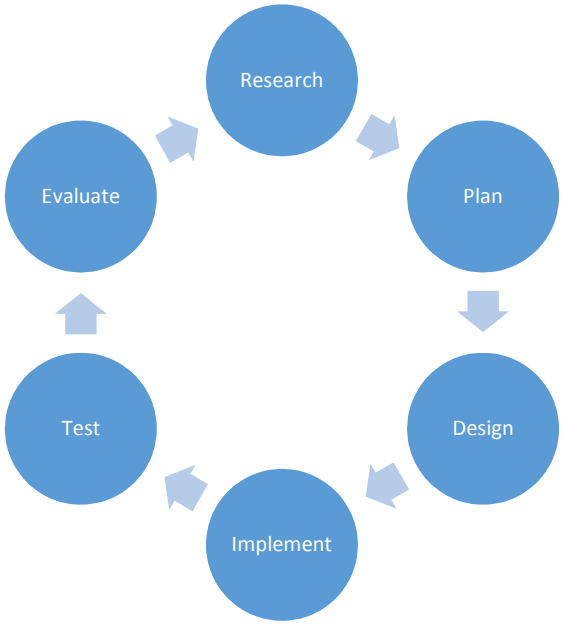


Figure 4 - Approach Methodology

These steps are discussed in details later on in this report.

Chapter 2

LITERATURE REVIEW

2.1 LASER LINE SEGMENTATION/EXTRACTION

Laser line extraction is the process in which the laser line is separated from the background after acquisition.The images are acquired in the dark, with the laser turned on. This gives us a relatively high contrast between the laser light and the dark background. We needed an algorithm that would give us the brightest part of the laser line, such that for a vertically placed laser line in an image, it would give us the pixel through which center of the laser line passes in each row.

2.1.1 Methods

Multiple methods were tested for laser line extraction. The methods tested were –

i. Binary Threshold
ii. Maximum Value
iii. First order difference method
iv. Canny edge
v. Inverted 2nd order Gaussian filter

Of these methods, Binary Threshold, Maximum Value and Inverted 2nd Order Gaussian Filter method were shortlisted after some preliminary testing. Their results are given below.

2.1.2 Results

The image in Figure 5 was used as a control to test all the algorithms of laser line segmentation. After a laser line was extracted, a qualitative judgement was made to see which algorithm passed the detected line from the centermost part of the laser. The camera sensor often gets salt noise at random pixel locations, which caused inaccurate detection in some of these algorithms.

Figure 5 - Red Laser Line against a dark background

2.1.2.1 Binary Threshold

Binary threshold segmentation is a basic algorithm in Digital Image Processing, where values of pixels above a certain threshold are given a value of 1, whereas all other pixels are given a value of 0. In our testing we used global threshold. After a binary image is obtained, the center point of the white pixels in each row was calculate, this represents the center of the laser. The results of this type algorithm is as follows (Detected laser line is shown in blue color) –

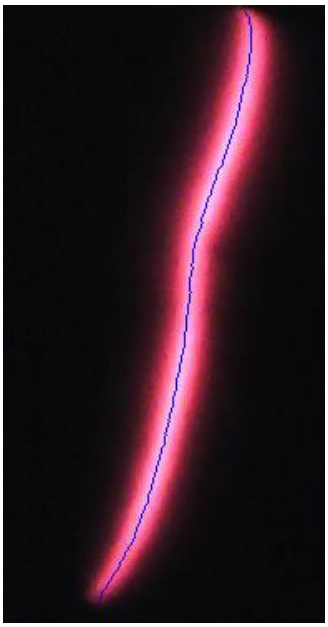


Figure 6 - Binary Threshold, Without Salt Noise

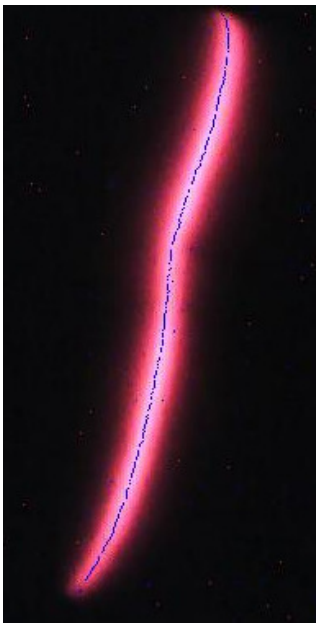


Figure 7 - Binary Threshold, With Salt Noise

It can be seen from Figure 6Figure 6 that in case of no noise, the algorithm gives us acceptable results, but when even a little salt noise is present, as shown in Figure 7 false laser line detection takes place, resulting in breaks in the detected laser line.

2.1.2.2 Maximum Value

In this method, the pixel with the maximum value in each row is said to be the center of the laser line. The results of this algorithm is shown below –

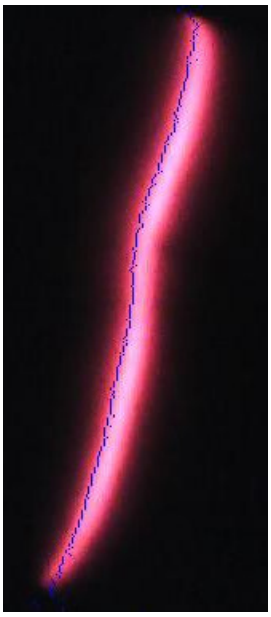


Figure 8 - Max Value Segmentation, Without Noise

As shown in Figure 8 the result of this algorithm is not well, and is offset from the center of the line, and is not smooth. This algorithm was not working well even without noise, so further testing was not done with noise.

2.1.2.3 Inverted 2nd order Gaussian Filter

This method was implemented using the research paper "Laser Line Detection with Sub-Pixel Accuracy"[4]. The results of this algorithm are shown as follows –

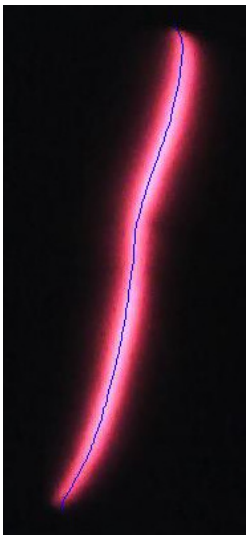


Figure 9 - Inverted 2nd Order Gaussian Filter - Without Noise

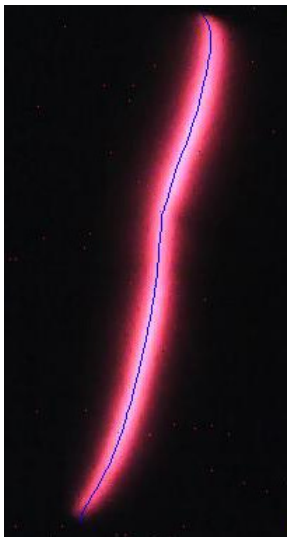


Figure 10 - Inverted 2nd Order Gaussian Filter - With Noise

It can be seen that from the Figure 9 and Figure 10 that this algorithm works best out of all of the above. This method was finalized for our implementation.

2.1.3 Other Methods

Other methods were not tested any further either due to false detection or due to long processing time. One research paper that we tested *"Detection of Laser Beam's Center-line in 2D Images"*[5]provided sufficient laser line detection but took a very long processing time. The algorithm outlined in the paper takes more than 20 seconds to process a single image of 640 x 480 pixels. At this speed, processing 200 images would take 4000 seconds, meaning a time period of more than 60 minutes. Since this exceeded the time that we initially proposed in our problem statement, this algorithm was not tested further.

2.2 DEPTH MAP CALCULATION

This process converts the extracted laser into a 2D depth slice. These depth slices are then used to calculate the point cloud. Multiple algorithms were tested for the purpose of depth calculation from the extracted laser as follows –

i. *"Laser Triangulation for 3D Profiling of a Target"*[6]

ii. *"A Simple Surface Mapping Technique using Laser Triangulation Method"* [7]

iii. Surface mapping using horizontal and vertical field of view.

A discussion of these methods is as follows –

2.2.1 Laser Triangulation for 3D Profiling of a Target

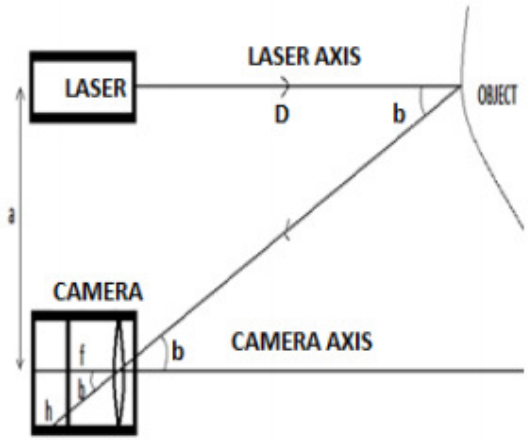


Figure 11 - Laser Triangulation For 3D Profiling Of A Target[6]

This type of method uses the above configuration of laser and camera. Using Trigonometry the following solution is obtained [6] –

$$\tan b = \frac{h}{f}$$

and

$$\tan b = \frac{a}{D}$$

resulting in: $D = \frac{axf}{h}$

This step is done for one row at a time, resulting in a distance for a single pixel per row. This step is repeated for each row, resulting in a single slice. The object is then rotated for multiple "slices" until the complete 3D point cloud is obtained.

2.2.2 A Simple Surface Mapping Technique using Laser Triangulation Method

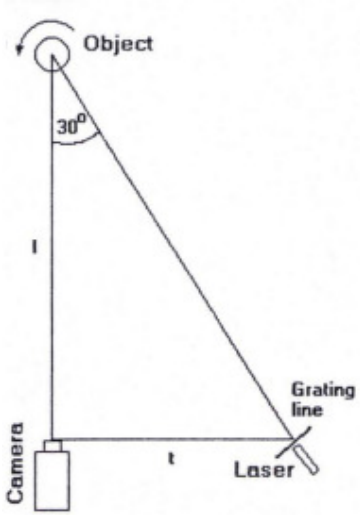


Figure 12 - A Simple Surface Mapping Technique Using Laser Triangulation Method[7]

This type of method uses the above configuration of laser and camera. Using Trigonometry the following solution is obtained [7]–

Here the variables are

i. l = Distance from the camera to the center of rotation of motor shaft (which is also the point where laser line and camera central axis intersect.
ii. t = The distance between the camera and laser line plane (laser line) such that a right angle triangle is formed

Now if l is supposed to be 30cm then t can be found out be trigonometric relationships

$$l = 30cm$$

$$tan30 = \frac{t}{l} = \frac{t}{30}$$

$$t = 30 \times tan30 = 17.3$$

Now consider an enlarged version of the figure –

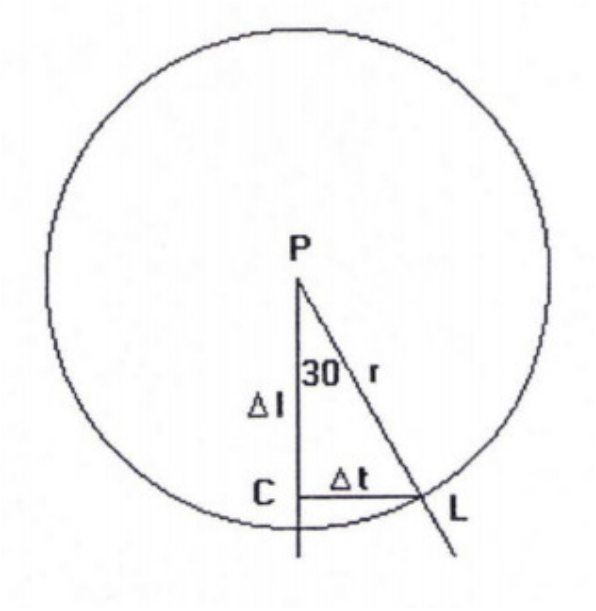


Figure 13 - Simple Surface Mapping Technique - Enlarged Figure[7]

Where

i. L - the point where laser line intersect the object
ii. Δt - is the distance between the camera axis and L
iii. C - is the point such that a perpendicular CL is formed in right angle triangle PCL
iv. Δl - is the base of the triangle PCL

The distance between the point C and L can be found in pixels by using a camera, and if we know the horizontal scaling (i.e. Pixels/mm) then this distance can be converted to mm, and we know 2 angles and a side in right angle triangle PCL. So remaining sides can be known.

$$r = \frac{\Delta t}{\sin(30)}$$

$$\Delta l = r.\cos(30)$$

Now consider the following figure

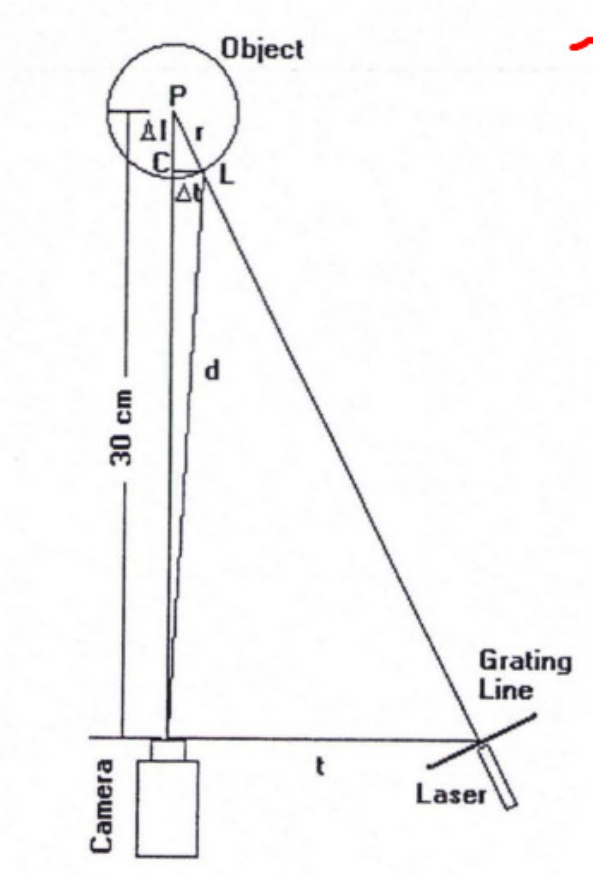


Figure 14 - Simple Surface Mapping Technique - Hypotenuse Calculation[7]

Distance d (hypotenuse) can be found by using the following trigonometric formula

$$d = \sqrt{(\Delta t^2 + (30 - \Delta l)^2)}$$

The 3D point can be found easily, n is the angle of the shaft

$$x = r.\cos(n)$$

$$z = r.\sin(n)$$

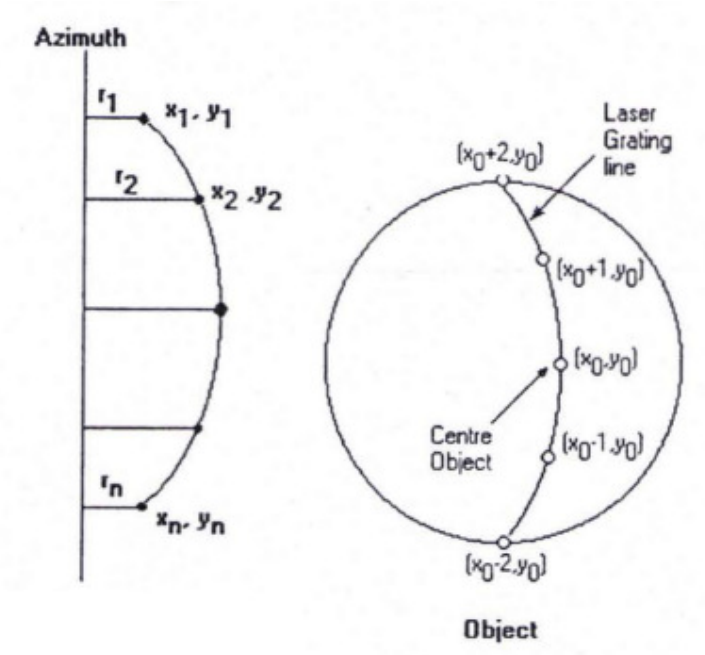


Figure 15 - A Simple Surface Mapping Technique - Point Cloud

And repeating the above process on the whole image we get all the points on 3D slice, and rotating the motor for multiple images we get 3D point cloud, which we can store in 3D matrix for further processing.

2.2.3 Surface mapping using horizontal and vertical field of view

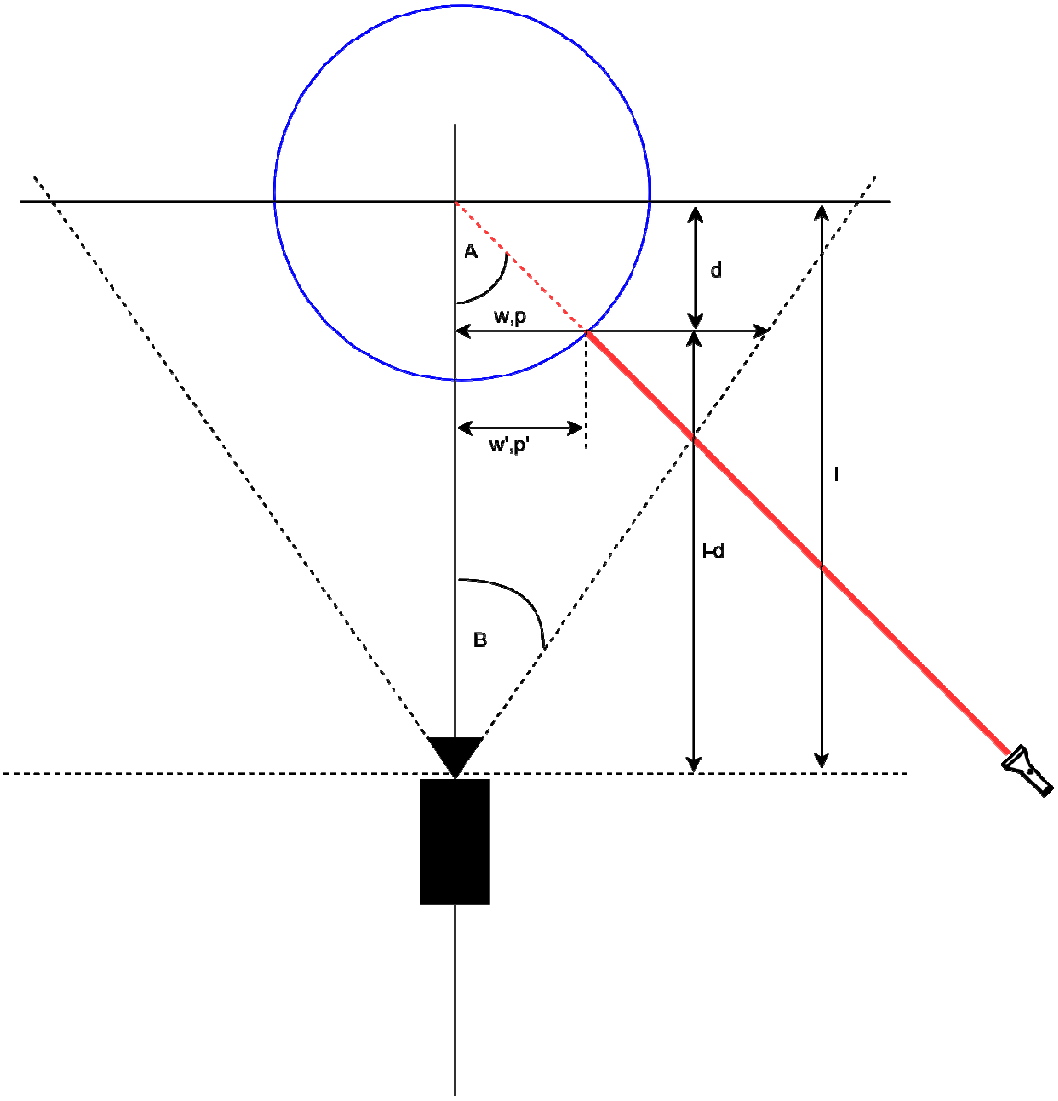


Figure 16 - Surface Mapping Using Horizontal And Vertical Field Of View

This technique uses the configuration above. In this figure the labeled distances and angles are –

i. A = Angle of Laser

ii. B = Half of the Field Of View (FOV) of the Camera

iii. p’ = distance in pixels from center of screen to pixel of laser in image obtained from camera

iv. w’ = actual perpendicular distance in mm from center to laser line

v. l = Perpendicular distance in mm from Camera to center of object

vi. d = Perpendicular distance from center of rotation to point on where object obstructs laser.

vii. p = distance in pixels from center line to the point mentioned in vi.

viii. w= distance in mm from center line to the point mentioned in vi.

Assuming that the image obtained from the camera is not distorted, we can deduce the following using trigonometry –

$$\frac{p'}{w'} = \frac{p}{w}$$

$$\tan B = \frac{p}{d} \rightarrow p = (l - d) \times tanB$$

$$\frac{p'}{w'} = \frac{p}{w} = \frac{p}{(l - d) \times tanB}$$

We know from the figure $\tan A = \frac{p}{d}$

$$p = d \times \tan A$$

$$\frac{p'}{w'} = \frac{p}{(l - d) \times \tan B}$$

$$\frac{p'}{d \times \tan A} = \frac{p}{(l - d) \times \tan B}$$

$$p' \times l \times \tan B = p \times d \times \tan A$$

$$p' \times l \times \tan B = d \times (p \times \tan A + p' \times \tan B)$$

$$d = \frac{p' \times l \times \tan B}{p \times \tan A + p' \tan B}$$

$$d = \frac{p'}{p \times r + p'}$$

2.2.4 Comparison of Depth Calculation Algorithms

Following is a line graph showing comparison between the algorithms. An object was placed at the center of location, and a reading was taken. The object was then moved 2cm closer and another reading was taken. This was repeated 6 times and all the readings were recorded.

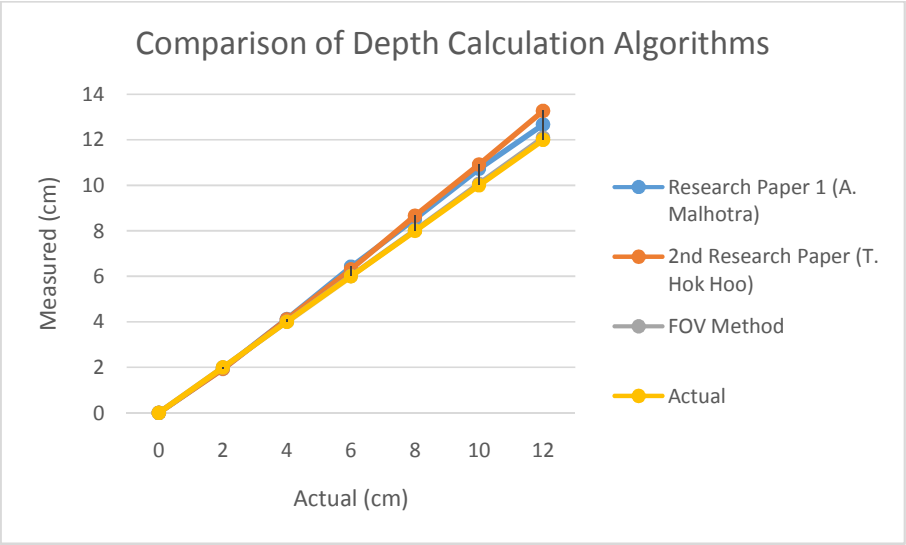


Figure 17 - Comparison Of Depth Calculation Algorithms

A comparison of the error at each distance is given as follows –

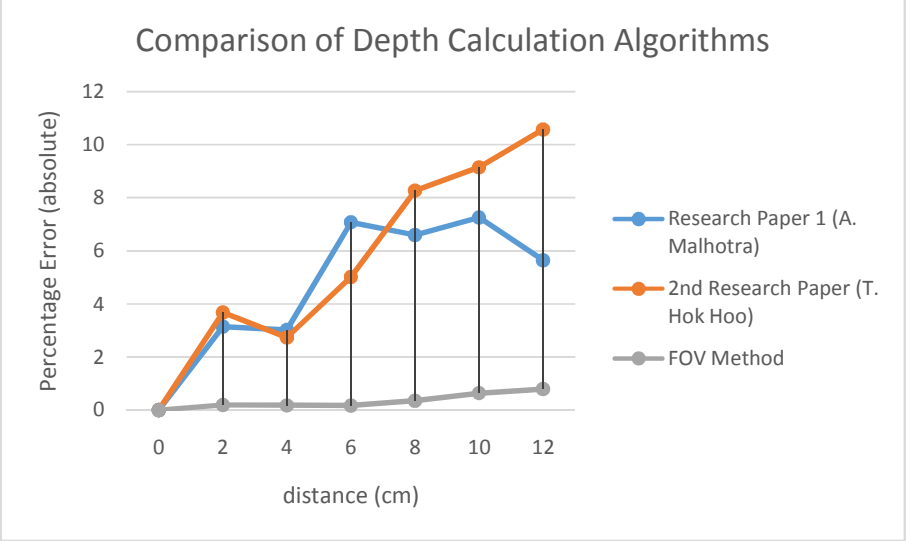


Figure 18 - Comparison Of Depth Calculation Algorithms

From the above graphs, it can be seen that all 3 algorithms work and give reasonable results. However, from our problem statement, it was required that an accuracy of 5mm or better be obtained. The first two algorithms, which we obtained from research papers of [7] and [6]had an accuracy of worse than 5mm at distances greater than 6 and 8 cm respectively. This left us with only one option, to use the third algorithm that we had derived ourselves. It can also be clearly seen from the error graph that the FOV method is most accurate of all of these, and least error prone.

2.3 HEIGHT CALCULATION

Before calculating the height of a point, first its depth h must be determined by the depth finding algorithm

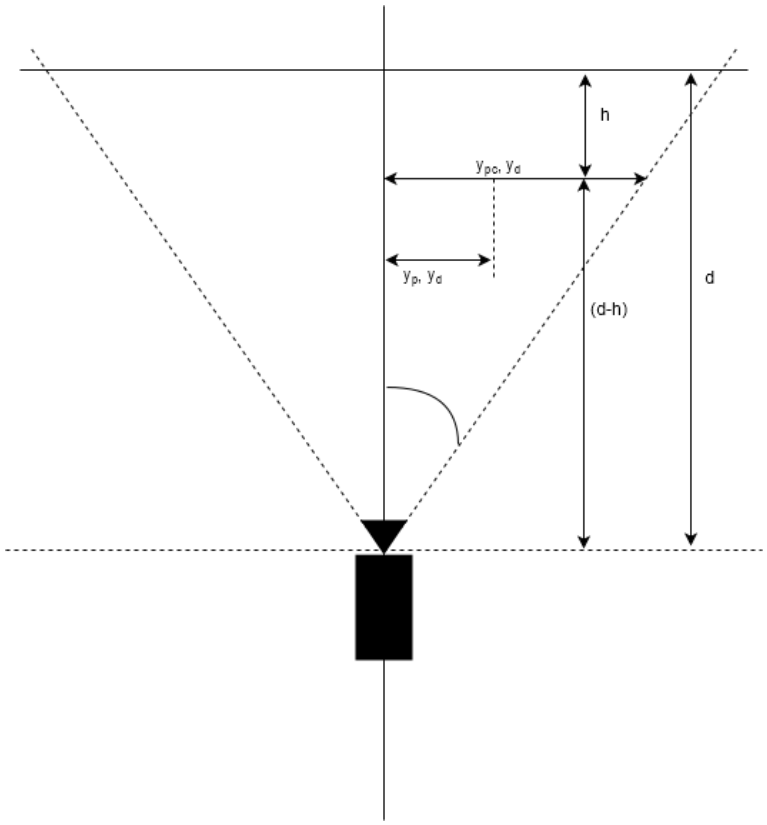


Figure 19 - Height Calculation

The Variables used in the figure

i. $\mathbf{y_{pc}}$= Distance in pixels from camera axis to the far most point in camera view (horizontal field of view)
ii. $\mathbf{y_p'}$ = Distance in pixels from camera axis to Laser line
iii. $\mathbf{y_d}$ = Distance in mm from camera axis to the far most point in camera view (horizontal field of view)
iv. $\mathbf{y_d'}$ = Distance of laser line (mm) from camera axis to laser line
v. **h** = Depth of a point found out by previous algorithm
vi. **d** = total distance from the center of camera to the point where the laser line intersects the camera axis

vii. **gamma** = Half of vertical field of view

Assumptions:

We assume the image is linear i.e. it is not distorted from the edges.

Derivation:

Then the ratio of pixels per millimeter will be same at a certain distance from the camera.

$$\frac{Ypc}{Yd} = \frac{Yp'}{Yd'}$$

Now the actual height of a point y_d' will be

$$Yd' = \frac{Yp' \times Yd}{Ypc} - Equation\ A$$

Using trigonometric formula on right angle triangle OPQ we get

$$\tan(gama) = \frac{Yd}{d-h}$$

This implies that

$$Yd = (d-h)\tan(gama) \quad Equation\ B$$

Putting value of Y_d form equation B in equation A we get the height of the pixel

$$Yd' = \frac{Yp' \times (d-h)\tan(gama)}{Ypc}$$

Chapter 3

FUNCTIONALITY

Our Project w

chart –

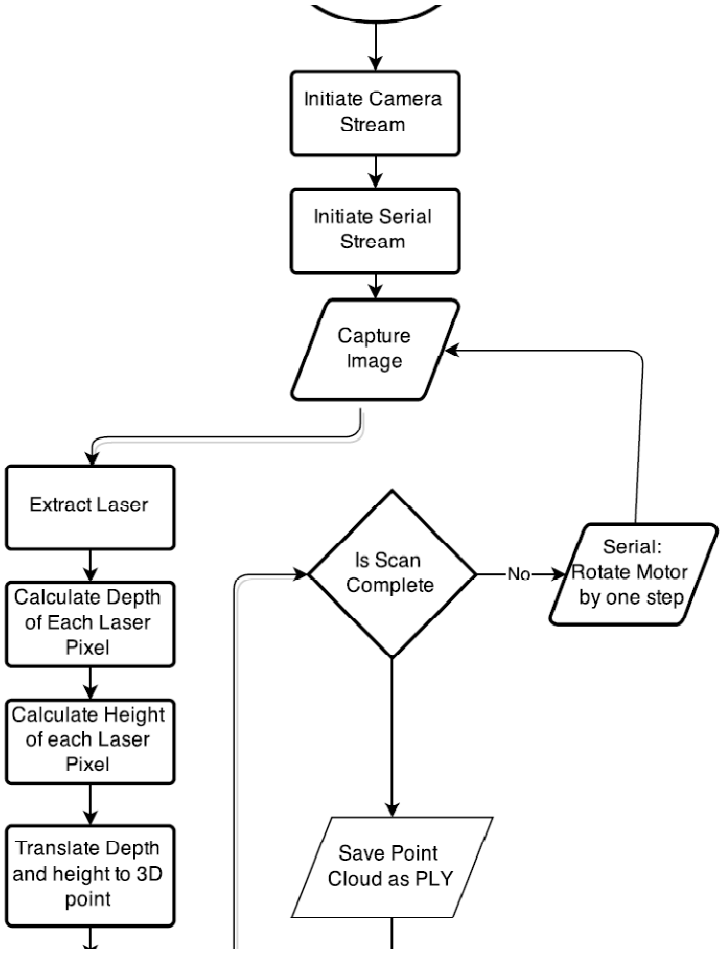


Figure 20 - Flow Chart

The functionality of our project can be divided into multiple areas –

i. Hardware Functionality
ii. Software Functionality
iii. Product Functionality

3.1 HARDWARE FUNCTIONALITY

The Scanner hardware needs to perform the following functions –

i. Be able to rotate the object
ii. Be able to turn the laser on/off
iii. Be able to communicate with the computer
iv. Be able to control lights for Color Scanning, as discussed later in Section 4.1.4
v. The hardware must be linked with the PC side software to be able to perform diagnostics
vi. The user should be able to see through an indicator when power is being supplied to the scanner
vii. The user should be able to switch off the scanner's main power using a switch
viii. The user should be able to halt the motor at any time through software

3.2 SOFTWARE FUNCTIONALITY

i. The software must implement an easy to use GUI and provide a seamless way to interact with the hardware.
ii. The user must be able to initiate a scan in minimum number of steps.
iii. The user must be able to export scans in PLY (Polygon file format) format. PLY format is the standard used for 3D scanning.
iv. The user must also be able to save and reload files.
v. The user must be able to see a live preview of the scan while its happening
vi. The user must have access to diagnostic tools to help diagnose problems that may be happening with the hardware.

3.3 PRODUCT FUNCTIONALITY

i. The Design of the product should stop light from interfering during acquisition process.
ii. The hardware look simple yet elegant.
iii. The design should allow the user to easily put objects in and take them out after the scan is complete.

Chapter 4

DESIGN, IMPLEMENTATION AND RESULTS

In this chapter, our journey from the initial point to the final prototype has been discussed.

4.1 FIRST PROTOTYPE

In our first prototype, we used a wooden base to mount all the hardware. We used wood as it was easy to experiment with positions of the hardware on the surface. Also, the wooden surface could be written on and marked easily using a pencil, allowing us to measure and collect data easily. In our first prototype the target was to implement a working algorithm. In order to achieve a dark background, the setup was covered with a large cardboard box. The target of the first prototype was to achieve a working test bench to test our algorithms. There were 2 main areas that were focused on in the first prototype. These areas being –

i. Hardware Selection
ii. Image Processing Algorithm testing

4.1.1 Hardware Selection

For constructing the project, decisions related to the hardware had to be made. Decisions regarding the following prototypes had to be made –

i. Camera
ii. Laser Source
iii. Motor for turntable
iv. Circuitry and Microcontroller Selection

The decisions in this process and the challenges faced are discussed below.

4.1.1.1 Camera Selection

Figure 21 - VGA Camera Used In First Prototype

Initially experimentation was done with a VGA camera. The camera was purchased for a price of 400 PKR ($4), which is a very low cost for a USB camera. We expected the camera to be of low quality, though we purchased such a camera for two main reasons. The first reason was low cost; we wanted to experiment with a low cost camera before spending more on a better camera. Secondly, we knew a low cost camera would produce noisy images, which would provide a good test to our algorithms.

Challenges

The camera we purchased had a maximum resolution of 640 x 480 pixels, which was lower than what we finally wanted. The camera produced a lot of salt noise in the dark. Salt noise is caused where the sensor of a camera may not have a good dynamic range. For instance, in the dark, whenever the laser source would turn on, pixels in the dark region, particularly close to the laser would get saturated and cause impulse noise pixels of very high brightness (close to maximum). A discussion of the different algorithms we tested are given in Section 2.1.2 . More over the lens of the camera caused chromatic aberrations and fringing effects.

4.1.1.2 Laser Source for Laser Line

We required a laser line for the triangulation method. A regular 5mW pointing laser was used as the source, however to generate a laser line, a point had to be converted into a line. There were multiple options available to convert a laser point to a line–

i. Using a Powell Line lens

ii. Using a cylindrical glass rod in front of the laser pointer

iii. Using a cylindrical glass lens (commonly called laser line generator lens)

Powell Lens

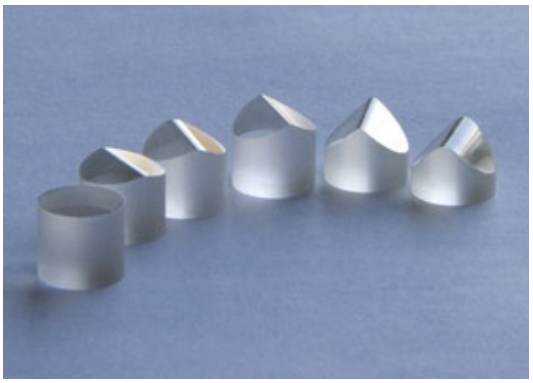


Figure 22 - Powell Lens[8]

A Powell lens is a cylindrical lens with a triangular top, whereas the bottom is aspherical. This type of lens offers the best performance of all the above mentioned lenses. Powell lenses are made from glass with a relatively high refractive index. The aspherical lens makes a uniform beam, with constant width, and uniform distribution of intensity. The Powell lens has to be chosen according the right laser frequency. Moreover the downside of a Powell Lens is that the costs quite a lot. The lowest price we found of a Powell Lens was $18 just for the lens, by a low quality manufacturer. Lenses from a more respectable manufacturer were in the range of $40-60 for readymade lenses (not made to order). The housing of these lenses was a separate cost that had to be considered.

Cylindrical glass rod in front of laser pointer

A common way of generating a laser line from a point is to place a cylindrical rod in front of the laser source. This method is the cheapest of all methods costing less than $1 for both the glass rod and the laser pointer, but it also produces the worst laser line out of the mentioned methods. In this method, the laser line produced is follows a Gaussian distribution for the luminosity, and produced a line of non-uniform beam

width. Although this method was tested very early on for a short time, it was quickly dropped because of the following reasons –

The fan angle of the usable laser line was very small – just 20-30^{o}. This mean that to produce a laser line of a long length, the laser source had to be placed far away from the object. Moreover, due to the far distance from the object, the focus of the beam would be affected and the quality of the laser line would worsen.

The Gaussian distribution of the luminosity meant that we had to apply further image correction to deal with problems that arose with the binary threshold in our laser segmentation step. Although this image correction was easy to correct, it caused a performance hit that could have resulted in a bottleneck in our projects performance.

Cylindrical glass lens (Laser line generator lens)

Figure 23 - Cylindrical Glass Lens with Screw Mount

The third method of laser line generation strikes a balance between price and performance. We bought lasers that were made for the use in laser leveling equipment. The price of the laser and lens combined was less than $3. The laser line being produced had a Gaussian distribution for the luminosity, however the effect was minimal, and did not affect our algorithms. Also the fan angle being produced in this method was greater than 80^{o}, this meant that the laser could be placed close to the object, while having enough length to cover the object from top to bottom. The added

advantage of this method was that the lens was a screw-type lens, which allowed the beam to be focused at different lengths.

Comparison

The features of the 3 laser line producing methods are compared below –

Table 2 - Comparison of Laser Line Producing Methods

	Powell Lens	**Cylindrical Rod**	**Cylindrical Lens**
Luminosity	Non-Gaussian	High Gaussian distribution	Gaussian, but very less effect
Uniformity	Highly Uniform	Highly Non-Uniform	Relatively Uniform
Fan Angle	Available in up to 90^{o} Fan Angle	$20\text{-}30^{o}$	Greater than 80^{o}
Focusable	Yes, but requires expensive holder	No	Yes, screw type
Cost	$18-60, just for the lens, approximately $0.5 for the laser pointer	Less than $1 for lens and laser	Less than $3 for the lens and laser

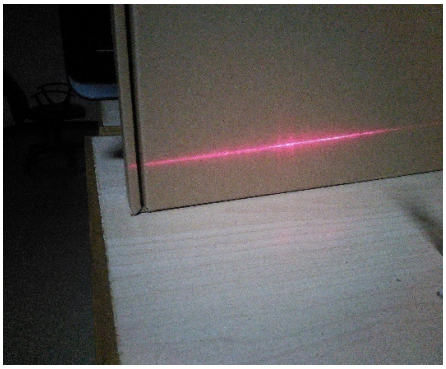


Figure 24 - Laser Line Using Glass Rod

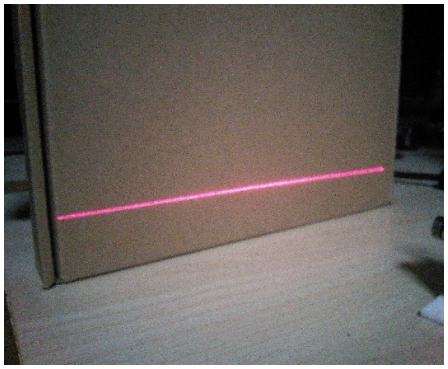


Figure 25 - Laser Line Generator Using Cylindrical Lens

Considering the comparison of the methods of laser line generation, a Cylindrical Lens with laser pointer was finalized. After testing this method, it was clear that it was sufficient for our purpose, and spending a much higher amount to purchase a Powell Lens was not required, as the benefit was minimal. In fact, the cylindrical lens allowed us to focus the beam, making it perfect for our purpose. Therefore our project uses the cylindrical lens with laser pointer.

4.1.1.3 Motor for turntable design

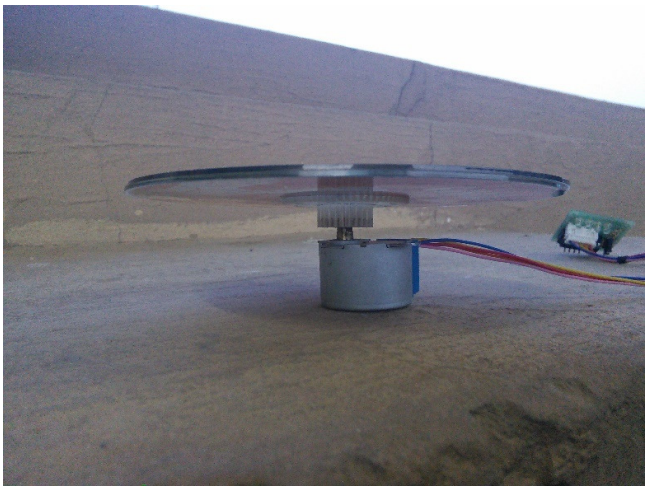


Figure 26 - Turntable for First Prototype

For the first prototype a hobby stepper was used for the rotation in the turn table. The stepper used had 100 steps per revolution. For the surface of the turntable, a CD (Compact Disc) was used, attached using a plastic gear. This motor was driven using a circuit with a ULN2003 (Darlington Transistor array) based driver.

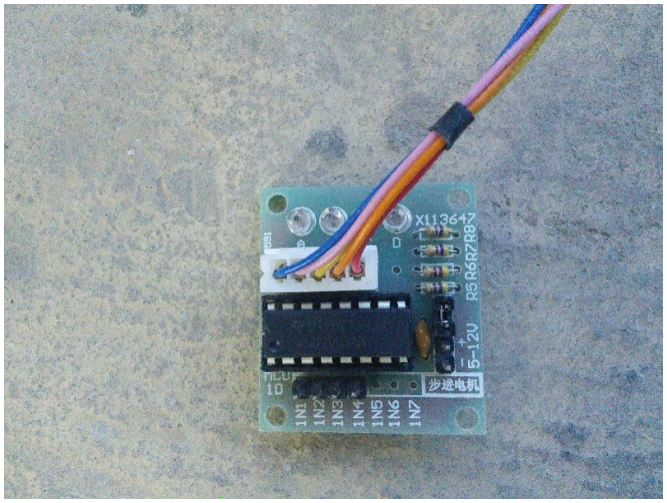
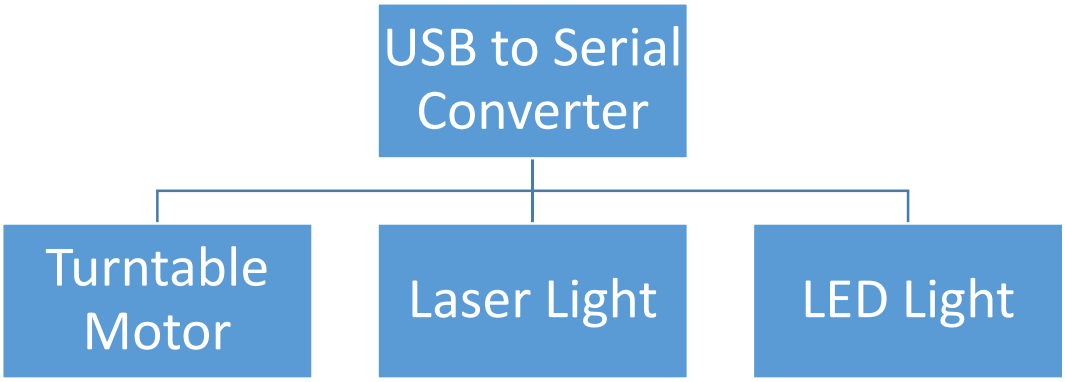


Figure 27 - ULN2003 Motor Driver

This motor was used just to provide a test bench for our initial algorithms. It did not have enough torque to rotate objects more than a few hundred grams. Also, the 100

steps/revolution meant that we were limited to only 100 slices per object, which produced low quality results.

4.1.1.4 Circuitry and Microcontroller Selection



For testing our algorithms, we required a board that was easy and quick to program, we did not need a very high performance microcontroller, as our tasks were not intensive on the microcontroller side. We had to take input from the serial port (or rather a USB port via a USB to Serial Converter) and change steps of the stepper motor accordingly. Considering these requirements, the Arduino Uno seemed a good choice for the project. We had a few benefits using the Arduino Uno, which used an Atmel ATmega 328P controller –

i. On board Serial to USB converter
ii. Hardware Serial with built in buffer
iii. In System Programming through USB
iv. No fuse bit configuration required when used with Arduino

At this stage of the project, breadboards were used to layout the circuit.

4.1.2 Image Processing Algorithm testing

In order to test the algorithms as outlined in Chapter 2, MATLAB® was used. MATLAB® provided us the following functionalities which was useful in this stage –

i. Easy Programming because of availability of making scripts and functions
ii. Readily available Documentation
iii. Many filters already available to use
iv. Availability of Image Processing toolbox
v. Graphing tools to monitor progress and compare results
vi. Ability to save variables states in a .mat file for later use
vii. Ability to produce Surface Plots

Although MATLAB® provides us with all these functionalities and benefits, it comes with a tradeoff of performance when compared with code C++ and C.The code written in MATLAB® is provided in the Appendix.

4.1.3 Results and Observations

We were able to produce 3D scans of objects that were within the requirements of our problem statement, however the quality of these scans could be improved dramatically. A number of observations were made, discussed as follows –

i. Camera was of low quality and low resolution
ii. Motor could only rotate very light objects
iii. Motor had very low steps/revolution
iv. MATLAB® took a lot of time to acquire images, and process them
v. The cardboard box that covered the assembly allowed light in from corners and edges which affected our image processing algorithms.

In order to mitigate these issues and solve the problems in the first prototype, the following decisions were made –

i. A High resolution webcam with at least a 2 Megapixel Sensor, with a high quality lens
ii. Acquire a motor with high torque to rotate heavier objects

iii. Acquire a motor with high steps/revolution, at least 400 steps/revolution
iv. Migrate the code to C++ and use OpenCV for image processing instead
v. Use a dark colored casing for our design, and ensure that all edges are covered properly

While testing our MATLAB® code we came up with the idea and technique of making Colored 3D Laser Scans, instead of Monochromatic 3D scans. An explanation of this technique is given in the next section.

4.1.4 Innovation – Colored 3D Laser Scanning

In order to obtain a Colored 3D Scan we needed another information from the camera, the color. Previously, we turned on the laser in the dark and captured the image. This meant that no color was seen by the camera, it could only see the red laser line distorted by the shape of the object. In order to obtain color, we had to generate light through a light source, and take an additional picture with the laser light turned off. The picture with the light off and laser on (say picture 1), as well as the picture with the light on and the laser off would be taken in the same position (say picture 2), i.e. without changing the angle of the turntable. After extracting the laser from picture 1, we knew the pixel coordinates of the laser line. By using these pixel coordinates, we could extract the color at the respective coordinates from picture 2 by storing the RGB values of the image at these coordinates.

4.2 FINAL PROTOTYPE

After the first prototype was made our focus changed towards product development. The algorithms that we tested in the first prototype proved well for our purpose. In this prototype our focus was on the following things –

i. Improvement of Hardware
ii. Improvement of Image Processing Performance
iii. Implementation of Color Laser Scanning
iv. Product Housing Design and Circuit Design
v. GUI design and Implementation

4.2.1 Improvement of Hardware

In order to improve the hardware we used the results and observation of the previous prototype. Improvements were done in the following areas –

i. Camera
ii. Motor Selection
iii. Microcontroller Circuitry

4.2.1.1 Camera

Figure 28 - Logitech C525 HD[9]

As discussed previously, a newer, higher quality webcam was required Image Acquisition. The webcam we upgraded to was Logitech® HD C525. This webcam provided us with the following features –

i. 1280 x 720 pixel resolution images
ii. Carl Zeiss® Lens, for minimal distortion and chromatic aberration

The cost of this webcam was 3200 PKR ($32).

4.2.1.2 Motor Selection

We needed to improve our motor for 2 reasons –

i. To increase number of slices in 3D scan by increasing steps/rev of the motor
ii. To increase the amount of weight that can be rotated by the stepper by increasing the torque of the motor.

In order to improve the steps/rev we had 2 options –

i. To buy a stepper with high steps/rev
ii. To buy a stepper with low steps/rev and attach a gearbox to decrease step angle, and therefore increase steps/rev

We analyzed both of these options keeping torque and price in mind. For motors that had a high steps/rev rating, torque was low. We found some motors that had both high steps/rev rating and high torque, but their size was quite large. Looking at the second option, buying a stepper with low steps/rev and attaching a gearbox, we had the advantage of low price, and high torque and small size. A comparison of these options is given below –

Table 3 - Comparison of Stepper Motors

	Gearbox	**Resolution**	**Size**	**Torque**	**Current**
Stepper 1 - **23HM22-2804S**[10]	No	400 steps/revolution	Nema 23	1.26Nm	2.8A
Stepper 2 - 16HS13-0604S-PG14 [11]	Yes	2800 steps/revolution	Nema 16	2.5 Nm	0.6A

Stepper 2 uses the Nema 16 motor, of 200 steps/rev along with a planetary gearbox of 14:1 ratio, effectively increasing the shaft resolution to 200 x 14 = 2800 steps/revolution. It is also pertinent to note that Stepper 2 uses 2.8/0.6 = 4.667 times less current than stepper 1.

Figure 29 - Stepper 1[10]

Figure 30 - Stepper 2[11]

Stepper 2 was used in our final prototype due to the following reasons –

i. It provided more torque than Stepper 1
ii. It had more steps than Stepper 1, by 7 times
iii. It occupied a smaller footprint than Stepper 1
iv. It used less current than Stepper 1

After the motor was decided, it was purchased from the source (www.omc-stepperonline.com) along with the recommended driver (ST-7128).

Figure 31 - ST-7128 Stepper Driver[12]

4.2.1.3 Microcontroller circuitry

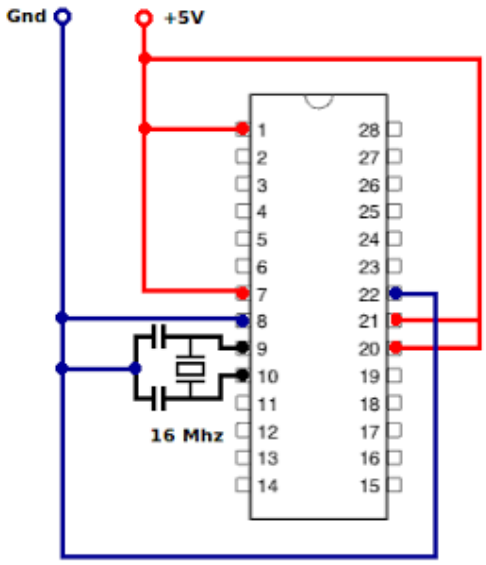


Figure 32 - Run ATmega 328 StandAlone

It was decided in this stage that we no longer wanted to use the Arduino Uno board, instead we decided to use the ATmega 328P independently. The programming was still done through the Arduino Software, though the Uno board was no longer used. In order to run the ATmega 328P independently of the Arduino Uno Board a few components were required, such as a crystal oscillator of 16.000 MHz, 18pf capacitors and a USB-Serial Converter. The circuit design is discussed in detail in section 4.2.4 .

4.2.2 Improvement of Image Processing Performance

As mentioned in Section 4.1.2 all algorithms were implemented in MATLAB®. Although MATLAB® provides an easy to use software for testing algorithms, it does not offer the same performance other languages such as C and C++ offer. Moreover to deliver software in the final product it would be much better to compile C++ code rather than MATLAB® code.

4.2.2.1 Migrating code to C++ and OpenCV

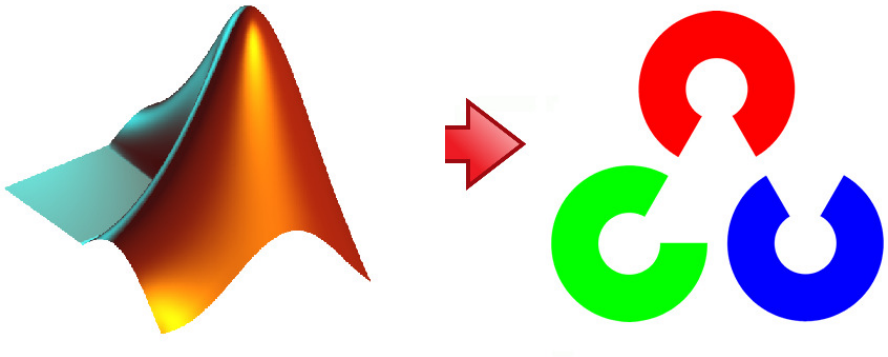


Figure 33 - Migrating Code from MATLAB to OpenCV

After testing algorithms on MATLAB® we moved our code to C++ (using OpenCV[13] Library) for performance enhancement. For Laser segmentation we used an Inverted 2nd Order Gaussian Filter (as explained in Section 2.1), whereas for Depth Calculation we used the FOV method (as explained in Section 2.2). In MATLAB we faced a bottleneck of image acquisition time, which took up to 1 second per frame to acquire when set to maximum resolution.

A comparison of Image Acquisition times in MATLAB® vs OpenCV is shown below –

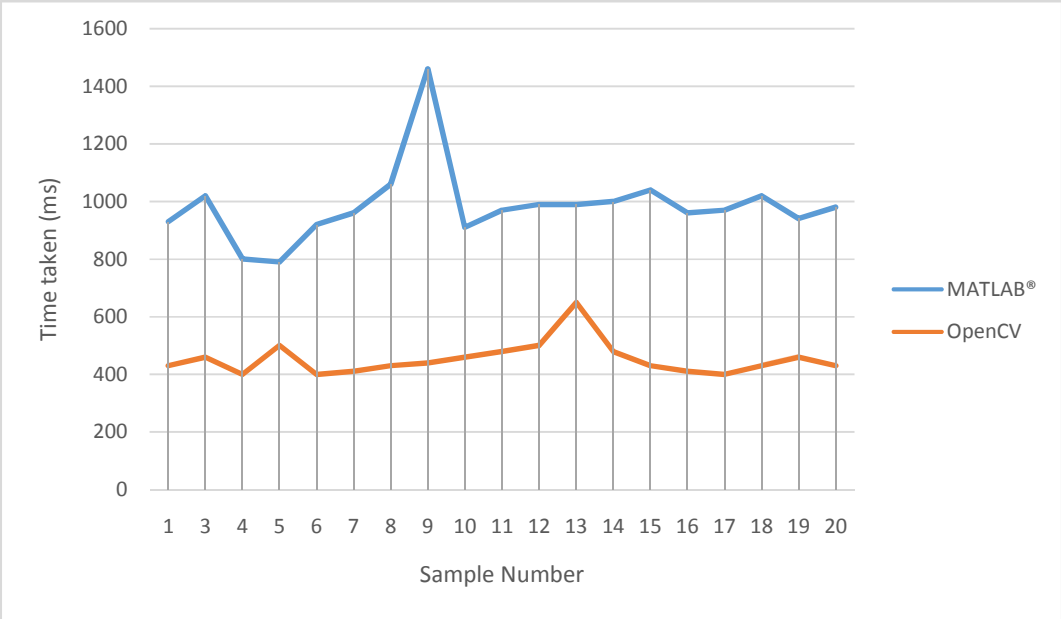


Figure 34– MATLAB® vs OpenCV Image Acquisition

It must be noted here that the time taken for acquisition includes the total time it takes to –

i. Capture the image
ii. Save the image &
iii. Load the image

Analyzing the results of the two languages it is seen that MATLAB® takes approximately 980 milliseconds(average of 20 samples) to acquire an image whereas OpenCV acquires and image in approximately 450 milliseconds (average of 20 samples).This results in greater than a 50% reduction in acquisition time, greatly improving the performance of the Acquisition Timing.

A performance improvement was also seen in the image processing step as well. Following is a comparison of the image processing times in MATLAB® and OpenCV –

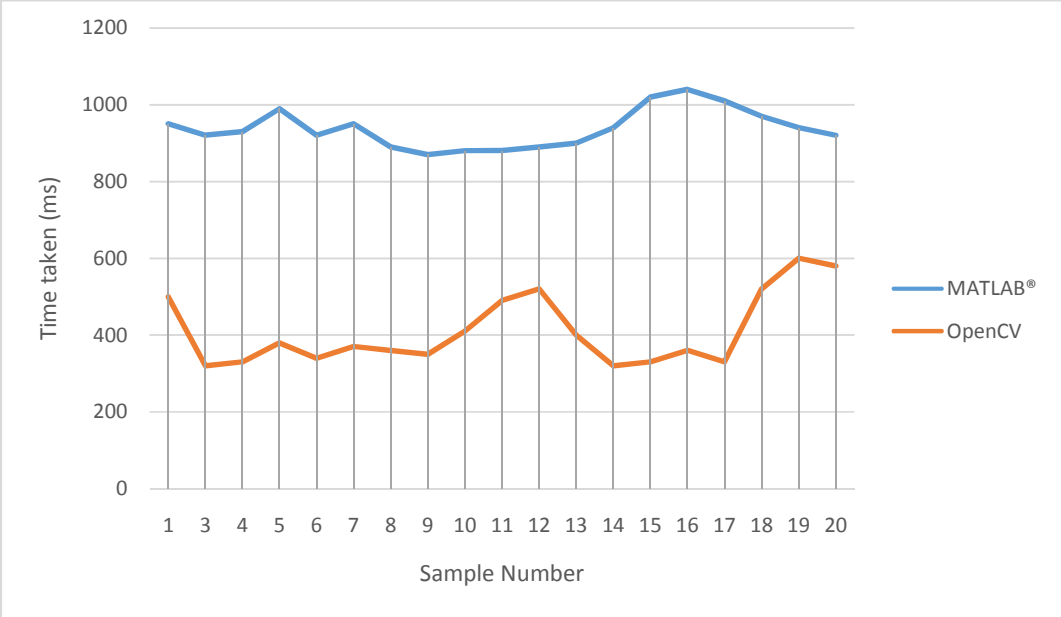


Figure 35 - Comparison of Image Processing Time in MATLAB(r) vs OpenCV

4.2.3 Implementation of Colored 3D Laser Scanning

By the time came for implementing the Colored 3D laser scanning, we had moved most of our code to C++ and OpenCV. This algorithm was tested directly on OpenCV and implemented on it as well. On the hardware side this required us to generate light, and turn off the laser before acquiring an image for color intensity purposes. We added Power LEDs to our design for lighting purposes. Power LEDs also require a driver to drive them.

4.2.4 Product Design and Circuit Design

This Section focusses on two areas –

i. Product Design
ii. Circuit Design

4.2.4.1 Product Design

While designing the product's housing we kept one thing in mind – Simplicity.

---

*"…focus and simplicity: Simplicity can be harder than complex: You have to work hard to get your thinking clean to make it simple. But its worth it in the end, because once you reach there, you can move mountains." – **Steve Jobs***

Choice of Material

We had some constraints on the type of materials we code use in the product. Some of these constraints were –

i. The material should be easy to mill/cut to our requirements
ii. It should be relatively low cost while maintaining a sophisticated yet simple look
iii. It should be available locally, so if more material was needed, it could be sourced quickly

iv. It should be relatively light weight

v. It should be dark in color, so that light from the laser is not reflected at unwanted angles inside the scanner. Moreover a dark color will provide the high contrast we require for image acquisition

Keeping these constraints in mind we looked at multiple materials we had access to, a comparison of these materials is given below –

Table 4 - Comparison ofthe Different Materials for Product Housing

Material	Easy to Mill/Machine	Lightweight	Available in dark color	Sophisticated look	Availability
Steel Sheets	No	No	Can be painted	No	Readily Available
Acrylic Sheets	Yes	Yes	Yes	Yes	Readily Available
Aluminum	Yes	No	Can be anodized or painted	Yes	Available
Wood	Yes	No	Can be lacquered	No	Readily Available

The following conclusion was made regarding the materials available –

i. *Steel Sheets* – Steel sheets were rejected due to their heavy weight, difficulty of working with, and due to the extra step to make it dark

ii. *Acrylic Sheets* – Acrylic sheets satisfied all our requirements.

iii. *Aluminum* – Aluminum provided us with the best material in terms of sophisticated looks, but it was still a heavy material in terms of weight. Moreover, the machining cost and the raw material cost were quite high when compared with Acrylic Sheets

iv. *Wood* – Wood is easy to machine with a milling machine, moreover it can easily be worked on using hand tools. Wood was used in our first prototype, for this reason. However, wood is a heavy material and does not give the same sophisticated look that Aluminum and Acrylic gives. Also, wood is not a material that a consumer wood expect a 3D laser Scanner would be.

Forms of plastic other than acrylic were not considered due to the extensive costs of tooling involved in making molds for injection molded plastic or for vacuum molded plastic. It was decided that we should use black acrylic sheets for the purpose of designing our product housing. Black Acrylic sheets met all our requirements while being easy to procure, low-cost to purchase and easy to machine.

Choice of Tools– Software for designing the Product

To produce models of the 3D scanner, SolidWorks® was used. SolidWorks® allows the user to create 3D models, do simulations, create renderings, as well as output files that can be used in milling machines, 3D printers, Laser scanners and many more.

Choice of Tools – Machining tools

For machining the acrylic we had multiple options –

i. Milling Machine

ii. Cutting and Drilling using Hand tools

iii. Laser Cutting

A comparison of these tools is given below –

Table 5 - Comparison of Machining Tools

	Cost	Finish Quality	Accuracy
Milling Machine	High	Good Quality	Highly accurate

Hand tools	Low	Low Quality	Inaccurate – prone to error
Laser Cutting	Medium	Excellent Quality	Highly Accurate

The use of hand tools was rejected due to low quality of work, since a high level of skill is required for any circular cutting. The decision between laser cutting and milling machine had to be made. After careful research, it was decided that laser cutting was a better option, the excellent quality being produced and the relatively low cost as compared with production using milling machinery.

The following was the final rendering of our 3D scanner in Solidworks®.

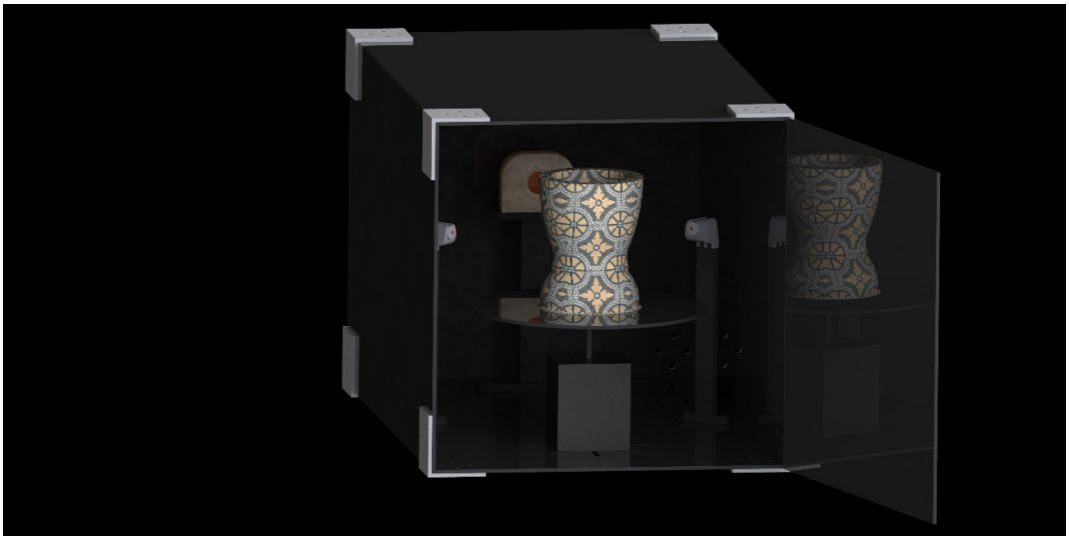


Figure 36 - Rendering of 3D Scanner

4.2.4.2 Printed Circuit Design

The PCB was designed using ISIS Proteus® 8 Professional. While designing the PCB a low component count was ensured to keep costs low. Moreover, only locally available ICs were used to avoid any problems with parts sourcing. The circuit had 4 main functions –

i. To Control the Motor Driver
ii. To Control the Lighting inside the Laser Scanner (Turn on/off the LED lights)
iii. To Control the Laser line (Turn on/off the Laser line)
iv. To Connect the Microcontroller to the computer

Following is the schematic that was implemented –

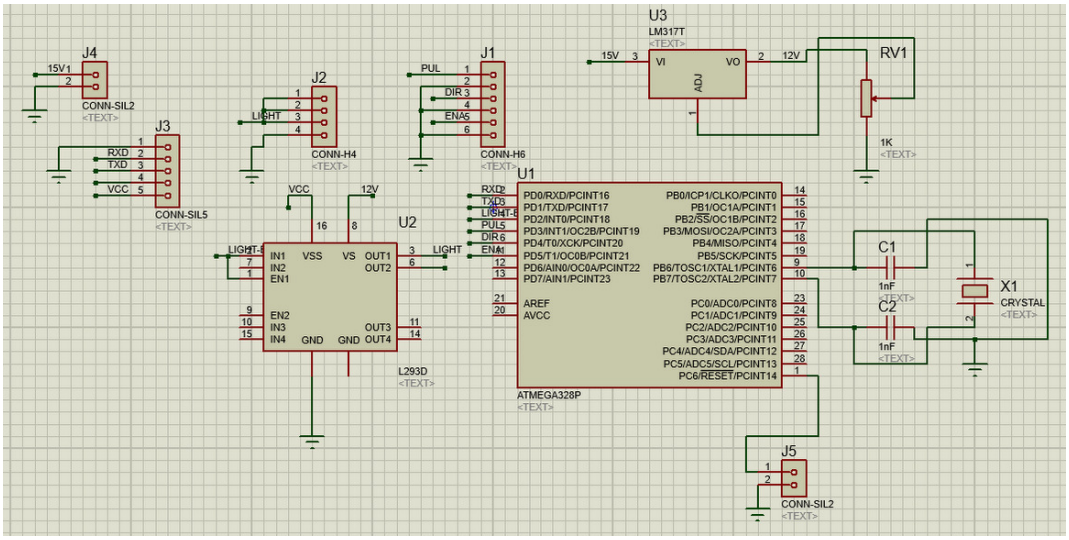


Figure 37 - PCB Schematic

Following is a picture of the final PCB–

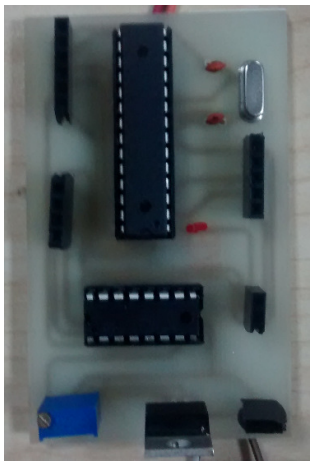


Figure 38 - Picture of PCB

4.2.5 GUI Design and Implementation

Like the Product design, the main focus in designing the GUI was simplicity and ease of use. The following is the GUI we designed –

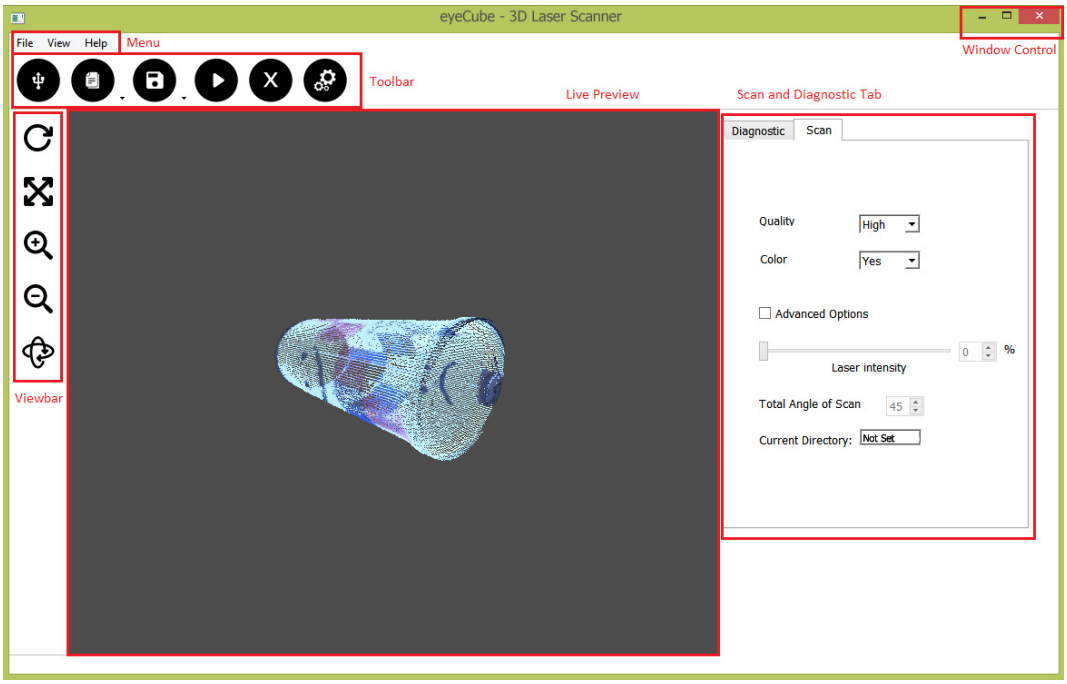


Figure 39 - GUI Design

There are 6 main areas of the GUI –

i. The Menu
ii. The Toolbar
iii. The View bar
iv. The live preview window
v. The scan and diagnostic Tabs

Qt creator was used for development of UI of application. It provides a drag and drop feature for integrating different UI components (known as Widgets in Qt) in a main window. Buttons, Action, Spin boxes, tabs are examples of widgets. Analogy of signals and slots was used to assign tasks to different action performed by a widget. When a widget is triggered (clicked, toggled etc.) a signal is generated to which a corresponding callback function (known as slot).

The functionalities of each of these areas is explained in the next sections

4.2.5.1 The Menu

The menu area of the GUI contains the following tabs –

i. The File Tab

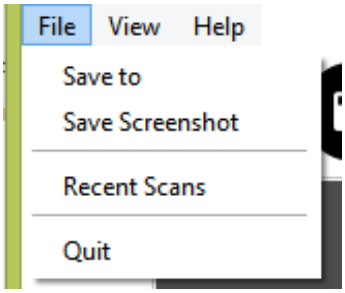


Figure 40 - GUI File Tab

In this Tab the following options are available –

a. *Save To*- To choose a location to save the 3D Scan
b. *Save Screenshot* – Takes a screenshot of the Live Preview Window
c. *Recent Scans* – Recent Scans shows a list of recent 3D scans
d. *Quit* – Exits the program

ii. The View Tab

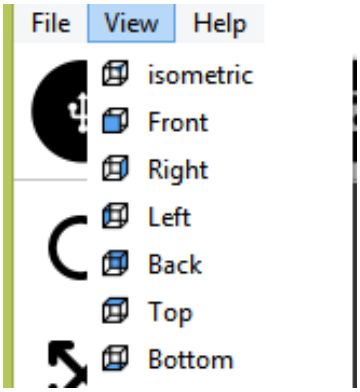


Figure 41 - GUI View Tab

The GUI Tab has the following options –

a. *Isometric* – Displays an Isometric view in the Live preview window
b. *Front* – Displays the front of the scanned object in the Live preview window
c. *Right, Left, Back, Top, Bottom* – Displays the respective view in the Live preview window

iii. The Help Tab

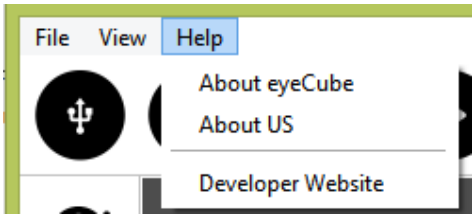


Figure 42 - GUI Help Tab

The Help tab has the following options available

4.2.5.2 The Toolbar

The toolbar has the following buttons available –

i. The connect button

Figure 43 - GUI Toolbar - Connect Button

This button is used to connect to the 3D Laser Scanner

ii. The Load Button

Figure 44 - GUI Toolbar - Load Button

This button is used to load recent scans

iii. The Save Button

Figure 45 - GUI Toolbar - Save Button

This button is used to save the current scan

iv. Start Scan

Figure 46 - GUI Toolbar - Start Scan

This button is used to start the scan

v. Stop Scan (Emergency Stop)

Figure 47 - GUI Toolbar - STOP Scan

This button is used to stop a scan midway

vi. Settings Button

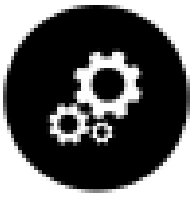


Figure 48 - GUI Toolbar - Settings Button

This button is used for the following settings –

a. S*etting Working Directory* – The working directory is a folder where all the temporary files are saved

b. *Camera Selection* – Since a webcam is being used in the laser scanner, it appears to the computer as a normal webcam. It needs to be selected so that any other camera that is otherwise attached to the computer is not used instead.

c. *COM Port Selection* – The COM Port for communication with the laser has to be selected to avoid confusion with any other Serial Port device connected to the computer

4.2.5.3 The View bar

The view bar contains the following options –

i. Rotate – allows the user to rotate the model

Figure 49 - GUI View bar - Rotate

ii. Restore default zoom – Resets the zoom to default

Figure 50 - GUI View bar – Restore Default Zoom

iii. Zoom in – Zooms into the model

Figure 51 - GUI View bar - Zoom In

iv. Zoom out – zooms out of the model

Figure 52 - GUI View bar - Zoom Out

v. Animate Model – Animates the model

Figure 53 - GUI View bar - Zoom Out

4.2.5.4 The Live Preview Window

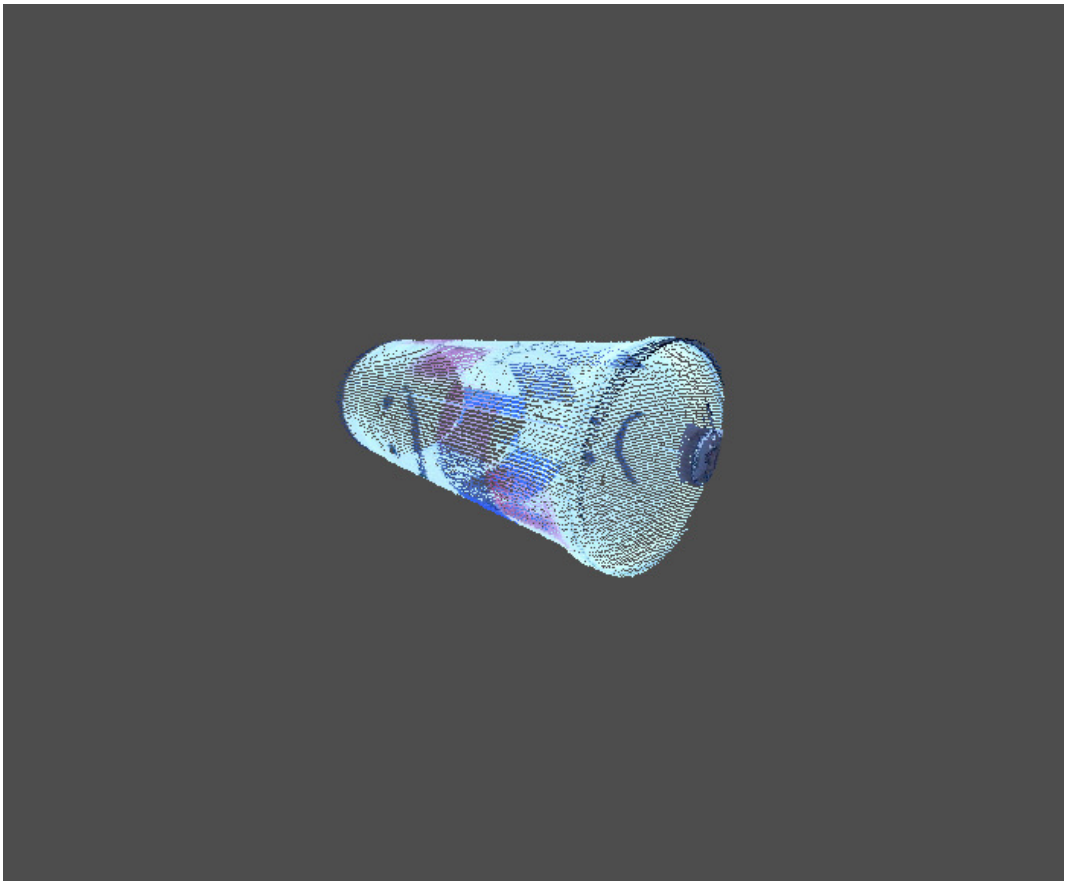


Figure 54 - Live Preview Window

The live preview window shows a preview of the object as it is being scanned, alternatively it shows a previously scanned model if one has been loaded.

4.2.5.5 The Scan and Diagnostic Tabs

The Scan tab shows the following information –

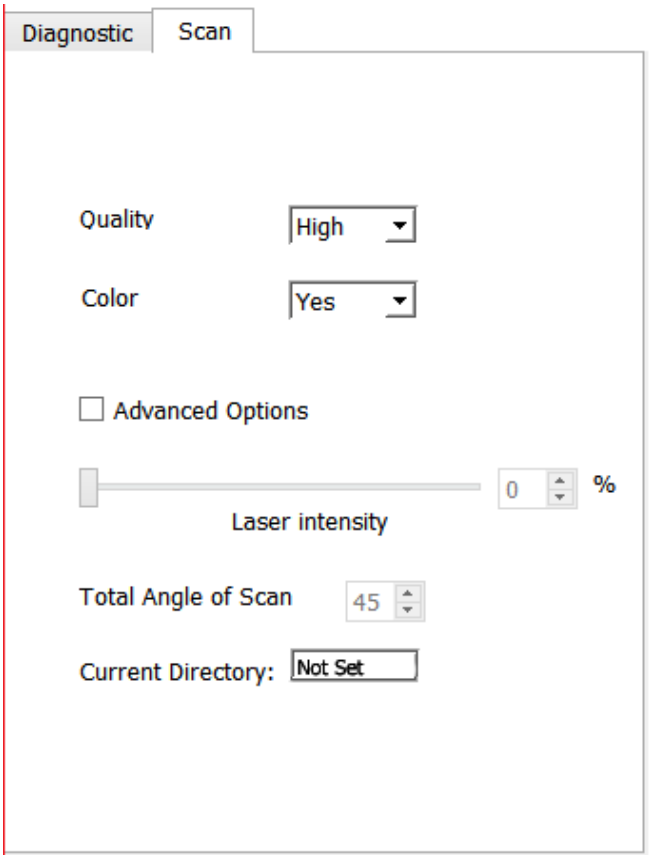


Figure 55 - GUI - Scan Tab

In this tab the user can select the quality of the scan, which affects the number of slices being taken. The user can also select color or monochrome scan. The user can also set the laser intensity and angle of scan (for e.g. if only 45^{o}of an object needs to be scanned), as well as change the current directory when Advanced options tick-box is ticked.

In the diagnostic tab the user can performance checks to see if his laser scanner is working fine.

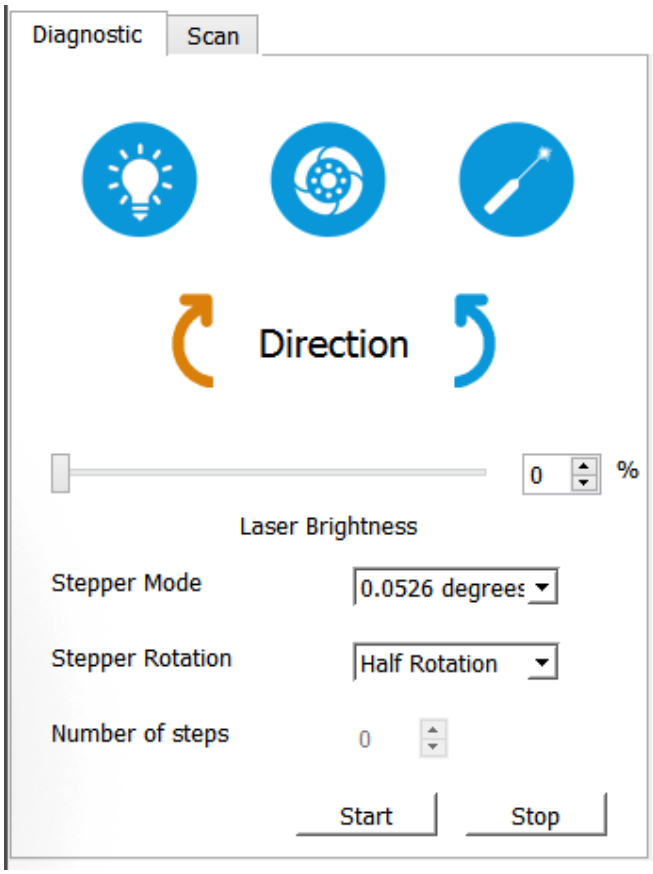


Figure 56 - GUI - Diagnostic Tab

As shown in the figure, the user can check the working of the LED lights in the scanner, the camera, as well as the laser line. The user can also rotate the stepper to see if it is working fine. Having these options can help the user solve any problem that may be faced. Since all the components are being checked, the user can track the root of the problem if any.

4.3 RESULTS

The results we obtained were reasonably well. The color information was accurate, and the profile was excellent. As shown in the results of our algorithm in section 2.2 the accuracy in depth is 99% which equals a maximum of 1.2mm in the longest end of our range at 120mm.

The images shown below were taken at 200 slices per revolution, and are the lowest resolution our project can scan. Even from these low resolution scans, the detail is obvious, the cricket ball's thread and the cat's whiskers can be seen in the 3D scans of the objects.

Figure 57 - Cricket Ball

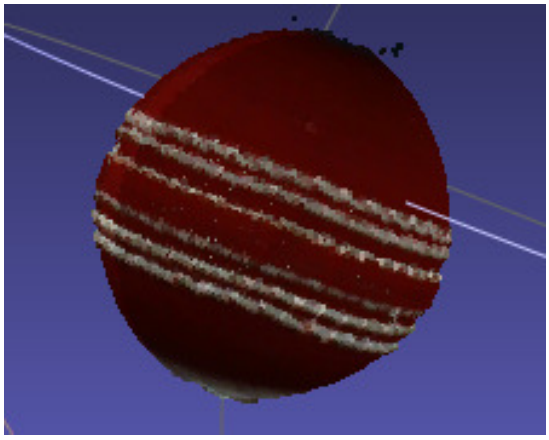


Figure 58 - Scan of a Cricket Ball

Figure 59 - Picture of Toy

Figure 60 - Scan of a Toy

Chapter 5

FUTURE RECOMMENDATIONS &CONCLUSION

5.1 FUTURE RECOMMENDATIONS

The product developed has many applications. Also, the algorithms discussed in this report are scalable and easily implementable in many different situations. These algorithms can be used in the following situations –

i. In a milling machine, to monitor milling of an object
ii. In a human body scanner for the purpose of brace design and prosthetic design
iii. To scan faces for digital versions of the same
iv. For character scanning for use in games and movies

5.2 LIMITATIONS &FUTURE IMPROVEMENTS

The following limitations exist in our project –

i. If laser is not seen by the camera due to the objects shape, that area will not be scanned
ii. There is dependence on the laser's angle for the resolution obtained
iii. We cannot scan transparent objects
iv. Very dark objects are sometimes not seen due to high absorption of light

There are a number of improvements that can be done to a future project. These are as follows –

i. Improve resolution of camera – to increase number of points in the point cloud
ii. Increase number of cameras – To ensure whole object is scanned.
iii. Increase number of lasers –To ensure no part of the object is missed.
iv. Increase power of laser – To have bright enough light to scan very dark objects

5.3 COMMERCIALIZATION

Since Rapid Prototyping is a growing field, and not much companies exist in low cost laser scanning, our project has a very viable future as a commercial product. Also, our software is easy to use and does not require any previous knowledge of 3D scanning, making it easy-to-use for the end user. Moreover, our product can easily be scaled into production because of the low part count and availability.

5.3.1 Impact on Society

3D laser scanning can be used in many areas that directly affect the quality of medical aid provided to people. One such area is that of prosthetic design as well as plaster cast design. Using 3D laser scanning, a doctor can take a scan of the fractured part of the body, and along with an x-ray, he can print a cast using 3D printing that is customized to the patient's needs. Similarly, in reconstructive surgery and prosthesis design, a doctor can customize a prosthetic to a patients damaged area. These applications can help millions of people worldwide.

5.4 CONCLUSION

This project proved to be a great challenge and learning experience for our group. We learnt many new skills and were able to develop many older ones. We were able to successfully implement the complete functionality that we set out to achieve. The developed product is able to do a complete scan of an object. The user gets an easy to use and intuitive interface to interact with the product.

REFERENCES

1] Scan. [Online]. Available: http://i0.wp.com/applications3d.com/wp-content/uploads/2014/03/Toy-development-Organic-surface-model-from-3d-scanning-Ready-for-mold-building.jpg.

2] Structured. [Online]. Available: http://chsopensource.org/2015/02/02/structured-light-3d-pico-projectors/.

3] Pixgood. [Online]. Available: http://pixgood.com/3d-laser-scanner.html.

4] O. M. T. S. R. L. A. Molder, "Laser Line Detection with Sub-Pixel Accuracy," Tallin, Estonia, 2014.

5] D. M. V. Matiukas, "Detection of Laser Beam's Center-line in 2D Images," Vilnius, Lithuania, 2009.

6] K. G. K. K. A. Malhotra, "Laser Triangulation for 3D Profiling of a Target," Noida, India, 2011.

7] M. R. A. T. Hok Hoo, "A Simple Surface Mapping Technique using Laser Triangulation Method," Nibong Tebal, Penang.

8] L. Optics, "Laser Line Optics - Laser line primer," [Online]. Available: http://www.laserlineoptics.com/powell_primer.html.

9] C. Logitech. [Online]. Available: http://www.logitech.com/en-us/product/hd-webcam-c525.

10] Stepper1. [Online]. Available: http://www.omc-stepperonline.com/9deg-nema-23-stepper-bipolar-28a-126nm1785ozin-23hm222804s-p-292.html.

11] Stepper2. [Online]. Available: http://www.omc-stepperonline.com/gear-ratio-141-planetary-gearbox-with-nema-16-stepper-motor-16hs130604spg14-p-38.html.

12] StepperDriverOMC. [Online]. Available: http://www.omc-stepperonline.com/bipolar-stepper-motor-driver-max-3a-current-128-high-subdivision-st7128-p-244.html.

13] OpenCV, "OpenCV," [Online]. Available: http://opencv.org/.

APPENDIX A : MATLAB CODES

Laser Extraction Code – By Max Value

```
close all
clear all
image=imread('d.jpg');%reading an image

image(:,:,1)=imnoise(image(:,:,1),'salt',0.00
1); %adding salt noise for testing
red=image(:,:,1);%separting the red part
[i m]=max(red');%selecting max values and
their indexes


[r c]=size(image(:,:,1));%getting the size
and dimensions
greeno=image;%making a blue line on the
original image
for k=1:3
   for i=1:r
      if(k==3)
         greeno(i,m(i),k)=255;
      else
         greeno(i,m(i),k)=0;
      end
   end
end
figure()
blue(:,:,1)=greeno(:,:,1);
blue(:,:,2)=greeno(:,:,2);
blue(:,:,3)=greeno(:,:,3);
imshow(blue)%new images with a blue line
imwrite(blue,'maxWithNoise.jpg');%saving
the new image file
```

Laser Extraction Code – Inverted Gaussian

```
function [ outo m] = imageProcess(
imgCrop )
%this applies the inverted 2nd order
guassian filter for laser filtering

%control parameters
enableGreyThresh=1;
%enabling grey thresh

%dilate and erosion
ndilate=3;%the no of dialations
nerode=3;%the no erosions
ndilate2=1;%the no of dialatoins for 2nd
time

%strucring element
se=[
   0 1 0;
   1 1 1;
   0 1 0
   ];
se1=[
   1 1 1;
   1 1 1;
   1 1 1
   ];
se=se1;

%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
%applying erosion, dialation and others
things for seeing which produces
%better results
flipo=(imgCrop);
if(enableGreyThresh==1)
   flipo=im2bw(imgCrop,
graythresh(imgCrop));
end
f=flipo;
%figure('Name' , 'Original image')

for k=1:2
   f=imdilate(f,se);
   %imshow(f)
end

for k=1:0
   f=imerode(f,se);
   %imshow(f)
end

flipo=f*255;
```

```
%flipo=imresize(flipo,10);
%figure('Name' , 'flipo');
%imshow(flipo)
% title('image')
im=flipo(1,:);



%# Create the gaussian filter with hsize =
[5 5] and sigma = 2
I=flipo;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  I(find(I(:,:)<100))=0;%seeting what
happens when some threshold is applied
%  I(find(I(:,:)>100))=255;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



% G = fspecial('gaussian',[5 5],2);
% Ig = imfilter(I,G,'same');
% %imshow(Ig)
% %figure();
% %plot(Ig(25,:))

sigma=6;
k=3;
% G1=fspecial('gauss',[round(k*sigma),
round(k*sigma)], sigma);
G1=fspecial('gauss',[ 1 , 200],
sigma);%creating a gassian filter
% [Gx,Gy] = gradient(G1);
% [Gxx,Gxy] = gradient(Gx);
% [Gyx,Gyy] = gradient(Gy);
%

%taking 2nd order guassin filter
[Gx] = gradient(G1);
[Gxx] = gradient(Gx);
% [Gyx,Gyy] = gradient(Gy);
nG1=G1/max(abs(G1));
nGx=Gx/max(abs(Gx));
nGxx=Gxx/max(abs(Gxx));
inGxx=nGxx*-1;

Ig = imfilter(I,Gx,'same');

meano=1/4*[1 1 1 1];
Ia=I(25,:);
Ia=imfilter(Ia,meano,'same');%applying the
mean filter for testing purposes

%figure('Name','origianl profile')
%plot(I(25,:))

%figure('Name' , 'average filter 1')
%plot(Ia)

%figure('Name' , 'average filter 2')
%plot(imfilter(Ia, meano,'same'))

%figure('Name' , '1st derivate')
%plot(Ig(25,:))

%figure('Name' , '2nd order derivate-
numeric derivative');
%plot(diff(Ig(25,:)))

%applying guassin filter for testing
%figure('Name', '2nd order derivative
convolution');
Ig1=imfilter(I,Gxx,'same');
%plot(Ig1(25,:))

%figure('Name' , '2nd order inverted
gussian derivative')
Ig2=imfilter(I,-1*Gxx,'same');
%plot(Ig2(25,:))

%figure('Name' , '2nd order inverted
normalized');
Ig3=imfilter(I,inGxx , 'same');%the actual
gussian filter used in the final results
%plot(Ig3(25,:))

%figure('Name' , '2nd order derivative
inverted');
%imshow(Ig3);

% %figure('Name', '1 order');
```

```
% %imshow(Ig);
% %figure('Name' , 'simple gussian');
% %imshow(imfilter(I,G,'same'));

%figure('Name' , 'After low pass filter');
lopass=imfilter(Ig3 ,
ones(5,5)*1/25,'same');
%imshow(lopass);


%figure('Name' , 'fitted on low pass filter');
%imshow(redLine(lopass))

% [outo m]=redLine(lopass);
[outo m]=redLine(im2bw(lopass,
graythresh(lopass)));
% [outo2 m]=redLine2(im2bw(lopass,
graythresh(lopass)));

Ig3a=Ig3;
%applying erosion and dialations
for k=1:ndilate
   Ig3a=imdilate(Ig3a,se);
end

Ig3b=Ig3a;
for k=1:nerode
   Ig3b=imerode(Ig3b,se);
end


Ig3a=Ig3b;

for k=1:ndilate2
   Ig3a=imdilate(Ig3a,se);
end


end
```

```
%reading imaegs
image=imread('d.JPG');
image(:,:,1)=imnoise(image(:,:,1),'salt',0.00
1);
toSave='gaussianWithNoise.JPG';
toSave2='g2.JPG';

%extracting read profile
imageInRed=image(:,:,1);
%for extracting laser line
[gg
m]=imageProcess3(imageInRed);%applying
2nd order guassin filter
subplot(1,2,1)
imshow(imageInRed);
subplot(1,2,2)
imshow(gg)
%imsave(gg)


%fitting a greem line
[r c]=size(image(:,:,1));
greeno=image;

for k=1:3
   for i=1:r
      if(k==3)
         greeno(i,m(i),k)=255;
      else
         greeno(i,m(i),k)=0;
      end
   end
end
figure()
blue(:,:,1)=greeno(:,:,1);
blue(:,:,2)=greeno(:,:,2);
blue(:,:,3)=greeno(:,:,3);
imshow(blue)
imwrite(blue,toSave);

%fitting a blue line
[r c]=size(image(:,:,1));
greeno=image;
```

Inverted Gaussian – Part II

```
clc
clear all
close all
```

```
for k=1:3
   for i=1:r
      if(k==2)
         greeno(i,m(i),k)=255;
      else
         greeno(i,m(i),k)=0;
      end
   end
end
figure()
blue(:,:,1)=greeno(:,:,1);
blue(:,:,2)=greeno(:,:,2);
blue(:,:,3)=greeno(:,:,3);
imshow(blue)
```

Binary Threshold – Part I

```
function [ o , m ] = redLine( ar )
%this functions finds the max from left and
right and then choses the mid
[c i]=max(ar');%get max from left
[c i2]=max(fliplr(ar)');%get max from right
i;
i2;
[dummy l]=size(ar);%get the size
l=l+1;
i3=l-i2;
m=floor((i+(i3))./2);%takes their average

[r c]=size(ar);
narr=zeros(r,c,3);
for k=1:3%making
%     narr(:,:,k)=double(ar)./255.0;
      narr(:,:,k)=double(ar);

   if (k>1)
      for i=1:r
          narr(i,m(i),k)=0;
      end
   end
end
o=narr;%the output image

end
```

Binary Threshold - Part II

```
function [ o m ] = imageProcess2( img )
%UNTITLED6 Summary of this function
goes here
%   Detailed explanation goes here
img=imnoise(img,'salt',0.001);%adding
slatnoise
bin=im2bw(img(:,:,1));%converting image
to binary
%bin=imnoise(bin, 'salt',0.001);
imshow(bin)%show the binary imaeg
imwrite(im2bw(img(:,:,1)), '1.jpg');%saving
binary image
[o m]=redLine(im2bw(img(:,:,1),
graythresh(img(:,:,1)')));%finds the mid and
creates the output image
imshow(o)%show output iamge
imwrite(o,'2.jpg')%save output image
end
```

```
blue(:,:,3)=greeno(:,:,3);
imshow(blue)
imwrite(blue,'3.jpg');
```

Binary Threshold – Part III

```
clc
close all
clear all
image=imread('d.jpg');%reading an image
img=image;
red=img(:,:,1);
img(:,:,1)=imnoise(red,'salt',0.001);%addin
g salt noise for testing purposes
blue=img(:,:,2);%the blue component of
image
green=img(:,:,3);%the green component of
an image
imwrite(img,'4.jpg');%writing the red
component for testing purposes

% image=imnoise(imag,'salt',0.001);

[o m]=imageProcess2(image);%imploying
and finding center by binary threshold

%imshow(o)

[r c]=size(image(:,:,1));
greeno=img;
for k=1:3
   for i=1:r
      if(k==3)
         greeno(i,m(i),k)=255;
      else
         greeno(i,m(i),k)=0;
      end
   end
end
figure()
blue(:,:,1)=greeno(:,:,1);
blue(:,:,2)=greeno(:,:,2);
```

3D Object Construction

```
%this reads the images stored and output
the model as a stl file, which can be used
for 3d printing or for editing
%the cad model of an object
clc
%variables for keeping trace of rooth path
for saving images, angle of
%motor used in scanning and no of images
theta=0:2.1176:360;
noOfImages=170;
rootPath='C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\icubeI
mages\ObjectsScan\woodPiece\3dReconst
ruction\icube';
num=1;
type='.png';
%1280*720

center=1280/2;%the center of iamges
fixed for testing purposes
% refImg;

%variables for storing the results
```

```
% length=720*noOfImages;
length=720;
x=zeros(length,noOfImages);
y=x;
z=x;
depthArray=x;



%performing a complete 3d scan
for k=1:1:170
   %setting of rooth path from which
images will be picked

path=horzcat(horzcat(rootPath,num2str(n
um)),type);
   disp(path)
   imag=imread(path);%reading the image
   img=imag(:,:,1);
   [o m]=imageProcess(img);    %extracting
the laser line
   px=abs(m-center);
   rdepth=findDx2(px);%finds the depth in
real dimensions
   %in the end
   %finding the depth of the object
   [dummy l]=size(rdepth);
   x1=zeros(1,l);
   y1=zeros(1,l);
%     z1=1:1:l;%if height is not counted
%finding the height of object
   z1=findDy(rdepth); %actual depth


   %coverting depth to x,y and z
coordinates
   x1=rdepth*cos(degtorad(theta(k)));
   y1=rdepth*sin(degtorad(theta(k)));


%     x(1+(k-1)*l:1:k*l)=x1;
%     y(1+(k-1)*l:1:k*l)=y1;
%     z(1+(k-1)*l:1:k*l)=z1;
   depthArray(:,k)=rdepth;
   x(:,k)=x1;
   y(:,k)=y1;
   z(:,k)=z1;
%     x=horzcat(x,x1);
%     y=horzcat(y,y1);
%     z=horzcat(z,z1);

num=num+1;
end

% scatter3(x,y,z);

%converting objects to stl models
figure
surf(x,y,z);
```

```
Implementation of 2nd Research Paper
clear
close all
%reference ro
ro=678;%selecting a row for testing
theta=25.0*pi/180.0;%converting theta to
radians

%reading multiple images for testing
img1=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday5\a (1).JPG');%0
img2=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday5\a (2).JPG');%2
img3=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday5\a (3).JPG');%4
img4=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday5\a (4).JPG');%6
img5=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday5\a (5).JPG');%8
img6=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday5\a (6).JPG');%out
% img7=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday3\a (7).JPG');%out
% img8=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday3\a (8).JPG');%out
% img9=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday3\a (9).JPG');%out
% img10=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday3\a (10).JPG');%out
% img11=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday3\a (11).JPG');%out
% img12=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday3\a (12).JPG');%out
% img13=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday3\a (13).JPG');%out


%extracting the laser line, and selecting a
row for its testing
[i1 m1]=imageProcess(img1(:,:,1));
as(1)=length(m1)
[i2 m2]=imageProcess(img2(:,:,1));
as(2)=length(m1)
[i3 m3]=imageProcess(img3(:,:,1));
[i4 m4]=imageProcess(img4(:,:,1));
[i5 m5]=imageProcess(img5(:,:,1));
as(3)=length(m1)
[i6 m6]=imageProcess(img6(:,:,1));
% [i7 m7]=imageProcess(img7(:,:,1));
% [i8 m8]=imageProcess(img8(:,:,1));
% [i9 m9]=imageProcess(img9(:,:,1));
% [i10 m10]=imageProcess(img10(:,:,1));
% [i11 m11]=imageProcess(img11(:,:,1));
% [i12 m12]=imageProcess(img12(:,:,1));
% [i13 m13]=imageProcess(img13(:,:,1));


%display some for them in figures for
seeing the results
figure('Name' , 'fig1')
imshow(i1)
% figure('Name' , 'fig2')
% imshow(i2)
figure('Name' , 'fig3')
imshow(i3)
% figure('Name' , 'fig4')
% imshow(i4)
figure('Name' , 'fig5')
imshow(i5)
% figure('Name' , 'fig6')
% imshow(i6)
% figure('Name' , 'fig7')
% imshow(i7)
% figure('Name' , 'fig8')
% imshow(i8)
% figure('Name' , 'fig9')
% imshow(i9)
% figure('Name' , 'fig10')
% imshow(i10)
```

```
% figure('Name' , 'fig11')
% imshow(i11)
% figure('Name' , 'fig12')
% imshow(i12)
% figure('Name' , 'fig13')
% imshow(i13)

%Center of images
center=m1(ro);

%selecting a particular row
px(1)=m1(ro);
px(2)=m2(ro);
px(3)=m3(ro);
px(4)=m4(ro);
px(5)=m5(ro);
px(6)=m6(ro);
% px(7)=m7(ro);
% px(8)=m8(ro);
% px(9)=m9(ro);
% px(10)=m10(ro);
% px(11)=m11(ro);
% px(12)=m12(ro);
% px(13)=m13(ro);

%caculating the depth
length=6;
absolute_dis_px=zeros(1,length);


for k=1:length
   absolute_dis_px(k)=center-px(k);
end

for k=1:length-1

different_dis_px(k)=absolute_dis_px(k+1)-
absolute_dis_px(k);
end
%mesurements in centimeter
d_cm=0:1:length;
l_cm=d_cm*tan(theta);


%calculation scale
scale_arr=absolute_dis_px(2:length)./l_cm
(2:length);
px_per_cm=mean(scale_arr);

%calculation the depth
for k=1:length

l_cm_exp(k)=absolute_dis_px(k)/px_per_c
m;
   r_cm_exp(k)=l_cm_exp(k)/sin(theta);
   depth_exp(k)=r_cm_exp(k)*cos(theta);

end

%printing the varibales real depth,
experimental depth, error, error
%percentage for checking the accuracy of
the research paper
depth_exp
r_cm_exp
absolute_dis_px
different_dis_px
scale_arr
px_per_cm
as
true_values=[0 1 2 3 4 5];
error = r_cm_exp-true_values;
plot(error);%plotting the error graphs

% clear
```

```
Implementation of 1st Research Paper
clc
clear

%generation necessary parameters (which
were determined experimentally)
ro=92;
%13.4
refL=22;
d=14.1+refL;
a=4.6;
theta=atan(a/d);
center=320;

%reading 10 images for testing
img1=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday7\a (1).JPG');%10
img2=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday7\a (2).JPG');%12
img3=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday7\a (3).JPG');%14
img4=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday7\a (4).JPG');%16
img5=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday7\a (5).JPG');%18
img6=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday7\a (6).JPG');%20
img7=imread('C:\Users\Fahad
Mubeen\Documents\MATLAB\FYP\FYP\m
onday7\a (7).JPG');%22

%extracting laser line
[i1 m1]=imageProcess(img1(:,:,1));
[i2 m2]=imageProcess(img2(:,:,1));
[i3 m3]=imageProcess(img3(:,:,1));
[i4 m4]=imageProcess(img4(:,:,1));
[i5 m5]=imageProcess(img5(:,:,1));
[i6 m6]=imageProcess(img6(:,:,1));
[i7 m7]=imageProcess(img7(:,:,1));

%show figures for testign
figure('Name' , 'fig1')
imshow(i1)
figure('Name' , 'fig2')
imshow(i2)
figure('Name' , 'fig3')
imshow(i3)
figure('Name' , 'fig4')
imshow(i4)
figure('Name' , 'fig5')
imshow(i5)
figure('Name' , 'fig6')
imshow(i6)
figure('Name' , 'fig7')
imshow(i7)
%storing pixels
px(1)=m1(ro);
px(2)=m2(ro);
px(3)=m3(ro);
px(4)=m4(ro);
px(5)=m5(ro);
px(6)=m6(ro);
px(7)=m7(ro);

size(img1(:,:,1));
px



%calculating the focal length
h(1)=px(1)-center;
h(2)=px(2)-center;
h(3)=px(3)-center;
h(4)=px(4)-center;
h(5)=px(5)-center;
h(6)=px(6)-center;
h(7)=px(7)-center;
f=h(7)/tan(theta); %taking reference focal
length calculation by sing 7th image

%calculating the depth at different points
depth(1)=f*a/h(1);
depth(2)=f*a/h(2);
depth(3)=f*a/h(3);
depth(4)=f*a/h(4);
depth(5)=f*a/h(5);
depth(6)=f*a/h(6);
```

```
depth(7)=f*a/h(7);



%printing the values of depth, real depht,
error, error percentage for
%checking puproses
depth=depth-14.1
experimentalDepth=depth+14.1
ref=10:2:22;
originaldepth=ref
%+14.1

error=ref-depth
errorPercentage=(ref-
depth)./originaldepth*100
```

Image Capturing Code

```
%for taking snaps of objects from all sides
stepMotor=12;
%getting the respective video input, and
desired resolution for camera
vid =
videoinput('winvideo',2,'I420_1280x720');
%setting the vid objects which will be used
for image capturing later on
set(vid, 'ReturnedColorSpace', 'RGB');

directory='FYP/icubeImages/';
baseName='icube';
extension='.png';
num=1;
%creating an arduino object with
respective com port and baud rate
arduino=inilizeCommunication('COM5',960
0);
for i=1:stepMotor:2040
   %starting serial communication and
rotating the motor

strToArduino=(fliplr(num2str(stepMotor)))
;
   for j=1:numel(num2str(stepMotor))

toArduinoChar(arduino,strToArduino(j));
   end
%    pause(1)
   img = getsnapshot(vid);%getting the
image
   path=horzcat( horzcat(horzcat(directory,
baseName),num2str(num)),extension)%sa
ving screen shot
   imwrite(img,path);
   num=num+1;
end
fclose(arduino);%closing the arduino serial
communication
```

APPENDIX B – FINAL SOFTWARE CODE (QT)

Main.c

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
   QApplication a(argc, argv);
   MainWindow w;
   w.show();

   return a.exec();
}
```

Mainwindow.h

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QtCore>
#include <QtGui>
#include <QMessageBox>
#include <QToolButton>
#include <QWidgetAction>
#include <QFrame>
#include<QHBoxLayout>
#include<QScrollArea>
#include<QVBoxLayout>
#include<QString>
#include<QFileDialog>
#include"otherfunctions.h"
#include "mycanvas.h"
#include"variables.h"
#include "mydialog.h"
using namespace variables;


QIcon *lightsOn;
QIcon *lightsOff;
string rootPathforIcons="C:\\Users\\Fahad
Mubeen\\Documents\\QtProjects\\FinalPr
oject\\";
void makeIcons(){
   QIcon *lightsOn = new QIcon();
   lightsOn-
>addPixmap(QPixmap((rootPathforIcons+"
light.png").c_str()));

   QIcon *lightsOff = new QIcon();
   lightsOff-
>addPixmap(QPixmap((rootPathforIcons+"
light-pressed.jpg").c_str()));


}


MainWindow::MainWindow(QWidget
*parent) :
   QMainWindow(parent),
   ui(new Ui::MainWindow)
{
   ui->setupUi(this);

   this->setStyleSheet("background-color:
#FFFFFF");

   ui->slaserslider->setEnabled(false);
   ui->slasercombo->setEnabled(false);
   ui->scanangle->setEnabled(false);
   ui->dMotorStep->setEnabled(false);

   connect(ui->slaserslider,
SIGNAL(valueChanged(int)),ui-
>slasercombo ,SLOT(setValue(int)));
```

```
   connect(ui->slasercombo,
SIGNAL(valueChanged(int)),ui->slaserslider
,SLOT(setValue(int)));

   ui->dlabel->setStyleSheet("QLabel {
background-color : pink; color : blue; }");
   ui->toolBar->setMovable(false);
   ui->mainToolBar->setMovable(false);

   ui->stepperAntiClockwise-
>setChecked(false);
   ui->stepperClockWise-
>setChecked(true);

   connect(ui->laserSlider,
SIGNAL(valueChanged(int)),ui-
>dlaserCombo ,SLOT(setValue(int)));
   connect(ui->dlaserCombo,
SIGNAL(valueChanged(int)),ui->laserSlider
,SLOT(setValue(int)));

   //First drop down on toolbar
      QMenu *menu = new QMenu();
      QAction *customformatload = new
QAction("Custom Format", this);
      QAction *pictures = new
QAction("Pictures", this);
      menu->addAction(customformatload);
      menu->addAction(pictures);

      QToolButton* toolbutton = new
QToolButton();
      toolbutton->setMenu(menu);
      toolbutton-
>setPopupMode(QToolButton::InstantPop
up);
      toolbutton->setFixedSize(8,50);
      QWidgetAction* toolButtonaction =
new QWidgetAction(this);
      toolButtonaction-
>setDefaultWidget(toolbutton);

      ui ->mainToolBar ->insertAction(ui-
>actionSave,toolButtonaction);
      // First drop down on toolbar
      //First drop down on toolbar
      QMenu *menu2 = new QMenu();
      QAction *customformatsave = new
QAction("Custom Format", this);
      QAction *pointcloud = new
QAction("Pictures", this);
      QAction *tdmodel = new
QAction("Custom Format", this);
      QAction *picturesave = new
QAction("Pictures", this);
      menu2-
>addAction(customformatsave);
      menu2->addAction(pointcloud);
      menu2->addAction(tdmodel);
      menu2->addAction(picturesave);

      QToolButton* toolbutton1 = new
QToolButton();
      toolbutton1->setMenu(menu2);
      toolbutton1-
>setPopupMode(QToolButton::InstantPop
up);
      toolbutton1->setFixedSize(8,50);
      //toolbutton-
>setIcon(QIcon("D:/Test/untitled3/save.pn
g"));
      QWidgetAction* toolButtonaction1 =
new QWidgetAction(this);
      toolButtonaction1-
>setDefaultWidget(toolbutton1);

      ui ->mainToolBar ->insertAction(ui-
>actionStart,toolButtonaction1);
      MyCanvas* SFMLView = new
MyCanvas(this, QPoint(65, 85), QSize(718,
600));
      SFMLView->show();


      //making functions
      makeIcons();
      serialo::setSerialPort("COM13");

variables::cap.init(variables::cameraNo);
}

MainWindow::~MainWindow()
```

```
{
   delete ui;
}
void
MainWindow::on_pushButton_clicked(){}
//remove this in future my frinds



void MainWindow::on_lights_clicked()
{


variables::lightStatus=!variables::lightStatu
s;
   cout<<"onlightsclicked
lightStatus::"<<variables::lightStatus<<endl
;
   cout<<"serial communication
status::"<<variables::SerialCommunication
Status<<endl;
   cout<<"dignostic
tab"<<variables::diganosticsStatus<<endl;
   if(variables::SerialCommunicationStatus
&& variables::diganosticsStatus){
      cout<<"turning lights on and
off"<<endl;
      if(variables::lightStatus){

serialo::writeToSerial(variables::lightsOn);
      }else{

serialo::writeToSerial(variables::lightOff);
      }
   }

}

void MainWindow::on_laser_clicked()
{

   laserStatus=!laserStatus;
   cout<<"on laserClicked
status::"<<laserStatus<<endl;
   if(laserStatus){

serialo::writeToSerial(variables::laserLine1I
ntensity(variables::laserIntensity));

serialo::writeToSerial(variables::laserLine2I
ntensity(variables::laserIntensity));
   }else{

serialo::writeToSerial(variables::laserLine1I
ntensity(0));

serialo::writeToSerial(variables::laserLine2I
ntensity(0));
   }
}



void
MainWindow::on_stepperClockWise_click
ed()
{
   ui->stepperClockWise-
>setChecked(true);
   ui->stepperAntiClockwise-
>setChecked(false);
   stepperDirection=clockwise_;
   cout<<"stepper motor clockwise
clicked"<<stepperDirection<<endl;
}

void
MainWindow::on_stepperAntiClockwise_cl
icked()
{
   ui->stepperAntiClockwise-
>setChecked(true);
   ui->stepperClockWise-
>setChecked(false);
   stepperDirection=anticlockwise_;

cout<<"stepperMotorAntiClockwiseClicked
stepperDirection::"<<stepperDirection<<e
ndl;
}
```

```
void
MainWindow::on_diagnosticsStart_clicked
()
{
  diganosticsStatus=start_;
  string toSend="";
//    if(1){
//
cout<<"lightSTatus::"<<variables::lightStat
us<<endl;
//
cout<<"laserStatus::"<<variables::laserStat
us<<endl;
//       cout<<"laser
Intensity::"<<variables::laserIntensity<<en
dl;
//
cout<<"motorStatus::"<<variables::motorS
tatus<<endl;
//    }
  //composing
  if(variables::lightStatus==true){
     toSend+="d";
  }else{
     toSend+="h";
  }
  if(variables::laserStatus==true){
     switch(laserIntensity){
     case 0:
        toSend+="2";
        break;
     case 1:
        toSend+="3";
        break;
     case 2:
        toSend+="4";
        break;
     case 3:
        toSend+="5";
        break;
     case 4:
        toSend+="6";
        break;
     case 5:
        toSend+="7";
        break;
     }
  }
  if(variables::motorStatus==true){
     switch(stepperMode){
     case high_:
        toSend+="e";
        break;
     case medium_:
        toSend+="f";
        break;
     case low_:
        toSend+="g";
        break;
     }
     string numbero;
     if(stepperRotation==custom){
         toSend+="c";
         string
theNumber=to_string(variables::noOfSteps
);
         std::string dest = std::string( 3-
theNumber.length(), '0').append(
theNumber);
         toSend+=dest;

     }else if(stepperRotation==half){
        toSend+="k";
     }else{
        toSend+="l";
     }
//      switch(stepperRotation){
//      case custom:
//         toSend+="c";
//         //apending data in 3 digit steps
//         //numbero=string(3-
to_string(variables::noOfSteps).length(),'0'
).(to_string(variables::noOfSteps));
//         //string
number=to_string(variables::noOfSteps);


//         //         toSend+=number;
//         break;
//      case half:
//         toSend+="k";
//         break;
//      case full:
//         toSend+="l";
```

```
//         break;
//     }

  }
  cout<<"sending serail
string::"<<toSend<<endl;
  serialo::writeToSerial(toSend);
}

void
MainWindow::on_diagnosticsStop_clicked(
)
{

   diganosticsStatus=stop_;
   cout<<"dignostic stop clicked
diagonistics
status::"<<variables::diganosticsStatus<<e
ndl;
}

void
MainWindow::on_numberOfSteps_textCh
anged(){  }//delete

void
MainWindow::on_laserSlider_sliderMoved
(int position)
{
//    laserIntensity=position;
}

void
MainWindow::on_stepperMode_activated
(int index)
{
   if(index==high_){
      variables::stepperMode=high_;
   }else if(index==medium_){
      variables::stepperMode=medium_;
   }else if(index==low_){
      //low_
      variables::stepperMode=low_;
   }
   cout<<"onSTepperMode activated
stepperMode::"<<variables::stepperMode
<<endl;
}


void
MainWindow::on_stepperRotation_activat
ed(const QString &arg1){}

void
MainWindow::on_laserSlider_valueChange
d(int value)
{
   variables::laserIntensity=value;
   cout<<"laserIntensity value changed
intensity::"<<variables::laserIntensity<<en
dl;
//    if(variables::laserStatus==true){
//       switch(laserIntensity){
//       case 0:
//
serialo::writeToSerial(serialo::serial,"2");
//          break;
//       case 1:
//          break;
//       case 2:
//          break;
//       case 3:
//          break;
//       case 4:
//          break;
//       case 5:
//          break;
//       }
//    }
}

//noOfSteps
void
MainWindow::on_lineEdit_returnPressed()
{
   //variables::noOfSteps=stoi(ui->lineEdit-
>text().toUtf8().constData());
   //cout<<"lineEdit No of
Steps::"<<variables::noOfSteps<<endl;

}
```

```
void
MainWindow::on_actionConnect_triggere
d()
{
   cout<<"connect clicked"<<endl;

variables::SerialCommunicationStatus=seri
alo::setSerialPort(variables::serialPortNam
e.c_str());
   cout<<"serial
status::"<<variables::SerialCommunication
Status<<endl;
}

void
MainWindow::on_actionLoad_triggered()
{
   //loading a file
   cout<<"load clicked"<<endl;

   cout<<"load clicked"<<endl;
   QString filename=
QFileDialog::getOpenFileName(this,
tr("Open File"),"",tr("(*.txt *.ply)"));
   string
toSave=filename.toUtf8().constData();
   cout<<"to load::"<<toSave<<endl;

   if(toSave!=""){
      eraseColorAndPoints();
      if(toSave.substr(toSave.length()-
1)=="t"){
         readPointCloud(toSave);
      }else if(toSave.substr(toSave.length()-
1)=="y"){
         readPlyFormat(toSave);
      }
   }else{
      cout<<"directory not selected for
saving"<<endl;
   }



//    if(filename!=""){
//
variables::toReadCustomFileName=filena
me.toUtf8().constData();
//       eraseColorAndPoints();
//
readPointCloud(variables::toReadCustomFi
leName);
//       cout<<"point cloud read"<<endl;
//       cout<<"point 3dxyz size i.e. No of
slices::"<<variables::point3dxyz.size()<<en
dl;
//       cout<<"poitn 3dxyz[0] size i.e. no of
points per
slice"<<variables::point3dxyz[0][0].size()<<
endl;
//    }


   //for ply formate
//    QString filename=
QFileDialog::getOpenFileName(this,
tr("Open File"),"",tr("*.ply"));
//    if(filename!=""){
//
variables::toReadCustomFileName=filena
me.toUtf8().constData();
//       eraseColorAndPoints();
//
readPlyFormat(variables::toReadCustomFil
eName);
//       cout<<"point cloud read"<<endl;
//       cout<<"point 3dxyz size i.e. No of
slices::"<<variables::point3dxyz.size()<<en
dl;
//       cout<<"poitn 3dxyz[0] size i.e. no of
points per
slice"<<variables::point3dxyz[0][0].size()<<
endl;
//    }

   //for pictures


   if(filename!=""){
      variables::isRead=true;
      cout<<"file selected"<<endl;
   }else{
```

```
        cout<<"file not selected"<<endl;
        variables::isRead=false;
    }


cout<<"fileNameis::"<<variables::toReadCustomFileName<<endl;
    cout<<"isRead::"<<isRead<<endl;
}

void MainWindow::on_actionSave_triggered()
{
    cout<<"save clicked"<<endl;
//    QFileDialog dialog(this);
//    dialog.setFileMode(QFileDialog::AnyFile);
//    dialog.setNameFilter(tr("Images (*.ply *.txt)"));
    QString filename="a.ply";
//    dialog.selectFile(filename);


    //QString filename= QFileDialog::getOpenFileName(this, tr("Open File"),"",tr("(*.txt *.ply)"));
    string toSave=filename.toUtf8().constData();


    cout<<"to save::"<<toSave<<endl;

    if(toSave!=""){
        if(toSave.substr(toSave.length()-1)=="t"){

writePointCloud(toSave+variables::customExtension,variables::colorYesNo);
        }else if(toSave.substr(toSave.length()-1)=="y"){

writePlyPointCloud(toSave+".ply",variables::colorYesNo);
        }
    }else
        cout<<"directory not selected for saving"<<endl;

//    QString dir = QFileDialog::getExistingDirectory(this, tr("Open Directory"),
//                                                    "/home",
//                                                    QFileDialog::ShowDirsOnly
//                                                    | QFileDialog::DontResolveSymlinks);
//    string toSave=dir.toUtf8().constData();
//    variables::toWriteFileName=toSave;
//    string type;

//    if(toSave!=""){
//        cout<<"driectory selected for saving"<<endl;
//        if(type==".ply"){
//            writePlyPointCloud(variables::toWriteFileName+".ply",variables::colorYesNo);
//        }else if(type==variables::customExtension){
//            writePlyPointCloud(variables::toWriteFileName+variables::customExtension,variables::colorYesNo);
//        }else if(type=="pictures"){
//            //later
//        }
//    }else{
//        cout<<"directory not selected for saving"<<endl;
//    }


}

void MainWindow::on_actionStart_triggered()
{
    cout<<"start clicked"<<endl;

if(variables::SerialCommunicationStatus==false && 0){
```

```
      cout<<"can't start because it not
connected"<<endl;
      variables::test=true;
   }
   else if(1){//change it to else later on
      cout<<"starting";
      asdf::startScan();



      //alternative code for testing
      /*
      asdf::camero cap;
      cap.init(0);
      variables::rootPath="C:\\Users\\Fahad
Mubeen\\Desktop\\aaa\\icube";
      string path;
      variables::colorYesNo;
      long double theta=0;
      long double inc=2.1176;
      //cap.getSnap("a.png");
      for (int i=1;i<50;i++){
         cout<<"analyzing frame::"<<i<<endl;

cap.getSnap(variables::rootPath+to_string(
i)+variables::extensionImage);
         cout<<"point cloud adding"<<endl;

asdf::addPointCloud("C:\\Users\\Fahad
Mubeen\\Desktop\\aab\\icube"+to_string
(i)+variables::extensionImage,theta+=inc);
      }
      cout<<"loop ended"<<endl;
//*/

   }

}

void
MainWindow::on_actionRotate_triggered(
)
{
   cout<<"OnRotate clicked"<<endl;
   variables::isRotateByMouse =
!variables::isRotateByMouse;
}

void
MainWindow::on_actionKill_triggered()
{
   cout<<"on action kill"<<endl;
}

void
MainWindow::on_actionMove_triggered()
{

}

void
MainWindow::on_actionOutzoom_trigger
ed()
{
   variables::isMinusZoomButtonClicked =
!variables::isMinusZoomButtonClicked;
}

void
MainWindow::on_actionFreely_triggered()
{
   variables::rotate=!variables::rotate;
}

void
MainWindow::on_actionSave_to_triggere
d()
{

}

void
MainWindow::on_actionSave_Screenshot_
triggered()
{

}

void
MainWindow::on_actionRecent_Scans_tri
ggered()
{

}
```

```
void
MainWindow::on_actionQuit_triggered()
{
   cout<<"on Quit clicked"<<endl;
}

void
MainWindow::on_actionFront_triggered()
{

}

void
MainWindow::on_actionLeft_triggered()
{

}

void
MainWindow::on_actionBack_triggered()
{

}

void
MainWindow::on_actionTop_triggered()
{

}

void
MainWindow::on_actionBottom_triggered
()
{

}

void
MainWindow::on_actionIsometric_2_trigg
ered()
{

}
```

```
void
MainWindow::on_actionAbout_eyeCube_t
riggered()
{

}

void
MainWindow::on_actionAbout_US_trigger
ed()
{

}

void
MainWindow::on_actionDeveloper_Websi
te_triggered()
{

}

void
MainWindow::on_stepperRotation_activat
ed(int index)
{

   if(index==custom){
      variables::stepperRotation=custom;
      ui->dMotorStep->setEnabled(true);
   }else if(index==half){
      variables::stepperRotation=half;
      ui->dMotorStep->setEnabled(false);
   }else if(index==full){
      variables::stepperRotation=full;
      ui->dMotorStep->setEnabled(false);
   }
   cout<<"stepperROtation clicked
stepperRotation::"<<variables::stepperRot
ation<<endl;
}

void
MainWindow::on_advanceoptions_clicked
()
{
   if(ui->advanceoptions->isChecked())
   {
```

```
        ui->slaserslider->setEnabled(true);
        ui->slasercombo->setEnabled(true);
        ui->scanangle->setEnabled(true);
    }
    if(!ui->advanceoptions->isChecked())
    {
        ui->slaserslider->setEnabled(false);
        ui->slasercombo->setEnabled(false);
        ui->scanangle->setEnabled(false);
    }
}


void MainWindow::on_lights_3_clicked(){}
//remove this in future my f

void
MainWindow::on_motorOnOff_clicked()
{

variables::motorStatus=!variables::motorSt
atus;
   cout<<"motoronoff clicked
motorStatus::"<<variables::motorStatus<<
endl;

if(variables::SerialCommunicationStatus){
      if(variables::stepperMode==half){

serialo::writeToSerial(variables::serialMoto
rHalf);
          //sending noOfSteps
          string
noOfSteps=to_string(variables::motorSteps
s).append((to_string(variables::motorSteps
).size()<2)?"0":"");
          cout<<"steps custom writing to
serial::"<<variables::serialMotorCustom+n
oOfSteps<<endl;
          serialo::writeToSerial(noOfSteps);
      }else if(variables::stepperMode==full){
          cout<<"steps half writing to
serial::"<<variables::serialMotorHalf<<endl
;

serialo::writeToSerial(variables::serialMoto
rFull);
      }else
if(variables::stepperMode==custom){
          cout<<"steps full writing to
serial::"<<variables::serialMotorFull<<endl;

serialo::writeToSerial(variables::serialMoto
rCustom);
      }
   }
}

void
MainWindow::on_quality_scan_activated(i
nt index)
{
   if(index==high_){
      variables::stepperMode=high_;
      variables::noOfSlicesToScan=highScan;

variables::thetaInc=variables::thetaSmallIn
c;
   }else if(index==medium_){
      variables::stepperMode=medium_;

variables::noOfSlicesToScan=mediumScan;

variables::thetaInc=variables::thetaMediu
mInc;
   }else if(index==low_){
      variables::noOfSlicesToScan=lowScan;
      variables::stepperMode=low_;

variables::thetaInc=variables::thetaLargeIn
c;
   }
   cout<<"on quality scan clicked
stepperMode::"<<variables::stepperMode
<<endl;
}

void
MainWindow::on_comboBox_2_activated(
int index){}//remove this in future my
friends
```

```
void
MainWindow::on_colorBox_activated(int
index)
{
   if(index==0){
      variables::colorStatus=colorYes;
      variables::colorYesNo=true;
   }else if(index==1){
      variables::colorStatus=colorNo;
      variables::colorYesNo=false;
   }
   cout<<"color status clicked
status::"<<variables::colorStatus<<endl;

cout<<"colorYesNo::"<<variables::colorYes
No<<endl;
}

void
MainWindow::on_advanceoptions_clicked
(bool checked)
{
   if(checked==true){

   }else if(checked==false){

   }
}

void
MainWindow::on_scanangle_valueChange
d(int arg1)
{
   variables::scanAngle=arg1;
}


// 11beerbhatti rizwan
//
////////////////////////////////////////////
/////////////////////
void MainWindow::keyPressEvent(
QKeyEvent *k )
   {

     QString a=k->text();

variables::keyboardEvent_isKeyPressed =
true;
     string temp = a.toUtf8().constData();
#if bhatti
     cout<<temp<<endl;
#endif
     variables::keyboardEvent_key =
temp[0];
   }


void
MainWindow::wheelEvent(QWheelEvent
*event)
{
   QPoint a=event->angleDelta();
   variables::scrollEvent_yIntensity = a.ry();
   variables::scrollEvent_isScrolled = true;
}
//
////////////////////////////////////////////
/////////////////////

void
MainWindow::on_actionInzoom_triggered
()
{
   variables::isPlusZoomButtonClicked = !
variables::isPlusZoomButtonClicked;
}

void
MainWindow::on_slasercombo_customCo
ntextMenuRequested(const QPoint &pos)
{

}

void
MainWindow::on_slasercombo_valueChan
ged(int arg1)
{
   variables::laserIntensity=arg1;
}
```

```
void
MainWindow::on_spinBox_valueChanged(i
nt arg1)
{

}

void
MainWindow::on_dlaserCombo_valueCha
nged(int arg1)
{
   cout<<"laser diagnostic intensity
changed"<<arg1<<endl;
   if(variables::diganosticsStatus==true &&
variables::SerialCommunicationStatus &&
variables::laserStatus){
      cout<<"chainging laser
intensity"<<endl;

serialo::writeToSerial(variables::laserLine1I
ntensity(arg1));

serialo::writeToSerial(variables::laserLine2I
ntensity(arg1));
   }

}

void
MainWindow::on_diagnosticsStart_clicked
(bool checked)
{
   variables::diganosticsStatus=true;
   cout<<"dignostic start clicked
diagonistics
status::"<<variables::diganosticsStatus<<e
ndl;
}

void
MainWindow::on_dMotorStep_valueChan
ged(int arg1)
{
   variables::motorSteps=arg1;
}
```

```
void
MainWindow::on_tabWidget_currentChan
ged(int index)
{
   if(index==0){
      variables::webcamViewOn=true;
      variables::diganosticsStatus=1;
      cout<<"diagnostic tab"<<endl;
   }else{
      variables::webcamViewOn=false;
      variables::diganosticsStatus=0;
      cout<<"scan tab"<<endl;
   }
}
```

Mydialog.cpp

```
#include "mydialog.h"
#include "ui_mydialog.h"
#include <QString>
#include <QFileDialog>
#include <QDebug>
#include<iostream>
#include<string>
//#include"variables.h"
using namespace std;

MyDialog::MyDialog(QWidget *parent) :
   QDialog(parent),
   ui(new Ui::MyDialog)
{
   ui->setupUi(this);
}

MyDialog::~MyDialog()
{
   delete ui;
}
//when dialog box is cliced
void MyDialog::on_pushButton_clicked()
{
   cout<<"I am here brothers "<<endl;
   QString filename=
QFileDialog::getOpenFileName(this,
tr("Open File"),"",tr("*.txt"));
   toSave=filename.toUtf8().constData();
   cout<<"to save::"<<toSave<<endl;
   if(toSave!=""){
      if(ui->customformat->isChecked())
      {
         cout<<"custom formate
saving"<<endl;
         forDialogBox="custom";

      }
      else if(ui->plyformat->isChecked())
      {
         //Do anything here for custom
format
         cout<<"ply formate saving"<<endl;
         forDialogBox=".ply";

      }
   }else{
      forDialogBox="";
   }
   this->close();

}
```

Mainwindow.h

```
#pragma once
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QKeyEvent>
#include <string>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
   Q_OBJECT

public:
   explicit MainWindow(QWidget *parent
= 0);
   ~MainWindow();

   void keyPressEvent( QKeyEvent *k );
   void wheelEvent(QWheelEvent *event);

private slots:
   void on_pushButton_clicked();

   void on_lights_clicked();

   void on_laser_clicked();

   void on_stepperClockWise_clicked();

   void on_stepperAntiClockwise_clicked();

   void on_diagnosticsStart_clicked();

   void on_diagnosticsStop_clicked();

   void on_numberOfSteps_textChanged();

   void on_laserSlider_sliderMoved(int
position);

   void on_stepperMode_activated(int
index);

   void
on_stepperRotation_activated(const
QString &arg1);

   void on_laserSlider_valueChanged(int
value);

   void on_lineEdit_returnPressed();

   void on_actionConnect_triggered();

   void on_actionLoad_triggered();

   void on_actionSave_triggered();

   void on_actionStart_triggered();

   void on_actionRotate_triggered();

   void on_actionKill_triggered();

   void on_actionMove_triggered();

   void on_actionOutzoom_triggered();

   void on_actionFreely_triggered();

   void on_actionSave_to_triggered();

   void
on_actionSave_Screenshot_triggered();

   void
on_actionRecent_Scans_triggered();

   void on_actionQuit_triggered();

   void on_actionFront_triggered();

   void on_actionLeft_triggered();

   void on_actionBack_triggered();

   void on_actionTop_triggered();

   void on_actionBottom_triggered();
```

```
    void on_actionIsometric_2_triggered();

    void
on_actionAbout_eyeCube_triggered();

    void on_actionAbout_US_triggered();

    void
on_actionDeveloper_Website_triggered();

    void on_stepperRotation_activated(int
index);

    void on_advanceoptions_clicked();

    void on_lights_3_clicked();

    void on_motorOnOff_clicked();

    void on_quality_scan_activated(int
index);

    void on_comboBox_2_activated(int
index);

    void on_colorBox_activated(int index);

    void on_advanceoptions_clicked(bool
checked);

    void on_scanangle_valueChanged(int
arg1);

    void on_actionInzoom_triggered();

    void
on_slasercombo_customContextMenuReq
uested(const QPoint &pos);

    void on_slasercombo_valueChanged(int
arg1);

    void on_spinBox_valueChanged(int
arg1);

    void on_dlaserCombo_valueChanged(int
arg1);

    void on_diagnosticsStart_clicked(bool
checked);

    void on_dMotorStep_valueChanged(int
arg1);

    void on_tabWidget_currentChanged(int
index);

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

Matlabfunctions.h

```
#pragma once
#ifndef MATLABFUNCTIONS_H
#define MATLABFUNCTIONS_H

#include
<opencv2/objdetect/objdetect.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include "opencv2/imgproc/imgproc.hpp"

using namespace cv;
#include<fstream>
#include<vector>
#include<ctime>
#include<iostream>
using namespace std;
#include"variables.h"
//using namespace variables;


namespace asdf {
//Contants for testing, and for general
parameters which remains same throught
out the program
#define debug_ 0
# define M_PI
3.14159265358979323846  /* pi */
#define l_ 1
#define w_ 201
#define sigma 6
#define binaryThreshold 100
#define horizontalResolution 1279
#define verticalResolution 719
#define twidth  1279
#define theight  719
#define centero  639
#define TotalNoOfPointsPerSlice 720
#define noOfSlices 20

   //vector<vector<vector<long double>>>
variables::point3dxyz;
   //vector<vector<vector<uchar>>>
variables::colour;
//Functions for finding the central laser
line after processing
   vector<int> redLine(Mat &img , int
horizontalResolution_ , int
verticalResolution_) {
#if printDebug

//cout<<"horizontalResolution::"<<horizon
talResolution_<<endl;

//cout<<"verticalResolution::"<<verticalRe
solution_<<endl;
#endif
      Mat fliped;
      flip(img, fliped, 1);
      vector<int> laserLineLocation;
#if printDebug
      //cout<<"redLine starting"<<endl;
#endif
      for (int i = 0; i < verticalResolution_;
i++) {
         //cout << "iteration:" << i << endl;
#if printDebug
      //cout<<"iteration:" << i << endl;
#endif
         Rect r = Rect(0, i,
horizontalResolution_-1, 1);
         Mat imgCrop;
         Mat fimgCrop;
         img(r).copyTo(imgCrop);
         fliped(r).copyTo(fimgCrop);
         //Mat imgCrop =
afterFilter(r).clone();
         double min, max, min_2, max_2;
         Point minLoc, maxLoc, minLoc_2,
maxLoc_2;
         //minMaxLoc
         minMaxLoc(imgCrop, &min, &max,
&minLoc, &maxLoc);
         // //minMaxLoc(secondDerivative,
&min, &max);
         //cout << "[" << i << "]::" << max <<
"::" << "xLoc:" << maxLoc.x << ":yLoc:" <<
maxLoc.y << endl;
         minMaxLoc(fimgCrop, &min_2,
&max_2, &minLoc_2, &maxLoc_2);
         int maxLoc_2nd =
horizontalResolution_ - maxLoc_2.x;
```

```
        int laserLineLoc =
floor((double(maxLoc.x) +
double(maxLoc_2nd)) / 2.0f);

laserLineLocation.push_back(laserLineLoc);
      }
      return laserLineLocation;
   }

   //Function for applying filters on the
image so laser line be extracted later on, it
applies an inverted gussian filter
   vector<int> imageProcess(Mat &img) {
#if printDebug
      cout<<"starting function"<<endl;
#endif
      vector<int> output;
      cv::Mat planes[3];
      split(img, planes);  // planes[2] is the
red channel
      cv::Mat bin;
      //   adaptiveThreshold(image, image,
255, CV_ADAPTIVE_THRESH_MEAN_C,
CV_THRESH_BINARY, 75, 10);
      threshold(planes[2], bin,
binaryThreshold, 255, THRESH_BINARY);
      Mat vKernalGuassian =
getGaussianKernel(w_, sigma, CV_64F);
      Mat kernelGuassian;
      transpose(vKernalGuassian,
kernelGuassian);
      Mat m = kernelGuassian;

      //using gradient of matlab
      Mat forConv = Mat::ones(1, 3,
CV_64F);
      forConv.at<double>(0, 0) = 0.5f;
      forConv.at<double>(0, 1) = 0.0f;
      forConv.at<double>(0, 2) = -0.5f;

      //frist derivative
      Mat firstDerivativeGuassian,
secondDerivativeGuassian;
      filter2D(kernelGuassian,
firstDerivativeGuassian, -1, forConv,
Point(-1, -1), 0, BORDER_DEFAULT);
      //2nd derivatives
      filter2D(firstDerivativeGuassian,
secondDerivativeGuassian, -1, forConv,
Point(-1, -1), 0, BORDER_DEFAULT);

      //using gradient ends
      //max value is 89 :0.000793662
      double min, max;

minMaxLoc(secondDerivativeGuassian,
&min, &max);

      //normalizeing the 2nd order
derivative
      Mat
normalized2ndOrderGussainDerivative =
secondDerivativeGuassian * (1.0f / max);
      //normalize(secondDerivative,
normalized2ndOrder, 0, 1,
NORM_MINMAX, -1);

      Mat
normalized2ndOrderGussianDerivativeInve
rted =
normalized2ndOrderGussainDerivative * -
1.0f;

      Mat afterFilter;
      filter2D(bin, afterFilter, -1,
normalized2ndOrderGussianDerivativeInve
rted, Point(-1, -1), 0, BORDER_DEFAULT);
#if printDebug
      cout<<"calling red line"<<endl;
#endif
      //finding the laser line after filters
      return
redLine(afterFilter,afterFilter.cols,afterFilte
r.rows);
   }

   //for conveting angle to radian
   long double toRad(long double angle) {
      return angle*M_PI / 180.0f;
   }


//functions for calculating the detpth of a
point
```

```
   vector<long double>
findDx(vector<int>&px, int
horizontalResolution_) {
      //variables having information about
the assembly and camera parameters
      long double betaTwice = 62;
      long double beta = betaTwice / 2.0;
      long double alpha = 45;
      long double pxc =
horizontalResolution_ / 2;
      //36
      long double dc = variables::distance;
      long double ratio = tan(toRad(alpha)) /
tan(toRad(beta));
      //dx=(px*dc)./(pxc*ratio+px);
      vector<long double> dx;
      for (int i = 0; i < px.size(); i++) {
         long double x = (px[i] * dc) /
(pxc*ratio + px[i]);
         dx.push_back(x);
      }
      0;
      static int bal = 3;//for testing
      bal++;
      return dx;
   }
//funcitons for finding the height of the
points
   vector<long double> findDy(vector<long
double> h , long double hc) {
      //variables having information about
the assembly and camera parameters
      vector<long double> output;
      //long double hc = 720;
      long double angle = toRad(36 / 2);

      long double d = variables::distance;
      long double ypc = hc / 2.0f;
      vector<int> yDashP;
      for (int i = 0; i < h.size(); i++) {
         yDashP.push_back(i - hc / 2.0f);
         long double o = ((yDashP[i] * (d -
h[i]))*tan(angle)) / ypc;
         output.push_back(o);
      }
      return output;
   }

   //creating theta array for new scan
   vector<long double> theta;
   void createTheta() {
      for (int i = 0; i*2.1176<360; i++) {
         theta.push_back((2.1176*i));
      }
   }

   //get color for a single slice
   vector<vector<uchar>>
getColorPointCloudSlice(Mat &img,
vector<int>&laserLine) {
      cv::Mat planes[3];
      split(img, planes);  // planes[2] is the
red channel
                  //slice no, point no, color
      vector<vector<uchar>> colour;

      for (int i = 0; i < laserLine.size(); i++) {
         vector<uchar> onePoint;

onePoint.push_back(planes[2].at<uchar>(i,
laserLine[i]));//red

onePoint.push_back(planes[1].at<uchar>(i,
laserLine[i]));//green

onePoint.push_back(planes[0].at<uchar>(i,
laserLine[i]));//blue
         colour.push_back(onePoint);
      }
      return colour;
   }
//for print mat(an image format used by
OpenCV)
   void printMat(Mat &m, string s = ",") {
      for (int i = 0; i < m.cols; i++) {
         cout << i << ":" << m.at<double>(0,
i) << "\t ";
         if (i % 3 == 0) {
            cout << endl;
         }
      }
   }

   //for converting mat to vector
   template<class type>
```

```
   vector<vector<type>> matToVector(Mat
&m) {
      vector<vector<type>> v;
      for (int i = 0; i < m.rows; i++) {
         vector<type> row;
         for (int j = 0; j < m.cols; j++) {
            type x = m.at<type>(i, j);
            row.push_back(x);
         }
         v.push_back(row);
      }
      return v;
   }


   //for conveting vector to Mat
   template<class type>
   Mat
vectorToMat(vector<vector<type>>&v) {
      Mat m(v.size(), v[0].size(), CV_8U);
      for (int i = 0; i < v.size(); i++) {
         for (int j = 0; j < v[0].size(); j++) {
            m.at<uchar>(i, j) = v[i][j];
         }
      }
      return m;
   }



//v/////////////////////////////////////////
/////////////////////////////////////////
////////
//function for creating a 3D point cloud,
and color by reading images
   void OpenCVCode() {

      createTheta();
      //point cloud sorage

      /*
      long double
variables::point3dxyz[noOfSlices][TotalNo
OfPointsPerSlice][3];


      long double
point3dCylinderical[noOfSlices][TotalNoOf
PointsPerSlice][3];
      */

      //1) slice no 2) point no 3) xyz or r-z-
theta

      auto point3dCylinderical =
variables::point3dxyz;

      string rootPath = "D:\\Users\\Fahad
Mubeen\\Documents\\MATLAB\\FYP\\icu
beImages\\ObjectsScan\\thermopoleCup\
\3dReconsturctoin\\icube";
      string rootPath2 = "D:\\Users\\Fahad
Mubeen\\Documents\\MATLAB\\FYP\\icu
beImages\\ObjectsScan\\thermopoleCup\
\ColorMapping\\icube";
      string extension = ".png";
      string extension2 = ".jpg";
      int startingNo = 1;//the starting image
number

      for (int i = 0; i < noOfSlices; i++) {
         cv::Mat image;
         cv::Mat image2;//2 is for color
         string path = rootPath + to_string(i +
1) + extension;
         string path2 = rootPath2 +
to_string(i + 1) + extension2;//for color
         image = imread(path,
CV_LOAD_IMAGE_COLOR);
         image2 = imread(path2,
CV_LOAD_IMAGE_COLOR);//for color
         clock_t begin = clock();
         vector<int> laserLine =
imageProcess(image);//find the laser line
[correct]
         clock_t end = clock();
          double elapsed_secs = double(end
- begin) / CLOCKS_PER_SEC;

cout<<"time::"<<elapsed_secs<<endl;

variables::colour.push_back(getColorPoint
CloudSlice(image2, laserLine));
```

```
        for (int p = 0; p < laserLine.size();
p++) {
            laserLine[p] = abs(laserLine[p] -
centero);
        }



        vector<long double>depthR =
findDx(laserLine , image.cols);
        vector<long double>height =
findDy(depthR , image.rows);



        //after getting laser vector
        vector<vector<long double>>
slice;//constructing one slice [3][no of
slices]
        slice.push_back(depthR);
        slice.push_back(height);
        //      vector<long double> theta;
        //      for (long double j = 0; j
<TotalNoOfPointsPerSlice; j++)
{//generating theta
        //          theta.push_back(360.0f /
noOfSlices*i);
        //      }
        slice.push_back(theta);

point3dCylinderical.push_back(slice);//pus
hing slice into 3d cylinderical coordinates

                                    //the x y z
coordinates
        vector<vector<long double>>
sliceXYZ;
        vector<long double> x;
        vector<long double> y;
        for (int j = 0; j < image.rows; j++) {


            long double xCord = depthR[j] *
cos(toRad(theta[i]));
            long double yCord = depthR[j] *
sin(toRad(theta[i]));
            //long double xCord = depthR[j] *
cos(toRad(360.0f / noOfSlices*double(j)));
            //long double yCord = depthR[j] *
sin(toRad(360.0f / noOfSlices*double(j)));
            // //// cout << "[" << j << "]::" <<
xCord <<":ycord::"<<yCord<< endl;
            if (j == 450) {
               int x = 3;
               int y = 3;
               x = y;
            }
            x.push_back(xCord);
            y.push_back(yCord);
         }
         sliceXYZ.push_back(x);//contring xyz
slice
         sliceXYZ.push_back(y);
         sliceXYZ.push_back(height);

variables::point3dxyz.push_back(sliceXYZ);
//pushing to 3d coordinate system



      }
      //   std:://// cout<<"program
ends"<<std::endl;

   }



   //
//////////////////////////////////////////
////
   /// \brief readFromFile
   /// \param fileName
   /// \return
   /// to read data from a file

   vector<string> readFromFile(string
fileName) {
      vector<string> v;
      ifstream toRead(fileName);
      string str;
      if (!toRead.is_open()) {
         cout << "file not open (read from
file)" << endl;
      }
      while (getline(toRead, str)) {
         if (v.size() % 10000 == 0) {
```

```
            cout << v.size() << endl;
          }
          v.push_back(str);
        }
        toRead.close();
        return v;
    }
    ///
    /// \brief writeToFile
    /// \param fileName
    /// \param v
    ///to write data to a file
    void writeToFile(string fileName,
vector<string>&v) {
        ofstream out(fileName);
        if (!out.is_open()) {
          cout << "file not open (write to file)"
<< endl;
        }
        for (int i = 0; i < v.size(); i++) {
          if (i % 10000 == 0) {
            cout << i << endl;
          }
          out << v[i] << endl;
        }
        out.close();
    }

    ///
    /// \brief The MyPoint class
    ///the my point class constains functiosn
which are used on points, it is also used in
importing and exporting data
    /// to other formats, it manages the
color, and position of 3D poitns
    class MyPoint {
    public:
        bool colorFlag = true;
        unsigned char color[3];
        long double point[3];

        void putDataByString(string str, char
sep = ',') {
          int j = 0;
          // separated by commas
          int start = 0;
          for (int i = 0; i < str.length(); i++) {
            if (str[i] == sep) {
              string sStr = str.substr(start, i -
start);
              if (j < 3) {
                point[j++] = stold(sStr);
              }
              else {
                if (colorFlag == true)
                  color[j++ - 3] = unsigned
char(stoi(sStr));
              }
              start = i + 1;
            }
          }
        }
        void putData(long double x, long
double y, long double z, unsigned char red
= 0, unsigned char green = 0, unsigned
blue = 0) {
          point[0] = x;
          point[1] = y;
          point[2] = z;
          color[0] = red;
          color[1] = green;
          color[2] = blue;
        }

        string toStringData(string sep = ",") {
          string str;
          if (colorFlag == true)
            str = to_string(point[0]) + sep +
to_string(point[1]) + sep +
to_string(point[2]) + sep +
to_string(int(color[0])) + sep +
to_string(int(color[1])) + sep +
to_string(int(color[2])) + sep;
          else
            str = to_string(point[0]) + sep +
to_string(point[1]) + sep +
to_string(point[2]) + sep;
#if debug_
          cout << "str is::" << str << endl;
          cout << "color 1 and 2 is::" << sep +
to_string(int(color[0])) +
to_string(int(color[1])) << endl;
          cout << "color end::" << sep +
to_string(int(color[2])) + sep << endl;
```

```
#endif
        return str;
      }


   };

   ///
   /// \brief writePlyPointCloud
   /// \param fileName
   /// \param colorFlag
   ///for exporting the data to ply formate
which can be used by mesh lab and other
softwares
   void writePlyPointCloud(string fileName,
bool colorFlag = true) {
      vector<string> toWrite;
      toWrite.push_back("ply");
      toWrite.push_back("format ascii 1.0");
      toWrite.push_back("element vertex "
+
to_string(variables::noOfPointsPerSliceTo
Write*variables::noOfSlicesToWrite));
      toWrite.push_back("property float x");
      toWrite.push_back("property float
y");
      toWrite.push_back("property float z");
      if (colorFlag) {
         toWrite.push_back("property uchar
red");
         toWrite.push_back("property uchar
green");
         toWrite.push_back("property uchar
blue");
      }
      toWrite.push_back("end_header");

      for (int i = 0; i <
variables::noOfSlicesToWrite; i++) {
         for (int j = 0; j <
variables::noOfPointsPerSliceToWrite; j++)
{
            MyPoint p;
            p.colorFlag = colorFlag;
            if (colorFlag == true)

p.putData(variables::point3dxyz[i][0][j],
variables::point3dxyz[i][1][j],
variables::point3dxyz[i][2][j],
variables::colour[i][j][0],
variables::colour[i][j][1],
variables::colour[i][j][2]);
            else

p.putData(variables::point3dxyz[i][0][j],
variables::point3dxyz[i][1][j],
variables::point3dxyz[i][2][j]);
            string stro = p.toStringData(" ");
            toWrite.push_back(stro);
         }
      }
      writeToFile(fileName, toWrite);
   }

   ///
   /// \brief readPlyFormat
   /// \param fileName
   /// \param colorFlag
   ///for reading ply formate
   void readPlyFormat(string fileName,
bool colorFlag = true) {
      //reading from file
      vector<string> toRead =
readFromFile(fileName);
      vector<MyPoint> v;

      int i = 0;
      if (toRead[6] == "property uchar red")
{
         colorFlag = true;
         i = 10;
      }
      else {
         colorFlag = false;
         i = 7;
      }
      variables::colorYesNo = colorFlag;
      for (; i < toRead.size(); i++) {
         if (i % 10000 == 0) {
            cout << "readply::" << i << endl;
         }
         MyPoint p;
         p.colorFlag = colorFlag;
         p.putDataByString(toRead[i], ' ');
```

```
            v.push_back(p);
        }
        variables::noOfSlicesToWrite =
v.size();
        variables::noOfPointsPerSliceToWrite
= 1;
        for (int i = 0, k = 0; i <
variables::noOfSlicesToWrite; i++) {
            if (i % 10000 == 0) {
                cout << "readply::" << i << endl;
            }
            vector<long double> x;
            vector<long double> y;
            vector<long double> height;
            vector<vector<long double>> slice;
            vector<vector<uchar>> colorOfSlice;
            for (int j = 0; j <
variables::noOfPointsPerSliceToWrite; j++,
k++) {
                x.push_back(v[k].point[0]);
                y.push_back(v[k].point[1]);
                height.push_back(v[k].point[2]);
                if (colorFlag) {
                    vector<uchar> onePoint;

onePoint.push_back(v[k].color[0]);

onePoint.push_back(v[k].color[1]);

onePoint.push_back(v[k].color[2]);

colorOfSlice.push_back(onePoint);
                }
            }
            slice.push_back(x);
            slice.push_back(y);
            slice.push_back(height);

variables::point3dxyz.push_back(slice);
            if (colorFlag)

variables::colour.push_back(colorOfSlice);
        }
    }

    ///
    /// \brief writePointCloud
    /// \param fileName
    /// \param colorFlag
    ///for writing poitn cloud to our custom
formate
    void writePointCloud(string fileName,
bool colorFlag = true) {
        vector<string> toWrite;

toWrite.push_back(to_string(variables::col
orYesNo));

toWrite.push_back(to_string(variables::no
OfSlicesToWrite));

toWrite.push_back(to_string(variables::no
OfPointsPerSliceToWrite));

        for (int i = 0, k = 0; i <
variables::noOfSlicesToWrite; i++) {
            for (int j = 0; j <
variables::noOfPointsPerSliceToWrite; j++,
k++) {
                if (k % 10000 == 0) {
                    cout << "write custom::" << k
<< endl;
                }
                MyPoint p;
                p.colorFlag = colorFlag;
                if (colorFlag == true)

p.putData(variables::point3dxyz[i][0][j],
variables::point3dxyz[i][1][j],
variables::point3dxyz[i][2][j],
variables::colour[i][j][0],
variables::colour[i][j][1],
variables::colour[i][j][2]);
                else

p.putData(variables::point3dxyz[i][0][j],
variables::point3dxyz[i][1][j],
variables::point3dxyz[i][2][j]);

                string stro = p.toStringData();
                toWrite.push_back(stro);
            }
        }
        writeToFile(fileName, toWrite);
```

```
    }
    ///
    /// \brief readPointCloud
    /// \param fileName
    ///for reading point cloud from our
custom formate
    void readPointCloud(string fileName) {
        //reading from file
        vector<string> toRead =
readFromFile(fileName);
        vector<MyPoint> v;
        variables::colorYesNo =
stoi(toRead[0]);
        bool colorFlag = variables::colorYesNo;
        variables::noOfSlicesToWrite =
stoi(toRead[1]);
        variables::noOfPointsPerSliceToWrite
= stoi(toRead[2]);
        for (int i = 3; i < toRead.size(); i++) {
            MyPoint p;
            p.colorFlag = colorFlag;
            p.putDataByString(toRead[i]);
            v.push_back(p);
        }


        for (int i = 0, k = 0; i <
variables::noOfSlicesToWrite; i++) {
            vector<long double> x;
            vector<long double> y;
            vector<long double> height;
            vector<vector<long double>> slice;
            vector<vector<uchar>> colorOfSlice;
            for (int j = 0; j <
variables::noOfPointsPerSliceToWrite; j++,
k++) {
                x.push_back(v[k].point[0]);
                y.push_back(v[k].point[1]);
                height.push_back(v[k].point[2]);
                if (colorFlag) {
                    vector<uchar> onePoint;

onePoint.push_back(v[k].color[0]);

onePoint.push_back(v[k].color[1]);

onePoint.push_back(v[k].color[2]);
colorOfSlice.push_back(onePoint);
                }
            }
            slice.push_back(x);
            slice.push_back(y);
            slice.push_back(height);

variables::point3dxyz.push_back(slice);
            if (colorFlag)

variables::colour.push_back(colorOfSlice);
        }
    }
    ///
    /// \brief erasePointAndColor
    ///for erasing point cloud so new scan
can be performed
    void erasePointAndColor() {

variables::point3dxyz.erase(variables::poin
t3dxyz.begin(),
variables::point3dxyz.end());

variables::colour.erase(variables::colour.be
gin(), variables::colour.end());
    }
    //
//////////////////////////////////////////
////////////


    ///
    /// \brief addPointCloud
    /// \param laserPath
    /// \param theta
    /// \param colorImagePath
    ///add new points to the existing point
cloud
    void addPointCloud(string laserPath,
long double theta,string
colorImagePath="") {
        Mat image = imread(laserPath,
CV_LOAD_IMAGE_COLOR);
#if printDebug
    cout<<"image process starting"<<endl;
#endif
```

```
      vector<int> laserLine =
imageProcess(image);
#if printDebug
   cout<<"image process complete"<<endl;
#endif
      if (variables::colorYesNo) {
#if printDebug
   cout<<"color adding"<<endl;
#endif
         Mat image2 =
imread(colorImagePath,
CV_LOAD_IMAGE_COLOR);
#if printDebug
   cout<<"color added"<<endl;
#endif


variables::colour.push_back(getColorPoint
CloudSlice(image2, laserLine));
      }

      for (int p = 0; p < laserLine.size(); p++)
{
         laserLine[p] = abs(laserLine[p] -
centero);
      }


      vector<long double>depthR =
findDx(laserLine , image.cols);
      vector<long double>height =
findDy(depthR,image.rows);
      //after getting laser vector
      vector<vector<long double>>
slice;//constructing one slice [3][no of
slices]
      vector<vector<long double>> sliceXYZ;
      vector<long double> x;
      vector<long double> y;
      for (int j = 0; j < laserLine.size(); j++) {
         long double xCord = depthR[j] *
cos(toRad(theta));
         long double yCord = depthR[j] *
sin(toRad(theta));
         x.push_back(xCord);
         y.push_back(yCord);
      }
      sliceXYZ.push_back(x);//contring xyz
slice
      sliceXYZ.push_back(y);
      sliceXYZ.push_back(height);

variables::point3dxyz.push_back(sliceXYZ);
//pushing to 3d coordinate system
   }


   //new functions for scanning
   void startScan(){
      asdf::erasePointAndColor();//erase old
point cloud and color
      variables::scanEnable=true;
      variables::scanningImageNo=0;
   }
   ///for stopping the scan
   void stopScan(){
      variables::scanEnable=false;
   }
}

using namespace asdf;



#endif // MATLABFUNCTIONS_H
```

Mycanvas.h

```
#include "qsfmlcanvas.h"
#include<SFML/OpenGL.hpp>
#include<SFML/Window.hpp>
#include<SFML/Graphics.hpp>
#include<SFML/System.hpp>

#include <iostream>
#include <string>
#include <QDir>

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
//using namespace std;
#include"matlabFunctions.h"
#include"variables.h"
```

```
//using namespace fahad;
//for roting the frame
void rotateTheFrame(bool rotate);
//for drawing the point cloud
void drawPointCloudWithColor();
// for erasing the color and poitns
void eraseColorAndPoints();
//provide keybaord controls for changing
the display of scans
void
RotateOnEvent_keyboardEvent(bool,char);
//handles the zoom in and zoom out
events
void ZoomOnEvent();
//for rations of scan with mosue
void RotateOnEvent_mouseEvent(bool,
int,int);
//to begin the scanning
void scan();

//some global variables for keeping track
of movements
bool rotateEvent = false;
float angleX = 0, angleZ = 0;
char direction = 'd';
float zoomValue = 1;
char zoomDirection = 'i';
bool zoomEvent;
int mousePosX = 0;
int mousePosY = 0;
int mousePosX_old = 0;
int mousePosY_old = 0;

class MyCanvas : public QSFMLCanvas
{
public :
   MyCanvas(QWidget* Parent, const
QPoint& Position, const QSize& Size);
   void OnInit();
   void OnUpdate();
private :
   sf::Clock myClock;
   sf::Texture myImage;
   sf::Sprite mySprite;
};
```

```
MyCanvas::MyCanvas(QWidget* Parent,
const QPoint& Position, const QSize& Size)
: QSFMLCanvas(Parent, Position, Size){}
sf::Clock Clock;
//for setting initial parameters of openGL
void MyCanvas::OnInit()
{
   cout<<"init openGL"<<endl;
      glClearDepth(1.f);
      glClearColor(0.3f, 0.3f, 0.3f, 0.f);
      glEnable(GL_DEPTH_TEST);
      glEnable(GL_NORMALIZE);
      glEnable(GL_COLOR_MATERIAL);
      glDepthMask(GL_TRUE);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluPerspective(90.f, 1.f, 1.f,
300.0f);//fov, aspect, zNear, zFar

}
int xx=0;

//for updating the frame continously
void MyCanvas::OnUpdate()
{
   //   glClearDepth(1.f);
   //   glClearColor(0.3f, 0.3f, 0.3f, 0.f);
   //   glEnable(GL_DEPTH_TEST);
   //   glEnable(GL_NORMALIZE);
   //   glEnable(GL_COLOR_MATERIAL);
   //   glDepthMask(GL_TRUE);
   //   glMatrixMode(GL_PROJECTION);
   //   glLoadIdentity();
   //   gluPerspective(90.f, 1.f, 1.f,
300.0f);//fov, aspect, zNear, zFar
   if(variables::webcamViewOn){
      window.pushGLStates();

      /*
      //window.clear();
      if(xx==1 && 0){

         cout<<"opengl setting camera
"<<endl;
         xx=0;
         glClearDepth(1.f);
         glClearColor(0.3f, 0.3f, 0.3f, 0.f);
```

```
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_NORMALIZE);
        glEnable(GL_COLOR_MATERIAL);
        glDepthMask(GL_TRUE);
        glMatrixMode(GL_PROJECTION);
        //glLoadIdentity();
        //gluPerspective(90.f, 1.f, 1.f,
300.0f);//fov, aspect, zNear, zFar
     }//*/

     glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT);//Prepare for
drawing Clear color and depth buffer
     glMatrixMode(GL_MODELVIEW);//
windowly some transformations for the
cube
     glLoadIdentity();
     static sf::Sprite sprite;
     static //asdf::camero cap;
     sf::Texture texture;
     sf::Image image;
     static int once=0;
//      if(once<5 || 0){
//         cout<<once<<endl;
//         once++;
//         variables::cap.getSnap("k.png");
//   }
        Mat a,frameRGB;

frameRGB=variables::cap.getSnap2();
        cv::cvtColor(frameRGB, a,
cv::COLOR_BGR2RGBA);
        image.create(a.cols,a.rows,a.ptr());
        texture.loadFromImage(image);
        sprite.setTexture(texture);
//      }
     window.draw(sprite);
     window.popGLStates();



//      xx=0;
  } /*else if(0){
     if(xx==0 && 0){
        cout<<"openGL setup 3d"<<endl;
        xx=1;
        glClearDepth(1.f);
        glClearColor(0.3f, 0.3f, 0.3f, 0.f);
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_NORMALIZE);
        glEnable(GL_COLOR_MATERIAL);
        glDepthMask(GL_TRUE);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(90.f, 1.f, 1.f,
300.0f);//fov, aspect, zNear, zFar
        glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT);//Prepare for
drawing Clear color and depth buffer
        glMatrixMode(GL_MODELVIEW);//
windowly some transformations for the
cube
        glLoadIdentity();
     }*/
  else{
     //window.clear();
     window.pushGLStates();
     /*if(xx==0 && 1){
        cout<<"openGL setup 3d"<<endl;
        xx=1;
        glClearDepth(1.f);
        glClearColor(0.3f, 0.3f, 0.3f, 0.f);
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_NORMALIZE);
        glEnable(GL_COLOR_MATERIAL);
        glDepthMask(GL_TRUE);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(90.f, 1.f, 1.f,
300.0f);//fov, aspect, zNear, zFar
     }*/

     //window.clear();
     // cout<<"else"<<endl;
     glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT);//Prepare for
drawing Clear color and depth buffer
     glMatrixMode(GL_MODELVIEW);//
windowly some transformations for the
cube
     glLoadIdentity();


     glTranslatef(0.f, 0.f, -200.f);
```

```
        rotateTheFrame(variables::rotate);

RotateOnEvent_keyboardEvent(variables::
keyboardEvent_isKeyPressed,
variables::keyboardEvent_key);
        ZoomOnEvent();

RotateOnEvent_mouseEvent(rotateEvent,
mousePosX,mousePosY);
        glRotatef(90, 1.f, 1.f, 0.f);
        drawPointCloudWithColor();

        window.popGLStates();
        if(variables::scanEnable){
           scan();
        }
        //   glScalef(10,10,10);

        sf::Event Event;
        while (window.pollEvent(Event))
        {
           //if (Event.type ==
sf::Event::MouseButtonPressed &&
Event.mouseButton.button ==
sf::Mouse::Middle){
           if (Event.type ==
sf::Event::MouseMoved &&
sf::Mouse::isButtonPressed(sf::Mouse::Left
))
           {
#if bhatti
              //              std::cout << "the
middle button was pressed" << std::endl;
              std::cout << "mouse x: " <<
Event.mouseButton.x << std::endl;
              std::cout << "mouse y: " <<
Event.mouseButton.y << std::endl;

#endif
              rotateEvent = true;
              mousePosX =
Event.mouseMove.x;
              mousePosY =
Event.mouseMove.y;
           }
           /*
           //}
           // Close window : exit
           //          if (Event.type ==
sf::Event::Closed)
           //              window.close();

           //          // Escape key : exit
           //          if ((Event.type ==
sf::Event::KeyPressed) && (Event.key.code
== sf::Keyboard::Left))
           //          {
           //              rotateEvent = true;
           //              direction = 'a';
           //              cout<<"LEFT KEY
DETECED"<<endl;
           //          }

           //          if ((Event.type ==
sf::Event::KeyPressed) && (Event.key.code
== sf::Keyboard::Right)){
           //              rotateEvent = true;
           //              direction = 'd';
           //              cout<<"RIGHT KEY
DETECED"<<endl;
           //          }

           //          if ((Event.type ==
sf::Event::KeyPressed) && (Event.key.code
== sf::Keyboard::Up)){
           //              rotateEvent = true;
           //              direction = 'w';
           //              cout<<"UP KEY
DETECED"<<endl;
           //          }

           //          if ((Event.type ==
sf::Event::KeyPressed) && (Event.key.code
== sf::Keyboard::Down)){
           //              rotateEvent = true;
           //              direction = 's';
           //              cout<<"DOWN KEY
DETECED"<<endl;
           //          }
```

```
        //          if ((Event.type ==sf::
Event::MouseWheelMoved) &&
(Event.mouseWheel.delta >= 0))
        //          {
        //              cout<<"WHEEL ROLL
DETECTED"<<endl;
        //              zoomEvent = true;
        //              zoomDirection = 'i';

        //          }

        //          if ((Event.type ==sf::
Event::MouseWheelMoved) &&
(Event.mouseWheel.delta <=0))
        //          {
        //              zoomEvent = true;
        //              zoomDirection = 'o';
        //          }

   //*/
      }


   }




   window.display();// Finally, display
rendered frame on screen

   //following code is not required
   /*
   // conditions for files
   //file custom loading
   if(variables::isRead==true){
      cout<<"isREad
statementSTarted"<<endl;
      variables::isRead=false;
      cout<<"erasing color and
points"<<endl;
      eraseColorAndPoints();

readPointCloud(variables::toReadCustomFi
leName);
      cout<<"point cloud read"<<endl;
      cout<<"point 3dxyz size i.e. No of
slices::"<<variables::point3dxyz.size()<<en
dl;
      cout<<"poitn 3dxyz[0] size i.e. no of
points per
slice"<<variables::point3dxyz[0][0].size()<<
endl;
   }//*/

}

// don't see code below
void drawPointCloudWithColor(){

   for (int i = 0; i <
variables::point3dxyz.size(); i++) {
      for (int j = 0; j <
variables::point3dxyz[0][0].size(); j++) {
         int scalingFactor=15;
         int s=15;
         glBegin(GL_POINTS);//draw 3d
point cloud
         if(variables::colorYesNo){

glColor3f(double(variables::colour[i][j][0])/
255.0f,double(variables::colour[i][j][1])/25
5.0f,double(variables::colour[i][j][2])/255.0
f);
         }

glVertex3f(variables::point3dxyz[i][0][j]*sc
alingFactor,variables::point3dxyz[i][1][j]*sc
alingFactor,variables::point3dxyz[i][2][j]*sc
alingFactor);
         glEnd();
      }
   }

}

//for rotatoing the frame
void rotateTheFrame(bool rotate){
   if(rotate){
      float angle;

angle=Clock.getElapsedTime().asSeconds();
      glRotatef(angle * 50, 1.f, 0.f, 0.f);
```

```
      glRotatef(angle * 30, 0.f, 1.f, 0.f);
      glRotatef(angle * 90, 0.f, 0.f, 1.f);
      //glColor3f(1,0,1);
   }
}

//for erasing the point cloud and its color
void eraseColorAndPoints(){

variables::point3dxyz.erase(variables::poin
t3dxyz.begin(),variables::point3dxyz.end())
;

variables::colour.erase(variables::colour.be
gin(),variables::colour.end());
}

//11beerbhatti
void RotateOnEvent_keyboardEvent(bool
rotate, char direction){
   glRotatef(angleX, 1.0f, 0.0f, 0.0f);
   glRotatef(angleZ, 0.0f, 1.0f, 0.0f);
   variables::keyboardEvent_isKeyPressed
= false;
   if(rotate)
   {

      switch(direction){
      case 'a':
         angleZ = angleZ - 5.0f;

         break;
      case 'w':
         angleX = angleX - 5.0f;

         break;
      case 's':
         angleX = angleX + 5.0f;

         break;
      case 'd':
         angleZ = angleZ + 5.0f;

         break;

      }
   }

}

//for rotatign the display with mouse
void RotateOnEvent_mouseEvent(bool
rotate, int x, int y){
   glRotatef(angleX, 0.0f, 0.0f, 1.0f);
   //glRotatef(angleZ, 0.0f, 1.0f, 0.0f);

   if(variables::isRotateByMouse){

      // glRotatef(angle+y, 0.0f, 0.0f, 1.0f);
      rotateEvent = false;
      if(rotate)
      {
         if(x>mousePosX_old)
         {
            angleX = angleX+x/100;
            mousePosX_old = x;
         }
         if(x<mousePosX_old)
         {
            angleX = angleX-x/100;
            mousePosX_old = x;
         }

         //angleZ = angleZ+y/100;
      }
   }
}
//for zoom in and zoom out
void ZoomOnEvent(){
   glScalef(zoomValue, zoomValue,
zoomValue);
   if(variables::scrollEvent_isScrolled){

      if(variables::scrollEvent_yIntensity >0)
      {
         zoomValue = zoomValue+0.25;
      }
      else
if(variables::scrollEvent_yIntensity <0)
      {
         if(zoomValue>0.5)
```

```
        {
          zoomValue = zoomValue-0.25;
        }

      }
#if bhatti
      cout<<"SCALED TO THE VALUE =
"<<zoomValue<<endl;
#endif
      variables::scrollEvent_isScrolled =
false;
   }
   if(variables::isPlusZoomButtonClicked){
      zoomValue = zoomValue+0.25;
      variables::isPlusZoomButtonClicked =
false;
   }

if(variables::isMinusZoomButtonClicked){
      if(zoomValue>0.25)
      {
         zoomValue = zoomValue-0.25;

variables::isMinusZoomButtonClicked =
false;
      }
   }
}

//for starting the scan
void scan(){
   if(variables::noOfLasers==2){
      cout<<"starting";
      //asdf::camero cap;
      //cap.init(variables::cameraNo);
      if(variables::scanningImageNo==0){

variables::cap.getSnap("garbage.png");
         variables::scanningImageNo++;
      }

if(variables::scanningImageNo<variables::n
oOfSlicesToScan){
         variables::scanningImageNo++;
#if printDebug
         cout<<"scan analyzing
frame::"<<variables::scanningImageNo<<e
ndl;
#endif

#if printDebug
         cout<<"point cloud adding"<<endl;
#endif

serialo::writeToSerial(variables::motorSigle
Step);
         if(variables::colorYesNo==true){

serialo::writeToSerial(variables::lightsOn);

variables::cap.getSnap(variables::rootPath
+"_color_"+to_string(variables::scanningI
mageNo)+variables::extensionImage);

serialo::writeToSerial(variables::lightOff);

            //turning laser 1 on

variables::laserLine1Intensity(variables::las
erIntensity);

variables::cap.getSnap(variables::rootPath
+to_string(variables::scanningImageNo)+"_
laser_1"+variables::extensionImage);
            //turning laser 1 off and 2 on
            variables::laserLine1Intensity(0);

variables::laserLine2Intensity(variables::las
erIntensity);

asdf::addPointCloud(variables::rootPath+t
o_string(variables::scanningImageNo)+"_la
ser_1"+variables::extensionImage,variable
s::theta,variables::rootPath+"color"+to_str
ing(variables::scanningImageNo)+variables
::extensionImage);


variables::cap.getSnap(variables::rootPath
+to_string(variables::scanningImageNo)+"_
laser_2"+variables::extensionImage);
```

```
asdf::addPointCloud(variables::rootPath+to_string(variables::scanningImageNo)+"_laser_2"+variables::extensionImage,variables::theta,variables::rootPath+"color"+to_string(variables::scanningImageNo)+variables::extensionImage);
            //turning laser 2 off
            variables::laserLine2Intensity(0);

        }else{
            //turning laser 1 on

variables::laserLine1Intensity(variables::laserIntensity);

variables::cap.getSnap(variables::rootPath+to_string(variables::scanningImageNo)+"_laser_1"+variables::extensionImage);
            //turning laser 1 off and 2 on
            variables::laserLine1Intensity(0);

variables::laserLine2Intensity(variables::laserIntensity);

asdf::addPointCloud(variables::rootPath+to_string(variables::scanningImageNo)+"_laser_1"+variables::extensionImage,variables::theta);


variables::cap.getSnap(variables::rootPath+to_string(variables::scanningImageNo)+"_laser_2"+variables::extensionImage);

asdf::addPointCloud(variables::rootPath+to_string(variables::scanningImageNo)+"_laser_2"+variables::extensionImage,variables::theta);
            //turning laser 2 off
            variables::laserLine2Intensity(0);
        }

variables::theta+=variables::thetaInc;
        Sleep(variables::stepDelay);
      }else{
        cout<<"scan complete"<<endl;
            variables::scanEnable=false;
        }
    }else if(variables::noOfLasers==1){
        cout<<"starting laser 1"<<endl;
        //asdf::camero cap;
        //cap.init(variables::cameraNo);
        if(variables::scanningImageNo==0){

variables::cap.getSnap("garbage.png");
            variables::scanningImageNo++;
        }


if(variables::scanningImageNo<variables::noOfSlicesToScan){
            variables::scanningImageNo++;
#if printDebug
            cout<<"scan analyzing frame::"<<variables::scanningImageNo<<endl;
#endif

#if printDebug
            cout<<"before motor step"<<endl;
#endif

serialo::writeToSerial(variables::motorSigleStep);
            if(variables::colorYesNo==true){
              //int k=1000;
              cout<<"ligths on"<<endl;

serialo::writeToSerial(variables::lightsOn);
              //Sleep(k);

variables::cap.getSnap(variables::rootPath+to_string(variables::scanningImageNo)+"_color_"+to_string(variables::scanningImageNo)+variables::extensionImage);
              //Sleep(200);

//serialo::writeToSerial(variables::lightOff);
              //cout<<"lights off"<<endl;
              //turning laser 1 on

//serialo::writeToSerial(variables::laserLine2Intensity(variables::laserIntensity));
```

```
            //Sleep(k);

//variables::cap.getSnap(variables::rootPat
h+to_string(variables::scanningImageNo)+
"_laser_1"+variables::extensionImage);
            //Sleep(300);
            //turning laser 1 off

//serialo::writeToSerial(variables::laserLine
2Intensity(0));
            cout<<"point cloud
adding"<<endl;

//asdf::addPointCloud(variables::rootPath
+to_string(variables::scanningImageNo)+"_
laser_1"+variables::extensionImage,variabl
es::theta,variables::rootPath+to_string(var
iables::scanningImageNo)+"_color_"+to_st
ring(variables::scanningImageNo)+variable
s::extensionImage);

         }else{
            //turning laser 1 on
            cout<<"laser 1 on"<<endl;

serialo::writeToSerial(variables::laserLine2I
ntensity(variables::laserIntensity));

variables::cap.getSnap(variables::rootPath
+to_string(variables::scanningImageNo)+"_
laser_1"+variables::extensionImage);
            //turning laser 1 off and 2 on
            //cout<<"laser 1 off"<<endl;

//serialo::writeToSerial(variables::laserLine
1Intensity(0));
            cout<<"addpoint cloud"<<endl;

//asdf::addPointCloud(variables::rootPath
+to_string(variables::scanningImageNo)+"_
laser_1"+variables::extensionImage,variabl
es::theta);
            cout<<"point cloud
added"<<endl;
         }

variables::theta+=variables::thetaInc;
cout<<"theta::"<<variables::theta<<endl;
            Sleep(variables::stepDelay);
         }else{
            cout<<"scan complete"<<endl;
            variables::scanEnable=false;
         }
      }
      cout<<"loop ended"<<endl;

}
```

```
            #pragma once
#ifndef MYDIALOG_H
#define MYDIALOG_H

#include <QDialog>
#include<string>
using namespace std;


namespace Ui {
```

```
class MyDialog;
}

class MyDialog : public QDialog
{
   Q_OBJECT
public:
//    static ;
   string forDialogBox;
   string toSave;
   explicit MyDialog(QWidget *parent = 0);
   ~MyDialog();
private slots:
   void on_pushButton_clicked();
private:
   Ui::MyDialog *ui;
};
//string MyDialog::forDialogBox="";
#endif // MYDIALOG_H
```

Otherfunctions.h

```
#ifndef OTHERFUNCTIONS
#define OTHERFUNCTIONS


#include<SFML/OpenGL.hpp>
#include<SFML/Window.hpp>
#include<SFML/Graphics.hpp>
#include<SFML/System.hpp>

#include<QSerialPort>
#include<iostream>
#include<QDebug>
#include<QThread>
using namespace std;
QT_USE_NAMESPACE


namespace serialo{
//creatign serial port object for
commnunicatoin
QSerialPort serial;

//for setting serial port
bool setSerialPort(/*QSerialPort
&serial,*/string name="COM7"){
   serial.setPortName(name.c_str());

serial.setBaudRate(QSerialPort::Baud9600)
;
   serial.setDataBits(QSerialPort::Data8);
   serial.setParity(QSerialPort::NoParity);
   serial.setStopBits(QSerialPort::OneStop);

serial.setFlowControl(QSerialPort::NoFlow
Control);
if(serial.open(QIODevice::ReadWrite)){
      cout<<"device opened"<<endl;
      return true;
   }else{
      cout<<"failed to open device"<<endl;
      return false;
   }
}
//for writting to serial port
void writeToSerial(/*QSerialPort &serial,
*/string toWrite){
   if (serial.isOpen() && serial.isWritable()){
      cout<<"is open and writeable"<<endl;
      QByteArray ba(toWrite.c_str());
      serial.write(ba);
      serial.flush();
      cout<<"data has been
send::"<<toWrite<<endl;
   }
}
}
#endif // OTHERFUNCTIONS
```

Qsmflcanvas.h

```
#pragma once
#include <QWidget>
#include <QTimer>
#include<SFML/OpenGL.hpp>
#include<SFML/Window.hpp>
#include<SFML/Graphics.hpp>
#include<SFML/System.hpp>
#ifdef Q_WS_X11
#include <Qt/qx11info_x11.h>
#include <X11/Xlib.h>
#endif
#include <iostream>
#include"otherfunctions.h"

///
/// \brief The QSFMLCanvas class
///the super class for crating SFML widget
which is used for displaying the 3D point
cloud
class QSFMLCanvas : public QWidget,
public sf::RenderWindow
{
   //Q_OBJECT
public:
   sf::RenderWindow window;
   explicit QSFMLCanvas(QWidget *parent,
const QPoint& Position, const QSize& Size,
unsigned int FrameTime = 0);
   virtual void showEvent(QShowEvent*);
   virtual QPaintEngine* paintEngine()
const;
   virtual void paintEvent(QPaintEvent*);
   virtual ~QSFMLCanvas();
   virtual void OnInit();
   virtual void OnUpdate();
private:
   QTimer myTimer;
   bool myInitialized;
};
```

Variables.h

```
#pragma once
#ifndef VARIABLES
#define VARIABLES
#include<string>
#include<iostream>
#include<vector>


using namespace std;
namespace variables {
//class for
class camero {
public:
   CvCapture* capture;
   camero() {

   }
   void init(int camNo = 1) {
      cout<<"init called::"<<camNo<<endl;
      capture =
cvCaptureFromCAM(camNo);
      cvSetCaptureProperty(capture,
CV_CAP_PROP_FRAME_HEIGHT, 720);
      cvSetCaptureProperty(capture,
CV_CAP_PROP_FRAME_WIDTH, 1280);
//       cout<<"width set"<<endl;

//       cout<<"height set"<<endl;
   }
   void getSnap(string str = "") {
      //cout<<"getSnap called"<<endl;
      IplImage* frame =
cvQueryFrame(capture); //Create image
frames from capture
      //cout<<"frame taken"<<endl;
      //cvShowImage("Camera_Output",
frame);   //Show image frames on created
window
      if (str != "") {
         //cout<<"saving images"<<endl;
         cout<<"path is::"<<str<<endl;
         cvSaveImage(str.c_str(), frame);
         cout<<"image saved"<<endl;
      }
   }
   Mat getSnap2(string str = ""){
      //cout<<"getSnap2 called"<<endl;
      IplImage* frame =
cvQueryFrame(capture); //Create image
frames from capture
      //cout<<"frame taken"<<endl;
      //cvShowImage("Camera_Output",
frame);   //Show image frames on created
window
      if (str != "") {
         //cout<<"saving images"<<endl;
         //cout<<"path is::"<<str<<endl;
         cvSaveImage(str.c_str(), frame);
         //cout<<"image saved"<<endl;
      }
      return Mat(frame,true);
   }
};

#define bhatti 0
   // for cummunications with arduino
   //diagnostics
   bool lightStatus;
   bool laserStatus;
   int laserIntensity=5;
#define clockwise_ 1
#define anticlockwise_ 0
   bool stepperDirection;
   // steps are 0.0536, 0.45 (0.5143)  , 1.8
#define high_ 0
#define medium_ 1
#define low_ 2
   int stepperMode;
#define custom 0
#define half 1
#define full 2
   int stepperRotation;
   int noOfSteps;
#define start_ 1
#define stop_ 0
   int diganosticsStatus=1;

   bool motorStatus;
#define colorYes 1
#define colorNo 0
   bool colorStatus;
```

```
   //for GUI
   bool isRead = false;
   std::string toReadCustomFileName;
   std::string toWriteFileName;
   std::string customExtension=".txt";
   std::string plyExtension=".ply";
   std::string extensionImage=".png";
   std::string rootPath="C:\\Users\\Fahad
Mubeen\\Desktop\\a\\";


   //for color writing
   bool colorYesNo=false;
   int noOfSlicesToWrite=0;
   int noOfPointsPerSliceToWrite=0;


   //new variables
   string workingDirectory="";
   bool
SerialCommunicationStatus=true;//flag
   string serialPortName;
   int cameraNo=1;
   int scanningImageNo=0;
#define printDebug 1
   int stepDelay=70;
   //for serialCommunication
   string motorSigleStep="g";
   const string smallStep="e";
   const string mediumStep="f";
   const string largeStep="g";

   const string lightsOn="d";
   const string lightOff="h";
   //for angle tracing
   long double theta=0;
   long double thetaInc=1.8;
   const long double
thetaSmallInc=0.12857;
   const long double
thetaMediumInc=0.5143;
   const long double thetaLargeInc=1.8;
   bool scanEnable=false;
   const string serialMotorFull="l";
   const string serialMotorHalf="k";
   const string serialMotorCustom="g";
   int motorSteps=0;
   int noOfLasers=1;
   int webcamViewOn=false;
   //for camera
   camero cap;
   //for laser lines functions
   string laserLine1Intensity(int intensity){
      cout<<"internsity
recieved::"<<intensity<<endl;
      switch(intensity){
      case 0:
         return "2";
      case 1:
         return "3";
      case 2:
         return "4";
      case 3:
         return "5";
      case 4:
         return "6";
      case 5:
         return "7";
      default:
         cout<<"incorrect value of
intensity"<<endl;
         return "garbage";
      }
   }
   string laserLine2Intensity(int intensity){
      cout<<"internsity
recieved::"<<intensity<<endl;
      switch(intensity){
      case 0:
         return "8";
      case 1:
         return "9";
      case 2:
         return "a";
      case 3:
         return "b";
      case 4:
         return "i";
      case 5:
         return "j";
      default:
```

```
        cout<<"incorrect value of
intensity"<<endl;
        return "garbage";
      }
   }



   //for scanning
#define highScan 1
#define mediumScan 1
#define lowScan 1
   int noOfSlicesToScan=202;
   int noOfPointsPerScan=720;
   int scanAngle=0;


   //forOpenCV

      vector<vector<vector<long double>>>
point3dxyz;
      vector<vector<vector<uchar>>>
colour;



   //new variables
      const int distance=36;

   //11beerbhatti
   //for event handling
   bool keyboardEvent_isKeyPressed;
   char keyboardEvent_key;
   bool scrollEvent_isScrolled;
   int scrollEvent_yIntensity;
   bool rotate=false;
   bool isRotateByMouse = false;
   bool isMinusZoomButtonClicked = false;
   bool isPlusZoomButtonClicked = false;

   bool test=false;
   int testi=1;
}
#endif // VARIABLES
```

APPENDIX C: EMBEDDED CODE

```
int lightPin=2;
int motorPin=3;
int laserPin=10;
int laserPin2=11;
int motorDirection=4;

void motorStep(){
 digitalWrite(motorPin,HIGH);
 delayMicroseconds(500);
 digitalWrite(motorPin,LOW);
 delay(2);
}
void motorSmallStep(){
 motorStep();
}
void motorMediumStep(){
 for(int i=0;i<4;i++)
  motorStep();
}
void motorLargeStep(){
 for(int i=0;i<14;i++)
  motorStep();
}
void motorFull(){
 for(int i=0;i<2800;i++)
  motorStep();
}
void motorHalf(){
 for(int i=0;i<1400;i++)
  motorStep();
}
void motorCustom(int steps){
 for(int i=0;i<steps;i++)
  motorStep();
}
void laserBrigtness(int intensity){
 analogWrite(laserPin,
(intensity)*255.0/5.0);
}
void laser2Brigtness(int intensity){

analogWrite(laserPin2,(intensity)*255.0/5.
0);
}
void setup() {
 Serial.begin(9600);
 pinMode(lightPin,OUTPUT);
 pinMode(motorPin,OUTPUT);
 pinMode(laserPin,OUTPUT);
 pinMode(laserPin2,OUTPUT);
 pinMode(motorDirection,OUTPUT);
 digitalWrite(motorDirection,HIGH);
}
int i;
void loop() {
 char g;
 int customNo=0;
 int first=0,second=0;

 if(Serial.available()>0){
  char c=Serial.read();
  Serial.print("char recieved::");
  Serial.println(c);
  switch(c){
   case '0'://clockwise
   break;
   case '1'://anticlockwise
   break;
   case 'l'://its l
   motorFull();
   break;
   case 'k':
   motorHalf();
   break;
   case 'c':
   while(customNo<2){
     if(Serial.available()>0){
      g=Serial.read();
      if(customNo==0){
       first=(g-48)*10;
      }else if(customNo==1){
       second=(g-48);
      }
      customNo++;
     }
    }
    motorCustom(first+second);
   break;
   case '2':
   laserBrigtness(0);
```

```
break;
case '3':
laserBrigtness(1);
break;
case '4':
laserBrigtness(2);
break;
case '5':
laserBrigtness(3);
break;
case '6':
laserBrigtness(4);
break;
case '7':
laserBrigtness(5);
break;
case '8':
laser2Brigtness(0);
break;
case '9':
laser2Brigtness(1);
break;
case 'a':
laser2Brigtness(2);
break;
case 'b':
laser2Brigtness(3);
break;
case 'i':
laser2Brigtness(4);
break;
case 'j':
laser2Brigtness(5);
break;
case 'd':
digitalWrite(lightPin,HIGH);
break;
case 'h':
digitalWrite(lightPin,LOW);
break;
case 'e':
motorSmallStep();
break;
case 'f':
motorMediumStep();
break;
case 'g':
    motorLargeStep();
    break;
  }
 }
}
```