PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

Mohamed El Bachir El Ibrahimi University, Bordj Bou Arreridj

Faculty of Science and Technology

Department of Electronics

Master's Thesis

-MCIL5-

Field: Electronics

Specialization: Industrial Electronics

By:

***Mr. BOUCHERK** Oussama*

Titled

Implementation of 3D Scanner for Small Businesses

Evaluated on: June 29th, 2025

Evaluation committee composed of:

Name and Surname	***Rank***	***Role***	***Institution***
Messaoudene Idris	MCA	President	University of BBA
Boussahoul Abd El Karim	MAA	Examiner	University of BBA
Abed Tarek	MAA	Supervisor	University of BBA

Acknowledgment

First and foremost, I would like to express my deepest gratitude to my family, whose unwavering support, love, and encouragement have been the foundation of all my achievements. I am especially thankful to **my parents**, whose sacrifices and guidance shaped my path, and to **my sister**, whose presence and support have always been a source of motivation and strength throughout this journey.

I extend my sincere appreciation to my supervisor, Mr. Abed Tarek, for his guidance, valuable feedback, and patience during the development of this thesis. His expertise and encouragement were essential at every stage of the work.

My heartfelt thanks go to my colleagues and friends, especially **Abd El Hamid**, for their continuous support, collaboration, and constructive discussions. I am also grateful to the members of our scientific club **ElectroMinds Club**, who inspired me and helped me grow both technically and personally.

I would also like to express my sincere appreciation to **Professor Amar Noui**, Dean of the Faculty, for his continued support and encouragement throughout my academic journey. His commitment and availability have been a source of motivation both during this project and throughout my years of study.

I extend my deep gratitude to **Mohamed El Bachir El Ibrahimi University** of Bordj Bou Arreridj for being more than just an institution. It has been a space of growth, learning, and discovery. I thank all its faculty members, staff, and departments who contributed—directly or indirectly—to shaping my academic and personal development over the years.

A special appreciation goes to all the amazing teachers I had the privilege to learn from during my university journey. Each of them has left an impact on my way of thinking, my understanding, and my passion for engineering. I sincerely hope to continue learning, remain in contact with them, and—one day—have the honor to teach alongside them.

To everyone who played a role in this journey—thank you.

Abstract

This thesis presents the design and development of a low-cost high-performance 3D scanner based on ultrasonic Time-of-Flight (ToF) technology. The system is composed of a microcontroller-controlled platform equipped with stepper motors and a VL53L0X distance sensor. The scanner captures layer-by-layer measurements of an object by rotating it and lifting the sensor along the vertical axis. The collected data is stored as a point cloud on an SD card and processed using a Python-based software pipeline. The reconstruction phase uses Poisson Surface Reconstruction (PSR) via the Open3D library, with alternatives such as the Ball Pivoting Algorithm (BPA). The complete system integrates mechanical, electronic, and software elements. Tests were carried out on real objects, and the results demonstrated the scanner's ability to produce usable 3D models for prototyping and educational purposes. Future improvements are proposed to enhance precision, automation, and reconstruction quality.

Résumé

Ce mémoire présente la conception et le développement d'un scanner 3D à faible coût et haute performance basé sur la technologie ultrasonore Time-of-Flight (ToF). Le système se compose d'une plateforme contrôlée par un microcontrôleur, équipée de moteurs pas-à-pas et d'un capteur de distance VL53L0X. Le scanner effectue des mesures couche par couche en faisant tourner l'objet et en déplaçant le capteur verticalement. Les données collectées sont stockées sous forme de nuage de points sur une carte SD et traitées à l'aide d'un pipeline logiciel développé en Python. La phase de reconstruction utilise l'algorithme Poisson Surface Reconstruction (PSR) via la bibliothèque Open3D, avec des alternatives comme l'algorithme Ball Pivoting (BPA) proposées. Le système combine des aspects mécaniques, électroniques et logiciels. Des tests ont été réalisés sur des objets réels, montrant la capacité du scanner à produire des modèles 3D exploitables pour le prototypage ou l'usage pédagogique. Des perspectives d'amélioration sont proposées pour accroître la précision, l'automatisation et la qualité des reconstructions.

ملخص

يعرض هذا المشروع تصميم وتطوير ماسح ثلاثي الأبعاد منخفض التكلفة وعالي الأداء يعتمد على تقنية وقت الطيران (ToF) باستخدام الموجات فوق الصوتية. يتكون النظام من منصة ميكانيكية يتحكم فيها متحكم دقيق، مزودة بمحركات خطوة بخطوة ومستشعر مسافة من نوع VL53L0X. يقوم الماسح بجمع بيانات ثلاثية الأبعاد عن طريق تدوير الجسم ورفع المستشعر عموديًا طبقةً تلو الأخرى. تُخزَّن البيانات كنقطة سحابية على بطاقة ذاكرة ويتم معالجتها باستخدام برنامج مكتوب بلغة Python. تتم إعادة بناء النموذج الثلاثي الأبعاد باستخدام خوارزمية Poisson Surface Reconstruction (PSR) كبديل. يجمع النظام بين الجوانب الميكانيكية Ball Pivoting ، مع إمكانية استخدام خوارزميةOpen3D من خلال مكتبة والإلكترونية والبرمجية. أُجريت اختبارات على أجسام حقيقية وأظهرت النتائج قدرة الماسح على إنتاج نماذج ثلاثية الأبعاد قابلة للاستخدام في النمذجة الأولية والتعليم. كما تقترح الدراسة تحسينات مستقبلية لتعزيز الدقة والأتمتة وجودة النماذج.

Table of contents

Table of Abbreviation

3D: Three Dimensions

GUI: Graphic User Interface

PSR: Poisson surface reconstruction

uC : microcontroller

IC : Integrated circuit

TOF: time of flight

LIDAR: Light Detection and Ranging

PC : Point Cloud

BP: Ball Pivoting

Table of Figures

General Introduction

Humans naturally perceive and interact with the world in three dimensions, while early computers were originally limited to processing and displaying two-dimensional data, primarily through digital numbers and basic graphics. It wasn't until around 1980 that IBM introduced the first implementations of 3D graphics on personal computers, marking the beginning of a new era in digital visualization.

As the demand for custom 3D models grew, especially in fields like manufacturing and industrial design, the need for fast and accurate model acquisition became essential. That’s where 3D scanning entered the scene. Although experimental 3D scanners existed as early as the 1980s, modern 3D scanning technologies have matured significantly and are now widely used across diverse domains such as reverse engineering, 3D printing, game development, animation, and even cultural preservation.

Today, 3D scanners are applied across a wide range of fields due to their ability to capture precise geometric data. In industrial inspection and quality control, they are used to detect dimensional deviations in manufactured parts. In medical imaging, 3D scanning supports the design of prosthetics, orthotics, and aids in surgical planning. The fields of archaeology and cultural heritage benefit greatly from non-contact digitization of historical artifacts, enabling preservation and restoration without physical interference. In architecture and civil engineering, scanners are used to create accurate as-built models and conduct structural assessments. The technology is also widely integrated into virtual and augmented reality, where detailed 3D models contribute to immersive and interactive environments. In education and research, 3D scanners facilitate physical modeling, simulation, and prototyping. Additionally, in forensics and law enforcement, they are employed to reconstruct crime scenes and analyze evidence in three dimensions, improving investigation accuracy.

These diverse applications continue to grow as the technology becomes more accessible, reliable, and integrated into digital workflows.

The early 3D graphics relied on techniques like rendering and ray tracing, but their performance was limited by the available hardware. The real shift happened around 1995 with the emergence

of dedicated graphics cards, which significantly accelerated 3D rendering capabilities. From that point on, the world of 3D computing began to evolve rapidly.

Parallel to the software advancements, hardware innovation was also taking shape. The concept of 3D wasn't limited to digital screens—3D printing technologies were simultaneously emerging. In 2005, the open-source RepRap project gave rise to a new wave of innovation, allowing individuals to transform digital 3D models into tangible objects. This revolutionized rapid prototyping and personal manufacturing, making 3D design more crucial than ever.

This thesis presents the design and development of a cost-effective 3D scanner, combining both hardware and software elements to achieve reliable 3D digitization. The system is tailored for educational and prototyping use, focusing on accessibility, simplicity, and practical application.

In this manuscript, Chapter One introduces the fundamentals of the 3D concept, explaining how the three-dimensional world is mathematically represented and processed by computer graphics systems, particularly through GPU-based calculations. Chapter Two explores various 3D scanning techniques and focuses on mesh construction methods used to convert point cloud data into 3D models. Chapter Three details the complete design of the 3D scanner, from hardware architecture and electronic integration to the software pipeline responsible for mesh generation. Finally, Chapter Four presents the developed prototype, accompanied by experimental tests, analysis, and the resulting 3D reconstructions.

Chapter 01: 3D Geometry and 3D Scanning

1.1 Introduction

This chapter lays the foundational knowledge necessary for understanding 3D scanning systems. It begins with the mathematical and computational principles of 3D graphics, emphasizing how digital devices represent and render 3D shapes. The historical evolution of 3D scanning technologies is also traced, highlighting key milestones. Subsequently, it presents an overview of the major 3D scanning techniques including photogrammetry, LiDAR, and Time-of-Flight (ToF), which is the core technique used in this project. Finally, it introduces the concept of surface reconstruction, with a focus on normal vector estimation and key algorithms used to convert raw scan data into complete 3D models.

1.2 Principle of 3D graphics in computers

Three-dimensional (3D) computer graphics represent one of the most impactful advancements in digital technology, allowing complex objects and environments to be visualized, manipulated, and interacted with through a two-dimensional (2D) screen. At its core, 3D modeling relies on mathematical abstractions to define objects in a virtual space. Each model is composed of a set of vertices, each positioned along the X, Y, and Z axes, representing width, height, and depth respectively. These vertices are connected by edges to form polygons—typically triangles or quadrilaterals—which collectively define the faces of a 3D mesh. The mesh is essentially a wireframe that outlines the geometry of a digital object.

To be visualized on a screen, 3D data undergoes a transformation pipeline involving several stages of mathematical operations. These include model transformation (to place the object in world space), view transformation (to position and orient the virtual camera), projection transformation (to map 3D coordinates to 2D coordinates), and viewport transformation (to convert the result into pixel locations on the display). These operations are carried out using matrix algebra, primarily through the use of 4x4 matrices, enabling translation, rotation, scaling, and projection transformations in a consistent and computationally efficient manner. As an example, we take:

- Rotation around X-axe:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \tag{1.1}$$

- Projection formula:

$$x_{screen} = \frac{x_{world} \cdot f}{z_{world}} \tag{1.2}$$

Rendering is the next step, where the mesh is converted into a visual representation with lighting, shading, and color. This is accomplished using either software rendering on the central processing unit (CPU) or hardware-accelerated rendering via the graphics processing unit (GPU). The rasterization process converts the transformed 3D geometry into a 2D image by computing which pixels correspond to which parts of the model. Modern rendering pipelines may also include advanced techniques such as z-buffering for depth handling, lighting models like Phong or Blinn-Phong shading, and texture mapping to add surface details. More complex methods like ray tracing and path tracing provide photorealistic results but are computationally intensive.

The underlying mathematics of 3D graphics primarily involves linear algebra and trigonometry. Vectors are used to define positions, directions, and normals, while matrix operations facilitate transformations. The dot product is used in calculating the angle between vectors, critical for determining light intensity on surfaces. The cross product helps derive surface normals, essential for accurate lighting calculations. Additionally, trigonometric functions such as sine, cosine, and tangent are widely used in projecting 3D coordinates onto 2D planes and in simulating rotations.

These concepts are not merely theoretical; they have become foundational in various industries. In video game development, 3D engines render immersive virtual worlds in real time. In the film and animation industry, 3D models and simulations are used to create detailed visual effects and animated characters. Engineering and architecture rely on 3D CAD software to design and test prototypes. In the medical field, 3D imaging technologies like CT and MRI scans help visualize anatomical structures with precision. Moreover, 3D data forms the backbone of virtual reality, augmented reality, robotics, and scientific simulations.

1.3 History of 3D Scanning

The origins of 3D scanning can be traced back to the mid-20th century, with early developments grounded in the principles of photogrammetry. This technique, which dates back even earlier in analog form, involves capturing multiple photographs of an object from different angles and using triangulation methods to calculate depth and spatial relationships. Photogrammetry laid

the conceptual foundation for digital 3D scanning, offering a method to reconstruct physical objects into digital form without physical contact. [1]

By the 1980s, advancements in computing and imaging technologies allowed researchers and companies to develop early automated 3D scanning systems. Structured light scanning emerged as one of the first practical methods. This approach involves projecting known patterns of light—such as stripes or grids—onto an object and observing how the patterns deform across its surface. The deformation is analyzed to calculate the depth of each point, enabling the construction of a 3D point cloud. One of the notable pioneers in this domain was IBM, which collaborated with academic institutions to explore the potential of structured light and laser triangulation for industrial and scientific applications.

As the demand for digitization increased, especially in fields like quality control, reverse engineering, and computer graphics, commercial interest grew significantly during the 1990s. Companies such as *Minolta*, *Steinbichler*, and *Cyberware* began manufacturing 3D scanners that relied on various technologies, including laser scanning, structured light, and contact-based digitizers. These systems became increasingly accurate and portable, making them viable for use not only in laboratories but also in manufacturing and fieldwork.

Another key advancement during this period was the adoption of time-of-flight (ToF) scanning technology. ToF systems emit a signal—usually infrared light—and measure the time it takes for the signal to reflect off the surface and return to the sensor. This method allows for non-contact, high-speed distance measurement and is particularly well-suited for scanning large surfaces and objects. Time-of-flight remains a central principle in many modern 3D sensors, including those integrated into mobile devices.

With the advent of more powerful microprocessors, high-resolution cameras, and affordable sensors in the early 2000s, 3D scanning began to shift from an industrial tool to a consumer technology. Devices like the Microsoft Kinect brought depth sensing into homes, primarily for gaming but also sparking interest in 3D scanning and motion capture. In parallel, open-source tools such as *Meshroom*, *COLMAP*, and *Open3D* enabled developers and hobbyists to experiment with photogrammetry and mesh processing, democratizing access to 3D reconstruction techniques. [2]

The 2010s witnessed the miniaturization and integration of LiDAR and *ToF* sensors into mobile platforms, leading to real-time scanning capabilities in smartphones and tablets. These compact sensors, often supported by advanced computer vision algorithms, made it possible to scan environments and objects with remarkable accuracy on-the-go. Fields such as heritage preservation, forensic analysis, and digital twin creation began to incorporate these tools for documentation and simulation.

Today, 3D scanning technologies are categorized by the underlying principles they rely on. Photogrammetry remains popular for its low cost and flexibility, especially in outdoor or large-scale scenarios. Structured light scanning offers high precision and is widely used in dental, industrial, and cultural heritage applications. Laser triangulation provides excellent accuracy for smaller objects and fine details, while time-of-flight scanning offers good performance across a wide range of distances and lighting conditions. Each method has distinct strengths and limitations, and hybrid systems often combine several approaches to achieve better performance and coverage.

In the context of this thesis, the 3D scanning method employed is based on time-of-flight principles using an infrared distance sensor. The system is designed to scan objects in a controlled manner using mechanical movements via stepper motors to generate a comprehensive point cloud. The acquired data will later be used to reconstruct the object's surface and generate a mesh representation, enabling further processing and visualization.

1.4 Techniques for 3D Scanning

The process of 3D scanning revolves around measuring spatial information from real-world objects and converting it into a digital point cloud representation. This point cloud is later used to reconstruct a 3D mesh of the scanned objects. There are several techniques for acquiring this data, each with its own methodology, precision level, and area of application. The three most prominent approaches are Photogrammetry, LIDAR, and Time-of-Flight.

1.4.1 Photogrammetry

Photogrammetry is a vision-based 3D scanning technique that reconstructs objects by analyzing multiple images taken from various angles. Through identifying key features across photographs, it triangulates their positions in 3D space, forming a dense point cloud. This method relies on

high-resolution images and advanced feature-matching algorithms, often aided by software like *Agisoft Metashape* or *Meshroom*. [1] [2]

Photogrammetry is widely used due to its low hardware requirements—only a camera is needed—and its ability to generate detailed models. However, the quality depends heavily on lighting conditions, image overlap, and surface texture. It performs best on textured objects but struggles with reflective or uniform surfaces

1.4.2 LIDAR

LIDAR (Light Detection and Ranging) is a high-precision technique that emits laser pulses to measure distances to objects. By capturing the time, it takes for the reflected light to return, LIDAR systems build an accurate 3D map of the environment. Often used in autonomous vehicles, topographic mapping, and archeology, LIDAR offers centimeter-level precision and works well over large areas.

LIDAR typically uses pulsed lasers and rotates in multiple axes to capture 3D data in all directions. It is powerful but expensive and often overkill for small-scale object scanning. LIDAR also requires calibration and can be affected by environmental factors such as dust, fog, or sunlight.

1.4.3 Time-of-Flight (ToF)

Time-of-Flight is a depth-sensing technique that measures the time taken by a signal—either light or sound—to travel from an emitter to the object and back to a receiver. This round-trip time is then converted into a distance using the following formula:

$$d = \frac{v \,.\, t}{2} \tag{2.1}$$

Where:

- d is the measured distance,
- t is the time taken for the round trip,
- v is the speed of the signal in the given medium

Two implementations of ToF exist:

- Optical ToF, using lasers or infrared light,

- Ultrasonic ToF, using sound waves.

Ultrasonic ToF is advantageous due to its affordability, low power consumption, and resistance to surface reflectivity or lighting conditions. However, it offers lower resolution compared to optical ToF due to the slower propagation speed of sound and larger beam width, which limits fine surface detail. Nevertheless, it is a practical and educational solution for low-cost 3D scanning, making it the ideal choice for our application.

1.5 Normal vectors

Normal vectors, also called normal in computer graphics, are one of the most critical parameters in the mesh reconstruction process (described in the next section). They define the orientation of the surface at each point in the point cloud and play a fundamental role in rendering the 3D object accurately. Without correct normal estimation, the reconstructed mesh may appear broken, uneven, or visually incorrect, especially when lighting and shading are applied.

In the context of 3D scanning and surface reconstruction, normals serve as vectors perpendicular to the surface at each point, providing the necessary directional information for constructing faces and determining which direction is considered “outside” of the mesh.

1.5.1 Estimation Methods

Since most low-cost 3D scanning setups—including this project—do not capture normal vectors directly, they must be estimated algorithmically after acquiring the raw point cloud. There are multiple ways to estimate normals:

- **Neighbor-based estimation**: The most common method involves selecting a group of neighboring points (usually using k-nearest neighbors or radius search), and then computing the surface normal that best fits these neighbors using Principal Component Analysis (PCA) or least squares fitting. The direction of the normal is then set perpendicular to the local surface plane.
- **Sensor or camera-based estimation**: In more advanced systems, the normal can be inferred directly from the direction of the sensor or camera that captured the point. This is often used in systems based on photogrammetry or stereo vision, where the camera view direction provides normal orientation.

- **Orientation propagation**: Once raw normals are estimated, the orientations might not all point in the correct direction. A normal orientation step is required, which uses the connectivity of the point cloud to propagate correct outward-facing directions and maintain a consistent mesh.

1.5.2 Importance in Surface Reconstruction

For surface reconstruction algorithms like Poisson Surface Reconstruction (PSR), having accurate and consistently oriented normals is mandatory. The PSR algorithm interprets normals as directions of gradient fields, which it integrates to recover the surface shape. Poor normal estimation leads to surface holes, artifacts, or incorrect geometry in the final mesh.

Open3D library is one of the best libraries that allows to generate mesh based on the three mesh generation algorithms, using a simple function after loading the point cloud the mesh gets generated according to the algorithm used and the chosen parameters

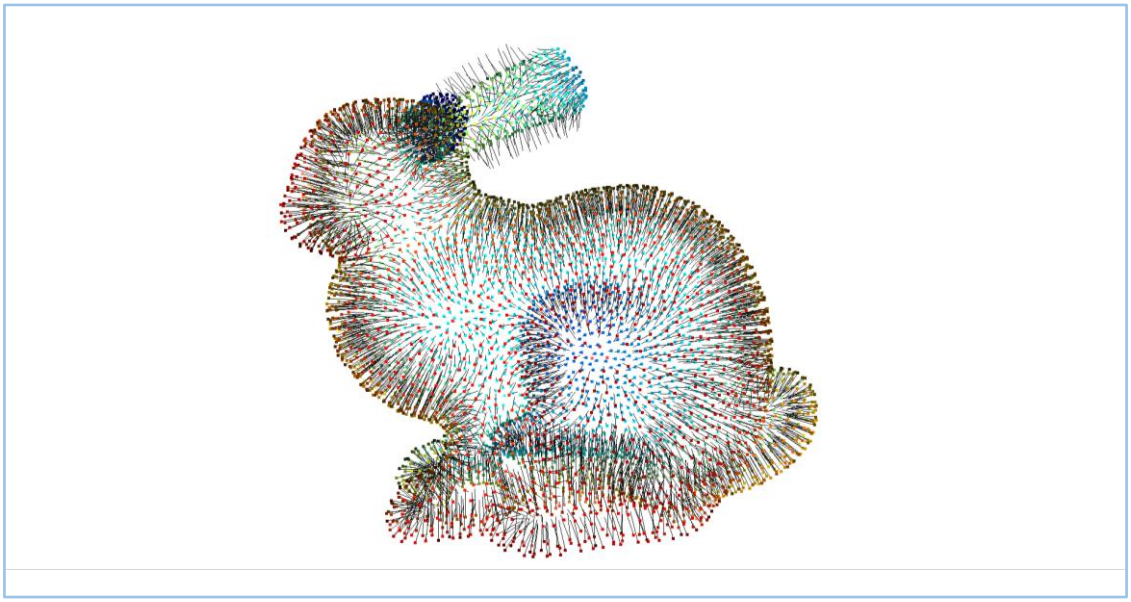


Figure 2. 1: Illustration of normal vectors orientation [4]

1.6 Mesh Reconstruction Algorithms

After collecting a point cloud from the 3D scanning process, the next essential step is mesh reconstruction**.** This involves converting the unstructured set of 3D points into a continuous, watertight surface that represents the object's shape. Multiple reconstruction algorithms exist, each with unique approaches, requirements, and trade-offs. [4]

1.6.1 Alpha Shapes

Alpha Shapes is a generalization of the convex hull algorithm used for surface reconstruction. It builds a mesh by connecting nearby points based on a given alpha parameter, which determines the level of detail. Smaller alpha values preserve finer features but risk overfitting, while larger values produce smoother, simpler shapes.

This method is well-suited for relatively dense and uniform point clouds. Alpha Shapes can preserve sharp edges and voids in the data but may struggle with noise and inconsistent point density. The algorithm is often used for object boundary detection and solid modeling but is not ideal for noisy datasets with missing normals.

The implementation is based on the convex hull of the point cloud. If we want to compute multiple alpha shapes from a given point cloud, then we can save some computation by only computing the convex hull once and pass it to "*create_from_point_cloud_alpha_shape*" function using Open3D library. [4]

Figures 2.2 and 2.3 show an example of the point cloud of an object and its generated mesh. [4]

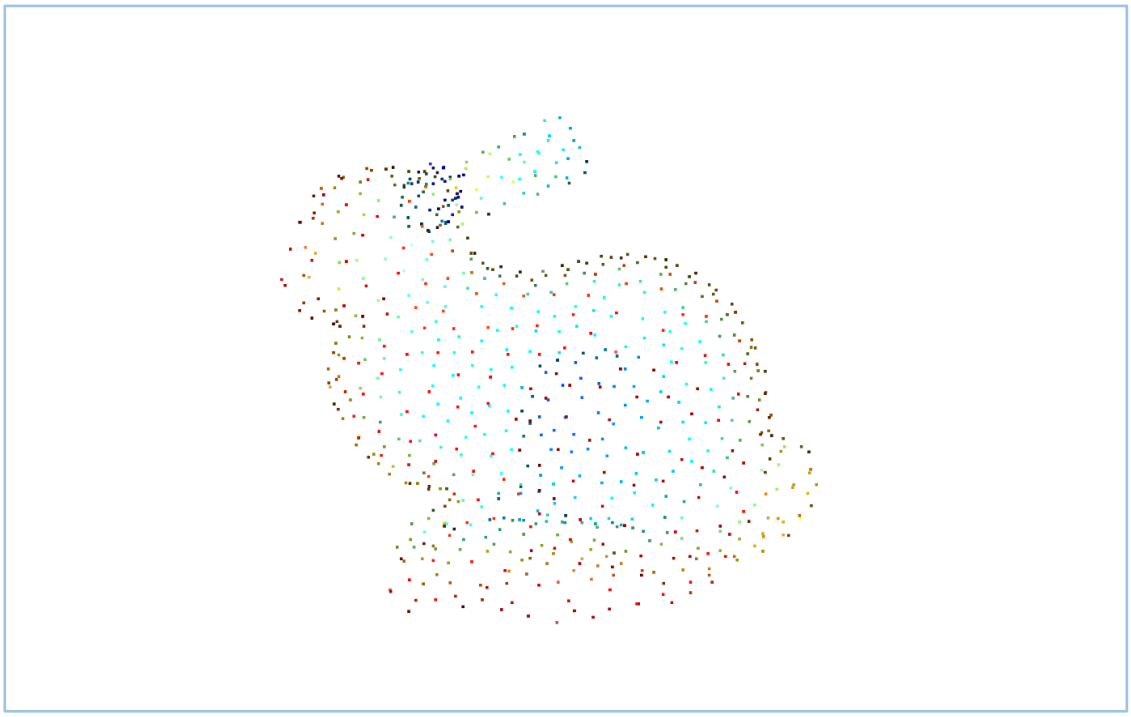


Figure 2. 2: Rabbit Point Cloud

Figure 2. 3: Generated Rabbit Model from Point Cloud

The quality of the mesh generated using the Alpha Shapes algorithm is highly dependent on the chosen alpha parameter. A low alpha value may lead to an *underfitted mesh*, where surface details are lost, and the model appears overly simplified—particularly in geometrically complex regions. On the other hand, using an alpha value that is too high can result in *overfitting*, where the algorithm begins to connect points incorrectly, producing distorted or non-manifold surfaces. This behavior is illustrated in the figure 2.4 [4], where excessive alpha values introduce artifacts and irregularities into the mesh structure.

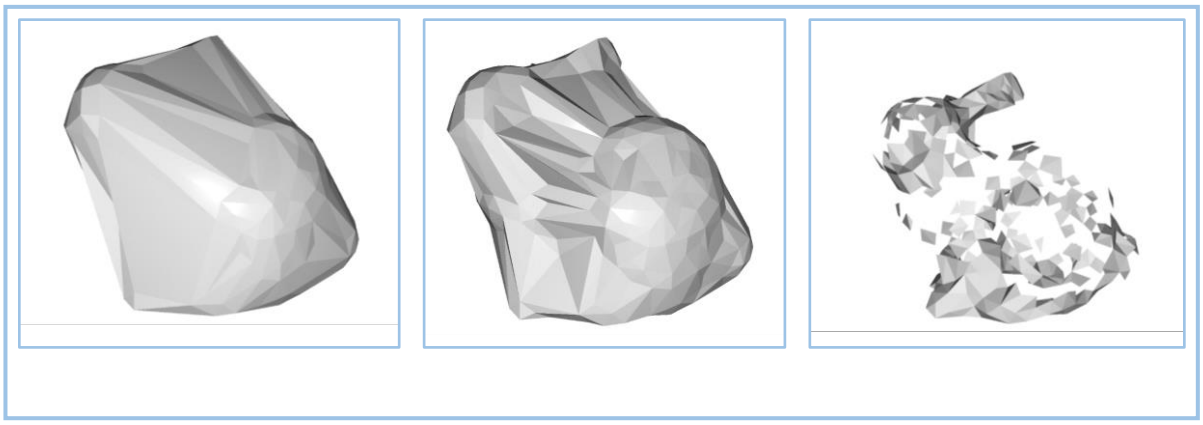


Figure 2. 4: Comparison of 3 Generated Meshes (a. alpha=0.136, b. alpha=0.037, c. alpha=0.01)

1.6.2 Ball Pivoting Algorithm (BPA)

Ball Pivoting is a geometric reconstruction method that simulates a ball of fixed radius rolling over the point cloud. Whenever the ball touches three points simultaneously, a triangle is formed and added to the mesh. This method mimics how physical scanning probes might move around an object.

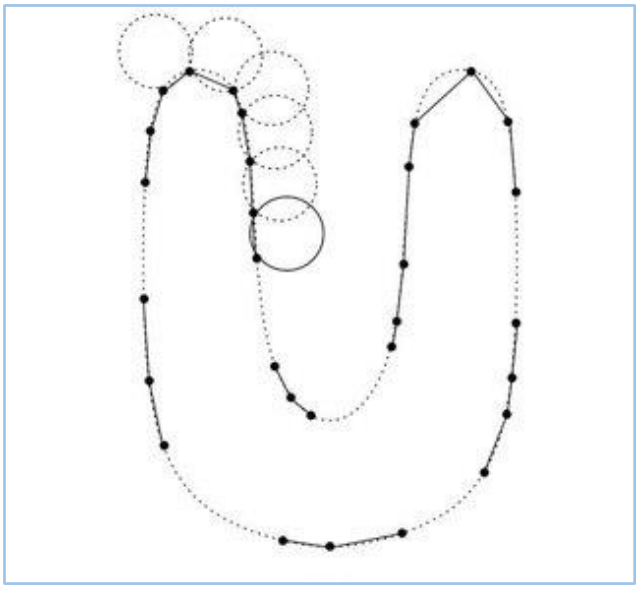


Figure 2. 5: Ball Pivoting Algorithm Representation [4]

BPA is particularly effective for well-structured, evenly sampled point clouds. It naturally preserves surface topology and handles smooth curves effectively. However, it is sensitive to gaps in data and may fail to reconstruct areas with low point density or noise. Additionally, it requires precise normal orientation for accurate results.

The following examples taken from the official website of Open3D library presents the results of the ball pivoting algorithm for the same point cloud (Rabbit point cloud). [4]

Figure 2. 6: Generated Rabbit Mesh Using BP

1.6.3 Poisson Surface Reconstruction

Poisson Surface Reconstruction is one of the most robust and widely used algorithms in the field of 3D scanning and surface reconstruction. Its objective is to convert a sparse, unstructured point cloud—typically obtained from 3D scanning—into a watertight 3D mesh that accurately represents the scanned object.

Unlike algorithms that work by connecting neighboring points directly (like Ball Pivoting), Poisson reconstruction approaches the problem from a global mathematical perspective. It treats the reconstruction task as a solution to a Poisson equation derived from the input point cloud and their estimated normals.

1.6.3.1 Principle of Poisson Reconstruction

The core idea behind Poisson reconstruction is based on vector field integration. Given a set of oriented points (points with associated normals), the algorithm interprets these normals as samples of the gradient of an indicator function—essentially a function that is 1 inside the object and 0 outside.

From this interpretation, the goal becomes to find the indicator function whose gradient best fits the input normals. This leads to solving the Poisson equation**:** [3]

$$\Delta\chi = \nabla . V \tag{2.2}$$

Where:

- X: is the indicator function to be reconstructed,
- V: is the vector field formed by the input normals,
- ∇: is the divergence of that vector field.

Once this function is computed, the surface can be extracted as an iso-surface (usually where χ=0.5) using marching cubes or a similar technique.

This global formulation makes Poisson Surface Reconstruction particularly robust to noise and outliers, as it does not rely on local connectivity alone.

1.6.3.2 Requirements

For Poisson Surface Reconstruction to produce meaningful results, certain conditions must be met:

- **Dense Point Cloud**: The input data must be reasonably dense and cover the object's surface sufficiently.
- **Accurate Normals**: Each point in the cloud must have a corresponding normal vector, preferably pointing outward from the object's surface. Normals can be estimated using nearest-neighbor analysis or based on the sensor's known geometry.

- **Watertight Surface Assumption**: Poisson reconstruction tends to close holes and fill missing areas, which is excellent for solid models but may not be suitable if the scanned object has intentional gaps or thin features.

1.6.3.3 Advantages and Limitations

Advantages:

- Robust against noise and missing data.
- Produces watertight meshes suitable for 3D printing and simulation.
- Works well with large datasets due to its global optimization approach.

Limitations:

- May generate unwanted geometry in areas with sparse data.
- Over-smoothing can occur, losing fine surface details.
- Normal vectors must be accurately estimated; poor normals lead to poor reconstruction.

The following example (Figure 2.7 and Figure 2.8) from Open3D library [4] presents the details PSR can cover in the mesh generation from the eagle point cloud

Figure 2. 7: Eagle Point Cloud

Figure 2. 8: Generated Eagle Mesh

1.7 Conclusion:

In this chapter, we established a theoretical framework for 3D scanning and reconstruction. We discussed how computers simulate 3D space, examined key scanning technologies, and reviewed the significance of normals and reconstruction algorithms. Among these, Poisson Surface Reconstruction (PSR) stood out as the primary method adopted in this thesis due to its robustness and reliability. This understanding provides the necessary background for the system design and implementation detailed in the following chapters.

Chapter 02: System Design

2.1 Introduction

In our study, Poisson Surface Reconstruction is applied after collecting the point cloud from the Time-of-Flight ultrasonic scanning system. Once the data is gathered and transformed into Cartesian coordinates, the normals are estimated programmatically based on the scanning path and local neighborhood analysis.

We then feed the processed point cloud into a Python-based reconstruction pipeline using the Open3D library. This library provides a built-in Poisson reconstruction method, making the integration smooth and efficient. The result is a smooth, watertight 3D mesh that approximates the scanned object. This model can be visualized, saved, or even exported for further use in simulation or 3D printing.

2.2 System description

The design of this device involves creating portable 3D scanner related to software on pc, the hardware makes the scan and sends it to the pc where the software collects the point cloud provided by the hardware, process the data and create 3D mesh that can be manipulated by 3D software.

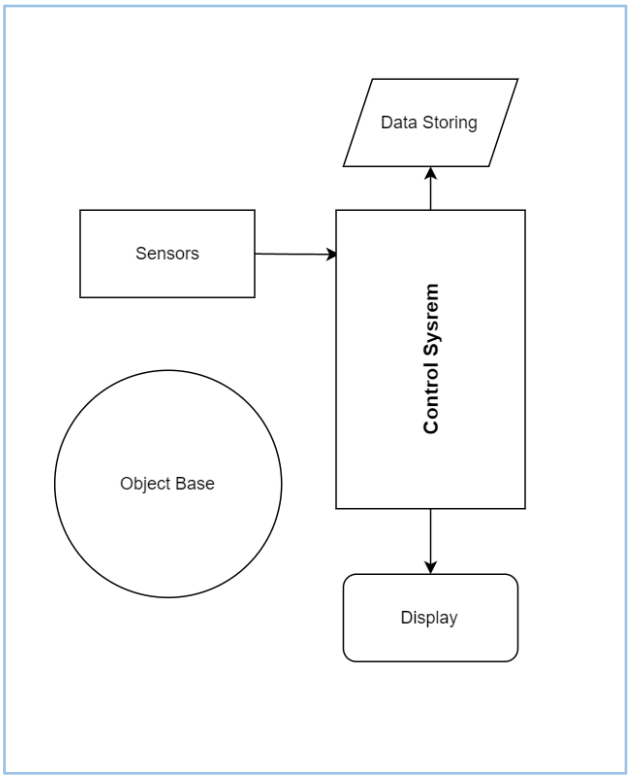


Figure 3. 1: Function Block of 3D scanner

2.3 Hardware Design

The 3D scanner is composed of two main systems: hardware and software. The hardware consists of a plastic platform that supports a rotating object base and a vertically adjustable sensor. These motions are powered by three stepper motors, each controlled via A4988 drivers and synchronized by a microcontroller.

The microcontroller manages motor movement, sensor readings, and the user interface, orchestrating the entire scanning sequence. It controls the rotation of the object and collects distance data using ultrasonic Time-of-Flight sensors. These sensors emit high-frequency sound pulses and listen for the returning echo—the delay in the echo determines the distance to the object's surface. By repeating this process across multiple positions with the help of a mechanical setup, the system computes 3D coordinates for each scanned point, gradually generating a complete point cloud of the object’s geometry.

A small LCD screen is attached to display real-time scan progress and system status. Additional feedback is provided by a buzzer that signals the start and end of scanning, and an LED that remains on during operation.

Captured data is stored on an SD card in CSV format, structured as a point cloud (X, Y, Z), which is later used for mesh reconstruction. The design focuses on being low-cost, simple to build, and effective for educational or prototype-level applications.

3.3.1 Function Block

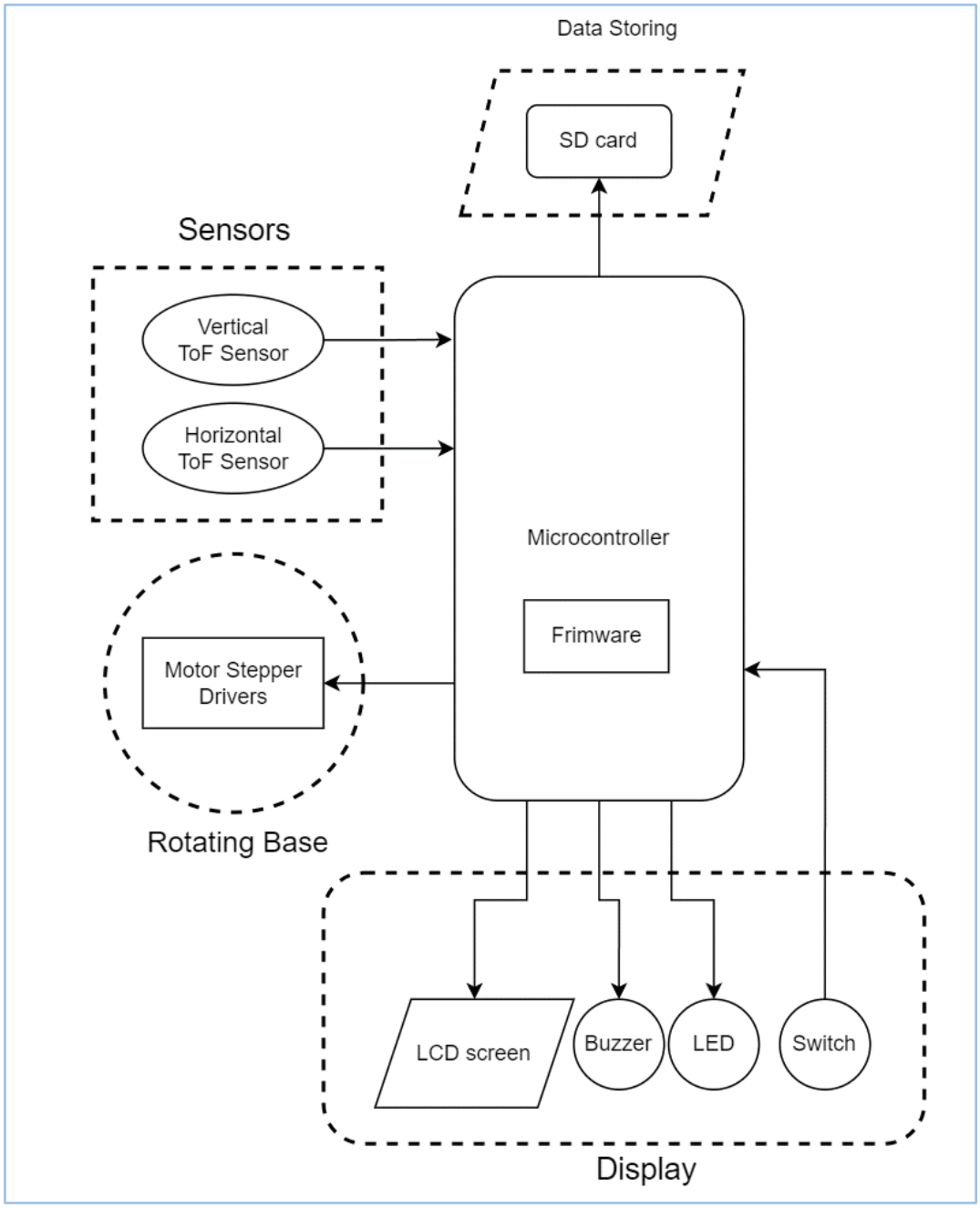


Figure 3. 2: Detailed Function Block

The core of the system is based on an AVR microcontroller from the Atmel family—specifically, the ATmega8A, an 8-bit microcontroller operating at a clock speed of 16 MHz. It offers 8 KB of flash memory and up to 28 general-purpose I/O pins, making it suitable for managing the scanner's various peripheral devices and control signals.

Among the key components connected to the microcontroller are the A4988 stepper motor drivers**.** Each driver can deliver up to 2 A of current, which is sufficient for the NEMA-series stepper motors used in the system to control the rotational and vertical movements required for scanning.

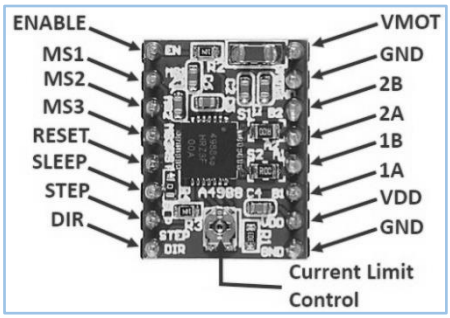


Figure 3. 3: A4988 Stepper Motor Driver

For distance measurement, the system uses the VL53L0X Time-of-Flight sensor. This sensor can measure distances from 20 mm to 800 mm, with an average precision of approximately ±5%. Its response time ranges from 20 ms to 200 ms, making it fast enough for real-time scanning applications. [5]

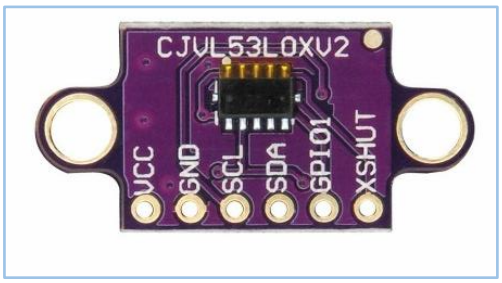


Figure 3. 4: VL53L0X Time Of Flight Sensor

The display is implemented through a 20×4 characters LCD equipped with an I²C interface module, allowing for simplified communication with the microcontroller using only two data lines. This display provides real-time feedback during the scan and allows users to configure settings such as scan start, resolution level, and operation status. It serves as the primary physical interface for interacting with the 3D scanner.

2.4 Software Design

2.4.1 Microcontroller Firmware

The core of the system is driven by embedded firmware written in the C programming language and uploaded to the microcontroller, which acts as the main controller managing all hardware operations. This firmware is responsible for handling the I/O operations, timing control, sensor data acquisition, stepper motor coordination, and user interaction through indicators and the LCD display.

The firmware starts by initializing all the connected peripherals, including digital I/O pins for stepper motor drivers, the I²C bus for sensor and screen communication, and the SPI interface used for SD card data logging. The VL53L0X Time-of-Flight distance sensor and the 2004 LCD screen are both interfaced via the I²C protocol, which significantly reduces the number of required wiring lines and simplifies communication between devices.

After taking the user's command (via button press or auto-start), the microcontroller triggers the distance sensor to take a measurement. At the same time, it drives the rotation stepper motor to complete a full 360° rotation at specific angular steps, synchronized with sensor reads. Once a full horizontal layer is scanned, the Z-axis stepper motor increments the height by 1 mm, and the process repeats until the full height of the object is scanned.

Each point measurement includes:

- The distance measured by the VL53L0X
- The current angular position from the rotation stepper (used to compute X and Y)
- The current height level (Z)

Using these values, the firmware performs real-time geometric calculations to convert polar data into Cartesian coordinates These coordinates are immediately written to the SD card in .csv format, which can later be processed by the Python software.

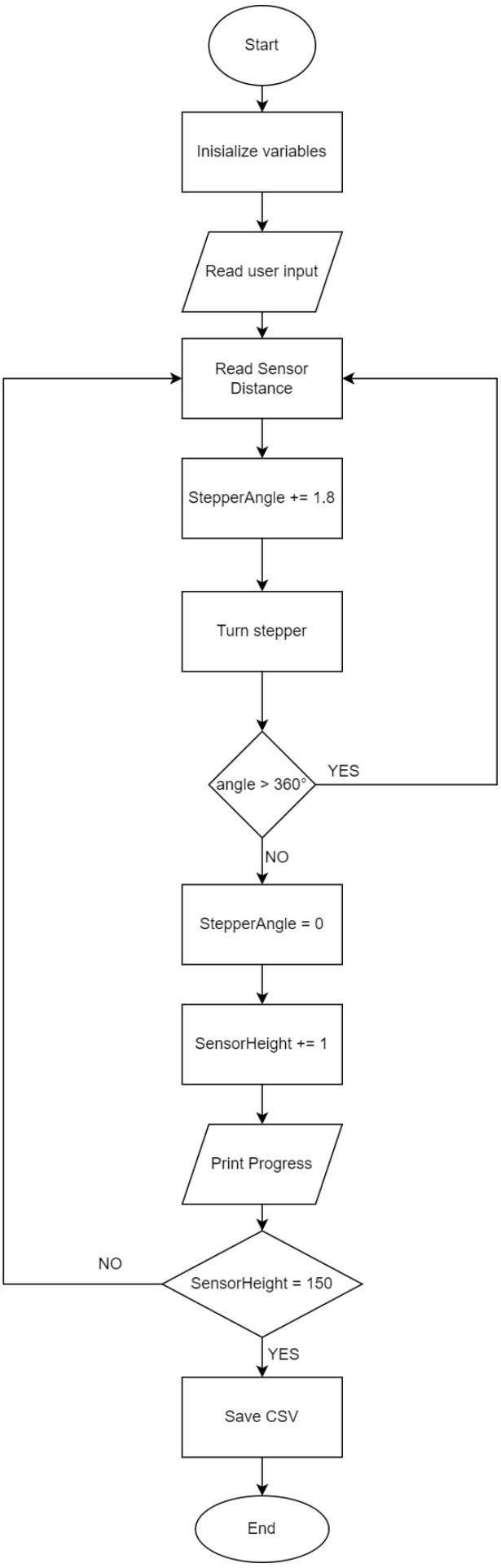
Additional firmware functionalities include:

- Basic user interface display via the LCD showing progress and status
- LED indicators for scanning state (active, finished, error)
- Buzzer signal for start and end of scan
- Implementation of a median filter in real-time to reduce noise during scanning
- Error handling routines for SD card access and sensor response timeouts

This firmware was optimized for the limited memory and computational capacity of the ATmega8 microcontroller, ensuring a stable and responsive system without the need for external processing during data acquisition.

2.4.1.1 Algorithm

The flowchart below outlines the firmware's operation in managing the hardware



2.4.1.2 Principle of Operation

The microcontroller serves as the central controller of the system. Once the user initiates the scan, the process begins with the sensor performing a 360° horizontal scan around the object at the initial height (Z = 0). This is achieved by rotating the platform using a stepper motor aligned along the Z-axis.

After completing the first layer, the vertical stepper motor raises the sensor by a fixed increment (1 mm), and a new rotational scan is performed at the next height level. This process repeats iteratively, layer by layer, until the full height of the object is scanned.

If a second sensor is installed in a vertical position facing the object, it performs a similar scanning sequence, capturing additional geometric details from a different perspective. All measurements are collected and saved in a CSV file on an SD card as 3D coordinates (X, Y, Z), forming the basis of the point cloud.

2.4.1.3 Point Cloud Creation

The VL53L0X sensor used in the scanner is a Time-of-Flight distance sensor. It measures the radial distance (R) between the sensor and the object's surface. Using basic trigonometry and system parameters, the 3D coordinates of each scanned point are calculated as follows:

$$X = R \times \cos\left(rad(\theta)\right) \quad (3.1)$$

$$Y = R \times \sin\left(rad(\theta)\right) \quad (3.2)$$

$$Z = h \quad (3.3)$$

Where:

R: the radius, which is the distance measured by the sensor

Theta(θ) : is the rotation angle which is 1.8° for our stepper

h : is the height defined by the system steps, which is 1mm in our design

The calculated coordinates are written to a .csv file on the SD card in real time. This file represents the raw point cloud data and is later used for mesh reconstruction and visualization on a computer.

2.4.2 3D Model Generation Software

2.4.2.1 Principle of Operation

The software pipeline for this project is designed to run on a PC. It is written entirely in Python, utilizing powerful libraries such as **NumPy**, **Open3D**, and **Matplotlib** to handle and process the scanned point cloud data efficiently.

Open3D is a widely used open-source library designed for working with 3D data. It provides a full set of tools for point cloud manipulation, surface reconstruction, and mesh visualization. In this project, Open3D was used to perform operations such as:

- *open3d.geometry.PointCloud():* to initialize and structure the point cloud object.
- *Vector3dVector():* to convert NumPy arrays (X, Y, Z) into a format usable by Open3D.
- *estimate_normals():* to compute surface normals for each point using a local neighborhood, which is crucial for Poisson reconstruction.
- *remove_statistical_outlier():* to filter out isolated or noisy points based on statistical distance thresholds.
- *create_from_point_cloud_poisson():* to apply Poisson Surface Reconstruction**,** generating a watertight triangle mesh from the oriented point cloud.
- *draw_geometries():* to visualize both the raw point cloud and the reconstructed mesh in a real-time 3D viewer.
- *write_triangle_mesh():* to export the final mesh in .stl format for further use (e.g., 3D printing or simulation).

These functions allowed the pipeline to clean, reconstruct, and render the data efficiently. Open3D's design supports both interactive and scripted workflows, making it ideal for research applications like this project. [4]

NumPy is the foundational library for numerical operations in Python, offering optimized matrix and vector manipulation. In this project, it played a key role in data loading and preprocessing:

- *numpy.loadtxt()* and *numpy.genfromtxt():* used to read the CSV file containing the point cloud coordinates (X, Y, Z).
- *numpy.cos()* and *numpy.sin():* applied to convert raw polar coordinates into Cartesian space using geometric formulas.
- numpy.median() and slicing techniques: used to implement the median filter, which removes inconsistent distance values from scan data.

General array indexing and manipulation allowed fast computation of point coordinates, filtering, and real-time updates.

NumPy's efficiency and simplicity made it the backbone of all mathematical operations within the scanning and reconstruction pipeline. [7]

Matplotlib is a 2D visualization library used here for quick inspection and verification of point cloud data during development. Specific uses include:

- *plt.figure()* and *Axes3D*: to initialize 3D plotting environments.
- *ax.scatter(X, Y, Z)*: to plot point clouds in three-dimensional space and observe structure, distribution, and density.
- *plt.title(), plt.xlabel(), plt.ylabel():* to annotate figures for better clarity during debugging and validation stages.

These visual previews helped evaluate the scan results before and after filtering, ensuring the accuracy of the dataset before feeding it into reconstruction. [8]

The full reconstruction pipeline is divided into five key stages: Data Loading**,** Pre-Processing**,** Mesh Construction**,** Visualization and Interaction**,** and Exporting. Each stage uses a combination of these libraries to handle complex tasks—from converting raw scan data into accurate 3D models to displaying and exporting the final result.

initially, the system loads the data recorded during the scan, which consists of point coordinates saved in a .csv file. This file is parsed and converted into structured arrays using NumPy for

efficient manipulation. Afterward, the point cloud undergoes preprocessing, where noise is filtered out, and normals are estimated. The estimation of normals is a fundamental step in surface reconstruction because it provides orientation information for each point, which is critical for generating smooth and continuous meshes.

Once the point cloud is clean and prepared, the mesh construction begins. This is done using the Poisson Surface Reconstruction (PSR) algorithm mention open 3D. This method interprets the set of oriented points as samples from the surface of a 3D object and tries to rebuild a watertight surface that best fits these samples. The algorithm simulates a mathematical "drop" of virtual material over the point cloud, allowing the final mesh to capture fine surface details while remaining smooth.

After constructing the mesh, the result is rendered in a 3D visualization window where users can interactively inspect the model. They can zoom, rotate, and pan to assess mesh quality, verify reconstruction integrity, and make any minor adjustments if necessary. Once validated, the mesh is exported in *STL* format—a widely used format in 3D printing and CAD applications.

This software was designed to provide a full pipeline from raw scan to exportable mesh, offering both automation for experienced users and interactivity for flexibility.

2.4.2.2 Data Loading

The data loading module is responsible for reading the .csv file that contains the scanned point cloud. The file includes three columns representing the X, Y, and Z coordinates of each captured point. These values are interpreted using the NumPy library and stored in a 2D array, which makes it easy to process and manipulate the data later.

This step also includes basic validation to ensure that the file is not corrupted and that all data points are numerically valid. The number of points and the dimensional consistency are also verified before moving to the next stage.

2.4.3 Pre-Processing

Pre-processing aims to prepare the point cloud for accurate mesh reconstruction. This stage involves:

- **Outlier Removal**: Random errors and stray reflections can introduce outlier points that do not belong to the object's surface. These are filtered using statistical methods such as radius-based or k-nearest-neighbor filtering.
- **Voxel Down sampling (Optional)**: Reducing the number of points using a voxel grid, which simplifies the model without significant loss in shape fidelity. This is useful for accelerating later computations.
- **Normal Estimation**: Estimating the surface normal vectors at each point. This is a critical step for surface reconstruction algorithms like PSR, as the quality of the mesh is highly dependent on the accuracy of these normals.

After these steps, the cleaned point cloud is briefly visualized using Matplotlib to confirm that the geometry appears correct and complete before moving on.

2.4.4 3D Model Generation

2.4.4.1 Poisson Surface Reconstruction

Poisson Surface Reconstruction is a technique that generates a watertight surface from a set of oriented points. [3] It treats the problem as a spatial Poisson problem, where it tries to find a function, whose gradient best fits the normal vectors of the point cloud. This method is highly robust and can handle small gaps and irregularities in the data, making it ideal for noisy or incomplete scans.

The key parameters used in PSR include:

- **Depth**: Controls the resolution of the resulting mesh. Higher depth results in finer details but increases computation time.
- **Scale**: Determines how closely the mesh fits the point cloud. A lower scale smoothens the mesh, while a higher value captures finer surface features.
- **Linear Fit**: An optional optimization that increases accuracy on smoother surfaces.

The algorithm generates a mesh by solving the Poisson equation and extracts the surface as a triangle mesh that closely follows the input data while filling small holes naturally.

2.4.4.2 Ball Pivoting

The Ball Pivoting Algorithm (BPA) is a local surface reconstruction method that builds a mesh by rolling a virtual ball over the point cloud and forming triangles where the ball touches three points. Unlike PSR, which uses a global implicit function, BPA connects points directly and is more suited for clean and uniformly sampled data.

In this project, BPA is considered as an alternative to PSR for its ability to preserve local surface details without requiring oriented normals or solving complex equations.

Key parameters include:

Ball Radius: Determines the sensitivity of surface detection. It must be chosen based on point spacing.

Multiple Radii Improves reconstruction in cases where point density varies.

BPA offers faster computation and better control over local geometry, making it ideal for low-noise scans where watertightness is not essential.

For our project we choose to prioritize PSR for it stability and large capability of generating simple and complex models from point clouds with the condition of well calculated normal vectors.

2.5 Software and Hardware Tools

2.5.1 Software

1. **Blender**: An open-source 3D modeling software based on C++ and Python, used to generate synthetic point clouds from 3D models. It was essential in simulating different scan scenarios to test reconstruction algorithms like PSR. [9]
2. **FreeCAD**: A parametric CAD tool used to design the scanner's physical casing and custom mechanical parts for 3D printing. [10]
3. **KiCad**: A powerful and open-source electronic design automation (EDA) tool used to create schematic diagrams and PCB layouts for the scanner's control circuit. [11]
4. **PyCharm**: A Python IDE used to develop all scripts for data loading, filtering, mesh reconstruction, and the GUI that controls the 3D visualization and export process. [12]

2.5.2 Hardware

1. **3D Printer**: Used to fabricate the plastic components of the scanner, including the rotating platform and sensor mount structures.
2. **Soldering Station**: Used to assemble the scanner's PCB, connecting the microcontroller, drivers, sensors, and peripheral components.
3. **CAO Laboratory**: Provided tools and environment for PCB manufacturing and prototyping of the electronic system.

2.6 Graphical User Interface (GUI)

To enhance usability, a simple and functional graphical user interface was implemented using Python-based libraries. The GUI guides the user through the full pipeline without requiring command-line operations. This makes the tool accessible even for users who are not familiar with Python or 3D data processing. The integration of interactivity through Open3D ensures real-time manipulation of the mesh, helping users verify the reconstruction visually before export. The GUI provides buttons and input fields for:

- Loading Point Cloud Files
- Setting Pre-processing Parameters (e.g., voxel size, outlier thresholds)
- Choosing the preferred algorithm
- Tuning PSR/BP Parameters (e.g., depth, scale)
- Running the Reconstruction
- Visualizing the Final Mesh in 3D
- Exporting the Model in ".stl" Format

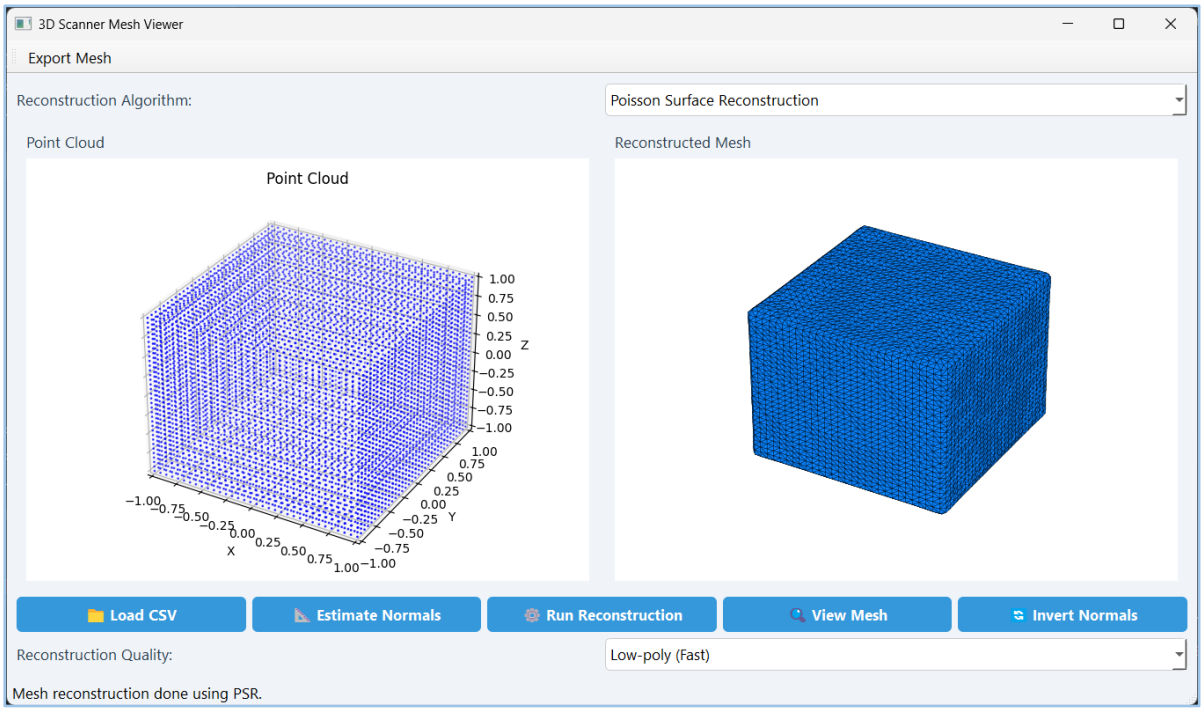


Figure 3. 5: Graphic User Interface

2.7 Conclusion:

The system architecture described in this chapter successfully integrates mechanical design, embedded control, and software processing into a cohesive 3D scanning platform. The use of low-cost components makes the solution accessible, while the software ensures a functional and efficient mesh reconstruction pipeline. By implementing essential features like real-time coordinate transformation, point cloud storage, and GUI-based reconstruction tools, the system is well-prepared for prototype testing and further enhancements.

Chapter 03: Prototyping, Tests and Results

3.1 Introduction

To reduce complexity and adapt to the available components, a simplified prototype of the 3D scanner was developed. This reduced version is based on a single VL53L0X Time-of-Flight sensor, one rotational axis, and one vertical translational axis, allowing the system to perform basic 3D scans. Despite being a scaled-down model, it replicates the scanning logic of the full system and serves as a proof-of-concept.

3.2 Hardware

The mechanical structure was 3D-printed and mounted on a stable plastic base. It supports two stepper motors: one for rotating the object (around Z-axis) and one for moving the sensor vertically (along Z-axis). The VL53L0X sensor is fixed in front of the object and measures radial distances layer by layer. The system is compact, low-cost, and tailored for small object scanning (≤15×15 cm).

3.3 Hardware Components

The electronic control system is based on an ATmega8 microcontroller. It manages motor control, sensor reading, and SD card storage. The circuit includes:

- Two A4988 motor drivers (for stepper motors),
- VL53 sensor,
- An SD card module for saving point cloud data in CSV format,
- A 16×2 LCD screen for live feedback and scan progress display.

Power is supplied via a regulated 5V source, ensuring stability for both logic and motor systems.

3.4 Scanning Process

The scanning Process follows a layered approach. The microcontroller rotates the object one full cycle (360°) at a fixed angle step (1.8°), then lifts the sensor by 1 mm and repeats the scan. This continues until the desired height is reached. For every angle-height pair, the sensor measures the radial distance.

Initially, the raw data collected contained significant noise, especially due to reflections, mechanical vibrations, and sensor error margins. The base algorithm was then improved by integrating a median filter to each set of radial measurements per layer, reducing outliers and increasing point cloud quality.

3.5 Scanning platform design

The scanning platform was specifically designed to fulfill the mechanical requirements of the 3D scanning process. It features a cylindrical rotating base with a diameter of 15 cm, where the scanned object is placed. This base is directly coupled to a Z-axis rotational stepper motor**,** allowing for precise 360° rotation during scanning.

On the left side of the platform, a vertical support structure holds the Z-axis translational stepper motor, which controls the height of the sensor during scanning. This support is reinforced by guiding rods that extend upward and connect at the top using a rigid fixture, ensuring mechanical stability and smooth vertical motion.

On the front-left section, a small control box houses the main motherboard (microcontroller-based), with a user interface composed of an LCD screen and a rotary encoder. This interface allows the user to interact with the system, set scanning parameters, and monitor real-time progress during the scan.

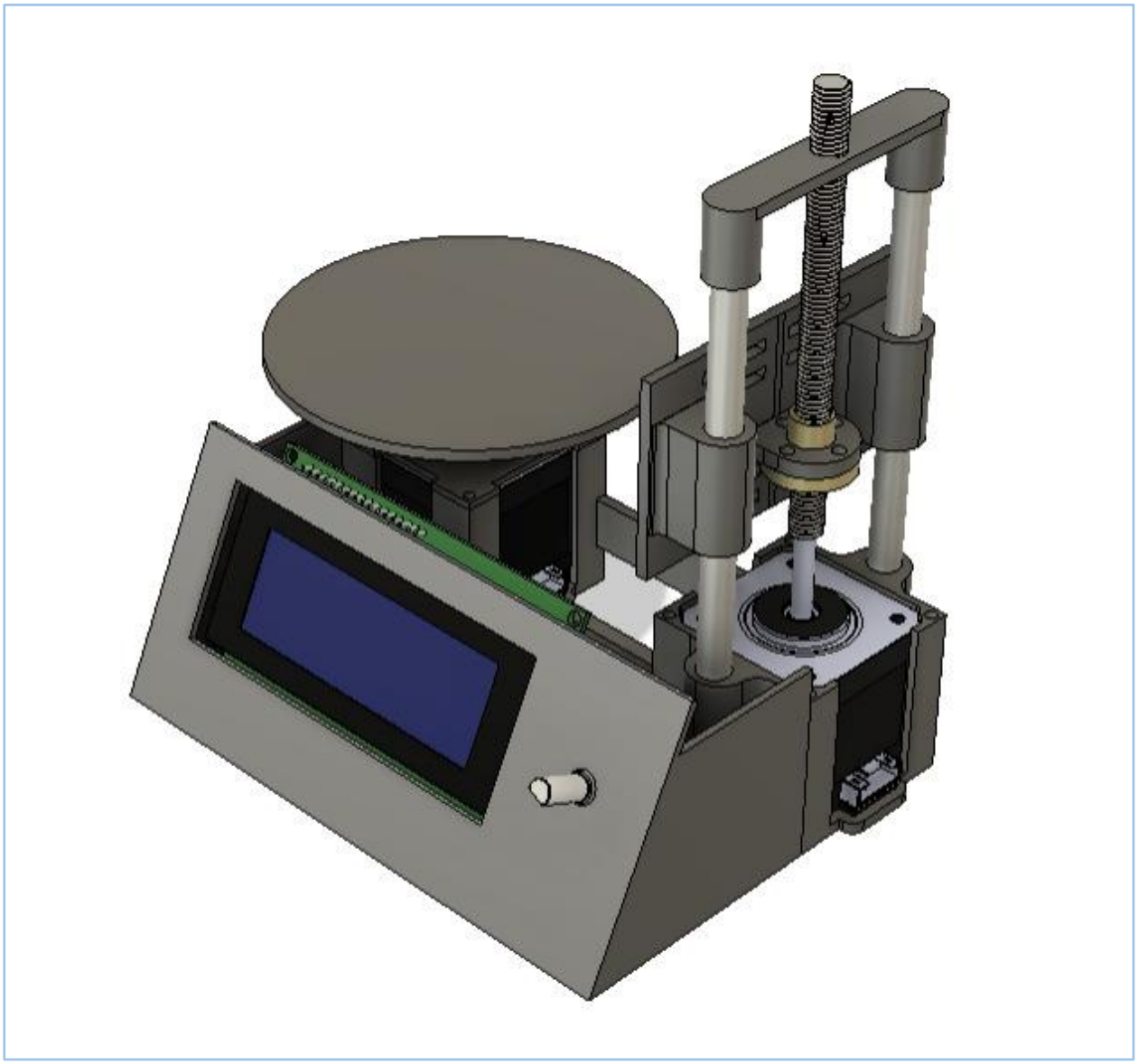


Figure 4. 1: 3D Scanner Prototype

3.6 Software

The developed software includes:

- Microcontroller firmware written in C, handling scan logic and SD card storage,
- 3D Model Generation Software: a python scripts on the computer for parsing CSV data, visualizing point clouds, reconstructing the mesh using Poisson Surface Reconstruction (PSR), and exporting it as STL files.

The pipeline includes data loading, preprocessing (filtering, outlier removal), normal estimation, mesh generation, and visualization through a user-friendly GUI.

3.7 Tests

A full scan was performed on a cone-shaped object (10 cm height × 10 cm diameter) using three resolution settings: low, medium, and high. The initial results, without any filtering, showed a highly noisy point cloud as illustrated. The data was scattered and inconsistent due to surface irregularities and sensor noise.

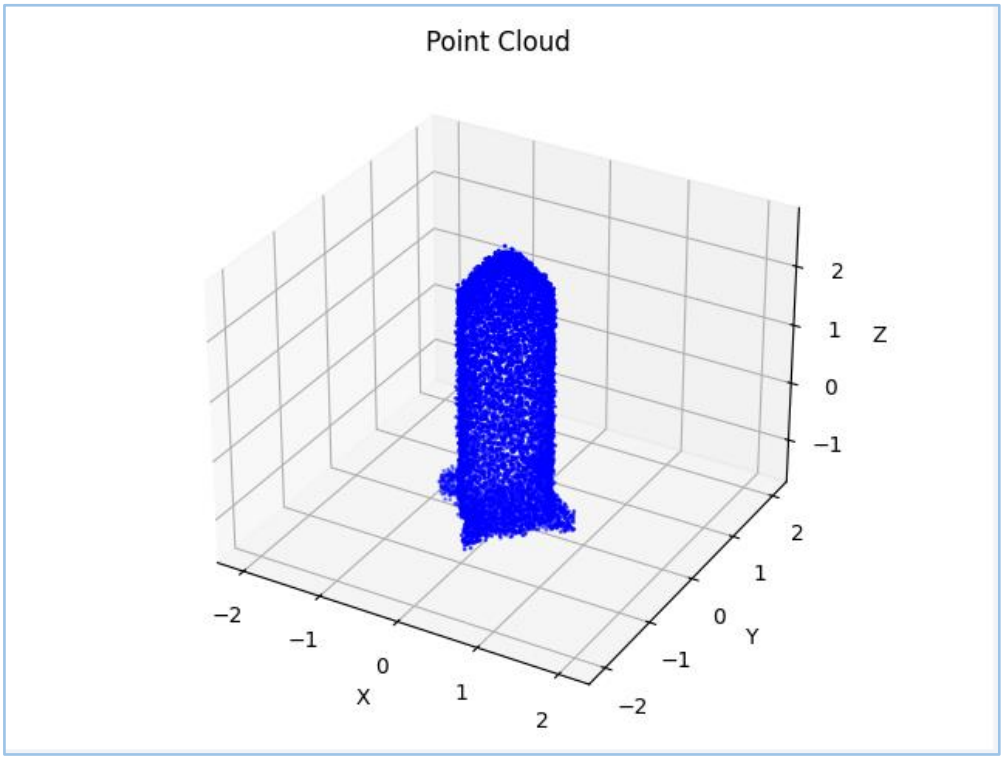


Figure 4. 2: Noisy Captured Point Cloud

After integrating the median filtering stage in the scanning process, the results showed considerable improvement. The updated point cloud was significantly cleaner and more coherent. Points were more concentrated around the object's surface, which helped improve the resulting mesh fidelity.

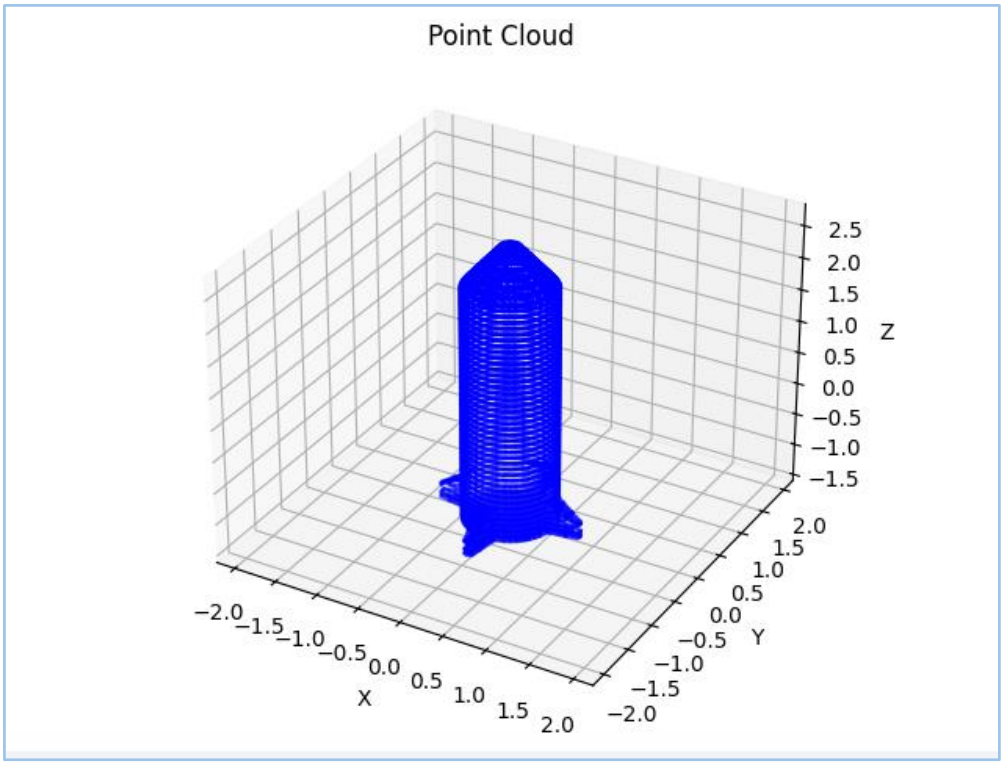


Figure 4. 3: Filtered Point Cloud

3.8 3D Model Generating

The filtered point cloud was processed using the Poisson Surface Reconstruction (PSR) script written in Python. The meshes for the three resolution levels were successfully reconstructed and visualized. Despite the limited scan angles and single-sensor design, the reconstructed 3D models provided a fairly accurate representation of the object.

The first figure presents the lowest quality mesh generation and the fastest, in the other hand the slowest and the more detailed mesh generation is presented in following figure.

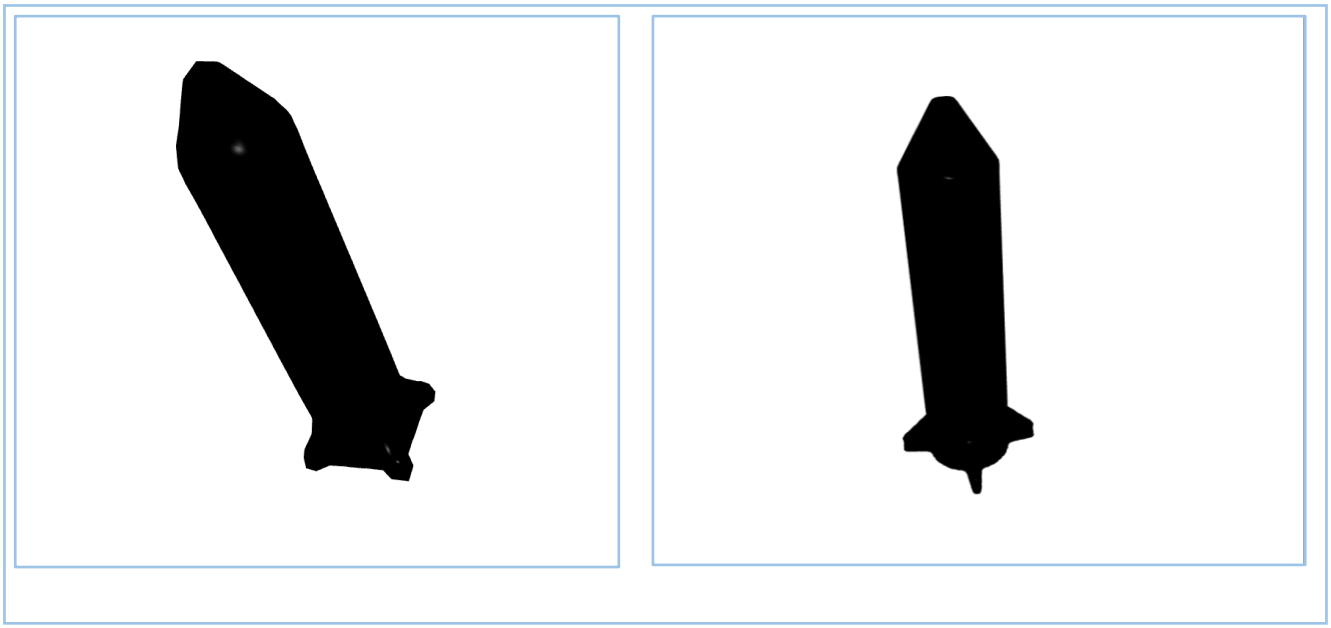


Figure 4. 4: Generated Mesh Comparison (a. Fastest lowest quality, b. Slowest more detailed)

3.9 Future enhancements

The developed prototype has proven to be stable and functional for small-scale, medium resolution scans using a single VL53L0X Time-of-Flight sensor. While the results are satisfactory for personal projects and educational demonstrations, the system's design and performance can be significantly improved to reach higher levels of precision, reliability, and usability. Several recommendations are proposed for future enhancements.

These recommendations aim to elevate the scanner from a basic prototype to a fully functional, high-performance system. Each proposed enhancement is modular and can be implemented incrementally depending on project scope, budget, and application goals. Future work may focus on combining some of these improvements to validate the system in real-world scanning scenarios.

1. **Multi-Sensor Integration:**

   One of the most impactful upgrades would be the integration of multiple VL53L0X sensors around the object or mounted at varying angles. This configuration allows simultaneous data acquisition from different perspectives, reducing blind spots and improving the overall surface coverage. It would also minimize scanning time and allow the system to capture complex geometries that a single-sensor setup might miss.

The current prototype operates with one rotational axis and one vertical translational axis. Adding a second rotational axis (such as tilt or horizontal rotation) would enable the scanner to inspect undercuts, cavities, and internal features. This would enhance the completeness of the point cloud and allow reconstruction of more complex or irregular shapes.

2. **Advanced Sensors (LIDAR or Structured Light)**

   Upgrading the system to use LIDAR or structured-light sensors (like Intel RealSense or Microsoft Azure Kinect) would provide higher resolution depth data and improve accuracy dramatically. These sensors can also deliver surface normal vectors directly, which are essential for precise mesh reconstruction—especially when using algorithms like PSR or Ball Pivoting. Although more costly, these sensors make the scanner suitable for professional applications like digital archiving or quality inspection.

3. **Improved Mechanical Precision**

   Using high-quality stepper motors with finer micro stepping, or replacing them with servo motors, could greatly increase the precision of movement. Additionally, integrating end-stops and a homing system would make the setup more repeatable and robust. Ensuring mechanical rigidity and reducing vibration would help in minimizing measurement noise.

4. **Microprocessor Upgrade**

   Replacing the microcontroller (ATmega8) with a Raspberry Pi or another microprocessor would introduce several advantages:

   - Onboard data processing and visualization
   - Faster scan-to-mesh pipeline
   - Graphical user interface directly on the scanner
   - Wi-Fi/Bluetooth capabilities for remote control and data transfer
   - Ability to run Python scripts for mesh reconstruction locally

   This would turn the scanner into a standalone smart device, reducing the need for constant PC interaction.

5. **Dynamic Filtering and Noise Handling**

   Although a median filter was effective in cleaning up the raw point cloud, more advanced filters could be implemented, such as Kalman filters or adaptive outlier rejection algorithms. These could be applied in real time during data acquisition, leading to cleaner data and reducing the load on post-processing stages.

6. **User Interface and Automation**

   A more interactive and intelligent user interface could guide users through the scanning process with real-time feedback. Automated calibration routines, resolution adjustment, and object alignment features could greatly enhance ease of use and reliability of scans.

7. **Battery-Powered and Portable Design**

   Future versions could be built into a compact, battery-powered, and wireless system. This would make the scanner portable and deployable in remote or field environments, extending its range of application to on-site inspections or outdoor scanning.

8. **Data Format and Cloud Connectivity**

   Supporting multiple export formats such as OBJ, PLY, or GLTF would make the system more versatile. Additionally, integrating cloud upload features could allow users to store, share, or process the data remotely, facilitating collaboration or large-scale data handling.

9. **Application-Specific Modifications**

   Depending on the target use-case (e.g., medical imaging, archaeology, or manufacturing), the scanner can be tuned with specialized algorithms, hardware enclosures, or mounting systems. Tailoring the setup for specific materials, shapes, or surface finishes could also improve results significantly.

3.10 Conclusion

The prototype demonstrated the feasibility of low-cost 3D scanning using a single sensor and a minimal hardware setup. Test results showed that, despite the limitations of resolution and coverage, usable 3D models could be generated with acceptable accuracy. The integration of filtering significantly improved point cloud quality, enabling smoother mesh reconstruction. The system proves to be a promising base for further development into a more refined and feature-rich 3D scanning tool.

General Conclusion

This memoir presented the design and development of a cost-effective 3D scanner based on a Time-of-Flight (ToF) distance measurement principle. The prototype was designed with a simplified hardware structure using a single VL53L0X sensor, one rotational axis, and one vertical axis to capture layer-by-layer point clouds. The mechanical structure, control electronics, and firmware were developed in-house, making the system modular, low-cost, and suitable for educational or experimental use.

The point cloud data collected was processed using Python scripts, and mesh reconstruction was achieved using the Poisson Surface Reconstruction algorithm through the Open3D library. Tests on objects such as a 10×10 cm cone demonstrated the scanner's ability to acquire and reconstruct basic 3D geometry, with improved results achieved by integrating a median filter into the data acquisition algorithm.

The project successfully validated the feasibility of low-cost 3D scanning using embedded systems. Despite its limitations in resolution and scanning coverage, the prototype forms a solid foundation for further development.

Future improvements could include adding more sensors, integrating higher-precision scanning technologies like LIDAR or structured light, and replacing the microcontroller with a microprocessor to allow on-device processing and a full standalone system. These enhancements could expand the scanner's application to reverse engineering, educational labs, and even 3D printing preparation.

References

Books & Academic Papers

[1]. Curless, B., & Levoy, M. (1996). *A volumetric method for building complex models from range images*. SIGGRAPH.

[2]. Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. Springer.

[3]. Kazhdan, M., Bolitho, M., & Hoppe, H. (2006). *Poisson Surface Reconstruction*. Symposium on Geometry Processing.

Websites Documentation

[4]. Open3D Team. (2023). *Open3D Documentation*. https://www.open3d.org/docs/latest/tutorial/ (05-2025)

[5]. STMicroelectronics. (2016). *VL53L0X Time-of-Flight Ranging Sensor Datasheet*.

[6]. Intel RealSense. (n.d.). *Depth Camera Technology Overview*. https://www.intelrealsense.com/technology/ (05-2025)

[7]. Numpy Library: https://numpy.org/doc/stable/ (03-2025)

[8]. Matplotlib Library documentation https://matplotlib.org/ (03-2025)

Software Tools & Platforms

[9]. Blender Foundation. (2024). *Blender Manual*. https://docs.blender.org/

[10]. FreeCAD Project. (2024). *FreeCAD Documentation*. https://wiki.freecad.org/

[11]. KiCad Project. https://www.kicad.org/

[12]. PyCharm – JetBrains. https://www.jetbrains.com/pycharm/