MAJOR PROJECT PHASE-II REPORT

Non-Invasive 3D

BY

Pranjal Maheshwari
Faculty No: 21COB287
Enrolment No: GL5402

Yusuf Hasan
Faculty No: 21COB280
Enrolment No: GK7123

Under the Guidance of

Prof. Saiful Islam

Department of Computer Engineering
Zakir Husain College of Engineering and Technology
Aligarh Muslim University (AMU)
Aligarh, India
Academic Year: 2024-2025

Dated: _____

Certificate

This is to certify that the Project Report entitled **"Non-Invasive 3D"**, being submitted by **Pranjal Maheshwari** and **Yusuf Hasan**, in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Engineering**, during the session 2024-25, in the Department of Computer Engineering, **Zakir Husain College of Engineering and Technology, Aligarh Muslim University, Aligarh**, is a record of candidate's own work carried out by him under my supervision and guidance.

Prof. Saiful Islam
Department of Computer Engineering
ZHCET, AMU

ACKNOWLEDGEMENT

We would like to express our thanks and sincere gratitude to our supervisor **Prof. Saiful Islam**, Computer Engineering Department, ZHCET, AMU, for his unwavering support and encouragement during the course of the project. We are thankful to him for providing vital inputs, wherever required, in keeping our work on the right path. His patience, availability, and vast knowledge of computing helped us complete the project.

Yusuf Hasan

Pranjal Maheshwari

Contents

List of Figures

ABSTRACT

The 3D Hand Scanner project is an innovative system designed to capture high-quality 3D scans of the human hand, overcoming the challenges that traditional scanners face with this complex subject. Hands, with their intricate shape and varying textures, are difficult to scan accurately using conventional methods. This project addresses these challenges by employing a flexible, low-cost architecture that utilizes multiple Raspberry Pi cameras. These cameras take simultaneous photographs from multiple angles, which are then processed into detailed 3D models using photogrammetry.

The development process involved extensive prototyping, with various design iterations focusing on optimizing camera setups, lighting conditions, and the coordination of the Raspberry Pi network. The final system is robust and scalable, capable of producing precise scans without requiring complex calibration. The user interface is intuitively designed to guide both operators and subjects through the scanning process, ensuring consistent and accurate results every time.

This 3D Hand Scanner project opens up significant opportunities in fields like hand therapy and personalized product design. By enabling accurate and reproducible 3D scans, it facilitates the creation of custom-fit products, such as 3D-printed braces, that are not only highly functional but also comfortable, durable, and reusable. The system's potential applications extend beyond healthcare, offering a versatile platform for innovative product designs and personalized manufacturing solutions.

In summary, this project represents a substantial advancement in 3D scanning technology, particularly for applications involving the human hand. Its combination of affordability, precision, and user-friendliness makes it a valuable tool in both medical and design fields, driving new possibilities in personalized care and product innovation.

1 INTRODUCTION

1.1 Overview of 3D Hand Scanning Technology

3D hand scanning is a sophisticated process aimed at capturing the complex geometric structures of the human hand. The human hand, with its intricate anatomical design, consists of 27 bones and multiple degrees of freedom, making it one of the most complex parts of the human body to digitize accurately. Achieving a high-resolution and detailed 3D scan requires precise synchronization between hardware and software to capture the hand's fine features in a short time frame, especially given the natural movement of the hand during scanning. The introduction of 3D scanning technology into the realm of reverse engineering, ergonomics, medical prosthetics, and other high-precision industries has revolutionized how human anatomy is captured and modeled in the digital space. Unlike traditional scanning methods that often rely on manual measurements or contact-based approaches, modern non-invasive 3D hand scanners operate without physically touching the subject, minimizing distortion or discomfort. This non-invasive approach is particularly important for applications where high accuracy and comfort are priorities, such as medical and ergonomic assessments. 3D scanners primarily function by generating point clouds, which are dense sets of data points in 3D space that represent the scanned object's surface. These point clouds are then converted into usable 3D models through mesh generation algorithms. In the case of hand scanning, capturing detailed finger joints, skin texture, and even subtle features like veins or contours of the skin is crucial for accurate modeling.

1.2 Technologies Behind Non-invasive 3D Hand Scanners

The hardware and software components behind 3D hand scanners can vary based on their use cases. Advanced systems employ a range of scanning technologies, each with its own strengths and weaknesses. Among the most common methods are laser triangulation, structured light, and photogrammetry. Laser Triangulation: Laser triangulation works by projecting a laser beam onto the object being scanned and using sensors to detect the reflected light. The triangulation of this light enables the calculation of precise distances, thus allowing the creation of highly accurate 3D models. This technique is commonly used in industrial applications where high precision is required but tends to be costly. Structured Light Scanning: This method uses a projector to cast a pattern (typically a grid or series of lines) onto the object, and cameras record the deformation of these patterns as they fall on the surface of the object. The software then calculates the depth and surface details based on the deformation. Structured light scanning is highly effective in capturing small details and is often preferred in medical and ergonomic applications where precise surface mapping is crucial. Photogrammetry: Photogrammetry is a cost-effective alternative to laser-based and structured light scanning systems. This

method involves taking multiple overlapping photographs of an object from different angles and then using specialized software to reconstruct the 3D model from these images. Photogrammetry excels in applications where budget constraints are critical, as it leverages standard imaging hardware like digital cameras. However, the quality of the final 3D model depends heavily on the resolution of the images and the effectiveness of the reconstruction software.

1.3 The Role of Low-cost Solutions in Expanding 3D Hand Scanning

The development of low-cost, open-source 3D hand scanning systems has democratized access to this technology, especially in educational, research, and small-scale commercial settings. A notable example is the Raspberry Pi-based photogrammetry hand scanner presented by Yang et al. (2021). This modular system uses 50 Raspberry Pi units equipped with 8-megapixel cameras, all synchronized to capture the hand from various angles. The system captures 96This modular design allows for easy assembly and reconfiguration based on the scanning requirements. With a total hardware cost of approximately 1,650 Euros, the system is affordable compared to traditional high-end scanners that can cost over 150,000 Euros. The ability to capture an entire hand scan in under 100 ms reduces the likelihood of movement artifacts, a common issue when scanning body parts that cannot remain perfectly still for extended periods. This accessibility and ease of use make low-cost systems highly appealing for applications such as custom prosthetic design, where capturing the exact dimensions and contours of a patient's hand is essential for creating a well-fitting prosthetic. Additionally, the Raspberry Pi system is wireless, allowing for flexibility in setup and operation. The captured images can be reconstructed into a 3D model using software like Agisoft Photoscan, providing high accuracy with a mean absolute error of just 0.62 ± 0.28 mm compared to professional-grade scanners like the Artec Spider.

1.4 Applications in Various Fields

The use of non-invasive 3D hand scanning has expanded rapidly across various industries, providing significant benefits in terms of accuracy, efficiency, and cost savings. Some of the primary areas of application include: 4.1 Healthcare and Prosthetics In the medical field, 3D hand scanning plays a crucial role in designing prosthetics. Traditional methods of designing prosthetics often require time-consuming molds and measurements, which could result in discomfort for the patient. With 3D scanning, accurate digital models of the patient's hand can be created quickly and non-invasively, allowing for customized prosthetics that fit perfectly. Furthermore, 3D scanning is being used in surgical planning, where precise models of the hand or other body parts are required before complex procedures. Surgeons can use these models to practice and plan surgeries, improving the success rates of operations.

1.5 Industry 4.0 and Reverse Engineering

In the context of Industry 4.0, 3D scanning technology is becoming integral to the manufacturing process, particularly in reverse engineering and quality control. By scanning existing products or components, manufacturers can recreate detailed CAD models without the need for original blueprints, facilitating the reproduction of parts that may be out of production or difficult to source. 3D scanning also improves the efficiency of the design and prototyping process, allowing engineers to quickly capture the geometry of physical objects and use that data in digital environments. This capability is especially important in industries like automotive, aerospace, and consumer electronics, where precision is paramount and any deviation from design specifications can result in product failure.

1.6 Virtual Reality and Human-Computer Interaction

In fields like virtual reality (VR) and human-computer interaction (HCI), 3D hand scanning allows for the creation of realistic hand models that can be used in digital environments. This is crucial for developing immersive VR experiences where hand movement and interaction with virtual objects must be as lifelike as possible. Accurate hand models can be integrated into VR systems to provide more natural user interactions, significantly enhancing the user experience.

1.7 Personalized Product Design

Ergonomic studies and personalized product design also benefit greatly from 3D hand-scanning technology. For instance, custom-made gloves, tools, or handles can be designed to fit the user's hand perfectly, improving comfort and functionality. In this context, the low-cost hand scanner developed by Yang et al. offers an accessible solution for small businesses and independent designers who require precise 3D models but may not have the budget for high-end scanners.

2 LITERATURE SURVEY

3D scanning technology has revolutionized industries such as medicine, architecture, and manufacturing by offering precise methods to digitize real-world objects into 3D models. Among the various 3D scanning techniques, photogrammetry has emerged as a versatile and accessible approach. It allows the generation of detailed 3D models using a series of overlapping 2D photographs. For applications like hand scanning, photogrammetry offers several advantages, including non-invasiveness, cost-effectiveness, and the ability to capture intricate details of the hand's geometry. While other 3D scanning methods like laser scanning and structured light scanning offer higher precision, photogrammetry remains favored due to its lower cost and broader accessibility. This literature survey explores the role of photogrammetry in 3D hand scanning, discusses the evolution of other 3D scanning techniques, and emphasizes the comparative benefits and challenges of each method. Photogrammetry, particularly when enhanced with AI-based algorithms, has seen significant advancements, making it a key player in applications requiring detailed yet affordable 3D modeling [1]. Photogrammetry reconstructs 3D models from 2D images by identifying common points between photographs and calculating their depth and spatial positions. The method involves triangulation, where the relative position of the camera and the object is used to generate a 3D model. Compared to other 3D scanning methods, photogrammetry is more accessible, as it can be done using consumer-grade devices like smartphones, which can be enhanced by depth sensors or stereo cameras. For hand scanning, photogrammetry captures intricate details such as finger joints, wrinkles, and palm contours, making it ideal for applications in prosthetics, medical imaging, and ergonomics. Research by Alhamadah et al. demonstrated that even with consumer-grade devices, photogrammetry can achieve less than 5Photogrammetry-based hand scanning typically involves taking a series of photographs from multiple angles, ensuring that there is significant overlap between images. These images are then processed using specialized software, which identifies common features in the images, aligns them, and generates a point cloud that can be converted into a 3D mesh. The advantage of this method lies in its simplicity, low cost, and scalability, making it ideal for both academic and industrial applications.

2.1 Evolution of 3D Scanning Techniques

Over the years, 3D scanning techniques have evolved significantly. While photogrammetry offers many advantages, other techniques like laser scanning and structured light scanning have also played a pivotal role in advancing 3D scanning capabilities. Each method has unique benefits depending on the application and requirements for accuracy, speed, and cost.

2.1.1 Laser Scanning

Laser scanning uses a laser beam to measure the distance between the scanner and the object. By rotating the laser around the object, it captures a point cloud, which represents the object's surface in 3D. Laser scanning is highly accurate, with sub-millimeter precision, making it ideal for applications like industrial design and medical imaging. However, laser scanning systems are expensive and require controlled environments to achieve optimal results [7]. In hand scanning, laser scanning is used to generate highly detailed models, especially for prosthetics design, where precise measurements are required for creating custom-fitted devices [8]. The downside is that laser scanning can struggle with certain surfaces, such as reflective or translucent materials, which may lead to incomplete data capture [9].

2.1.2 Structured Light Scanning

Structured light scanning projects a pattern of light onto the object, and cameras record the deformations of the light pattern to determine the object's geometry. This method is faster than laser scanning and captures a large number of data points in a short time. Structured light scanners are commonly used in medical applications and quality control for industrial products [10]. In the context of hand scanning, structured light scanning provides a balance between speed and accuracy, making it an ideal method for capturing dynamic gestures or modeling hands for ergonomic studies. However, like laser scanning, structured light scanning requires a controlled lighting environment, and its accuracy can be affected by surface properties [9].

2.1.3 Hybrid Techniques

To overcome the limitations of individual methods, hybrid systems that combine photogrammetry with laser scanning or structured light scanning have been developed. These hybrid methods capitalize on the strengths of each technique. For example, combining laser scanning's precision with photogrammetry's affordability provides a more comprehensive solution for applications requiring high accuracy at lower costs [11].

2.2 Advantages of Photogrammetry:

- Non-Invasiveness: Photogrammetry allows for scanning without any direct contact, which is particularly beneficial in medical applications where the subject may be sensitive or fragile [4].
- Affordability: Compared to more expensive methods like laser scanning, photogrammetry requires only a camera or smartphone and software, making it accessible for a wide range of users [2].
- Scalability: Photogrammetry can be applied to objects of varying sizes, from small objects like human hands to larger environments, making it a highly versatile technique [5].
- Detail Capture: Modern photogrammetry, especially when combined with AI algorithms, can capture minute details, including skin texture and fine hand geometry, critical for applications like prosthetics [6].

Recent advances in AI and machine learning have further improved the accuracy of photogrammetry-based models. These algorithms help in point cloud optimization, noise reduction, and feature extraction, making photogrammetry more competitive with traditional 3D scanning methods [12]. Photogrammetry has been applied to various industries, particularly in hand scanning for prosthetics, medical imaging, ergonomics, and human-computer interaction (HCI). Below are some key applications:

- Prosthetics Design: Photogrammetry is used to create highly accurate models of patients' hands, which are essential for designing custom prosthetic devices. These models ensure that the prosthetic fits perfectly, improving both comfort and functionality [8].
- Medical Imaging and Rehabilitation: In conditions such as arthritis or carpal tunnel syndrome, 3D models generated through photogrammetry can be used to track the progress of treatment and rehabilitation. These models help doctors and therapists monitor changes in the hand's structure over time [10].
- Ergonomics: Photogrammetry is used in the design of tools, automotive controls, and consumer products, ensuring that hand-operated devices are ergonomically sound. Detailed 3D hand models allow designers to evaluate how users interact with products, leading to improved designs that prioritize user comfort [11].
- Gesture Recognition in HCI: Photogrammetry-based hand models are integral to the development of gesture recognition systems used in virtual reality (VR) and augmented reality (AR). These systems rely on precise hand models to track movements and interpret gestures, enabling intuitive, touchless control interfaces [12].

Photogrammetry has emerged as one of the most accessible and flexible methods for 3D scanning, especially for hand scanning applications. Its affordability, non-invasive nature, and ability to capture detailed surface geometry make it a popular choice for fields like prosthetics, medical imaging, and ergonomics. While laser scanning and structured light scanning offer higher accuracy in some contexts, photogrammetry remains a competitive alternative, especially when enhanced with AI and machine learning algorithms. The integration of hybrid methods combining photogrammetry with other scanning techniques further expands its potential, making it a key technology for both commercial and research applications.

3 DOCUMENTATION AND DESIGN

3.1 Overview of System Modules

The 3D hand scanner system comprises several key modules:

- **Data Capturing Module** Purpose: Captures high-resolution images from various angles to create a comprehensive scan. Components: Raspberry Pi Zero Wireless: Each unit provides 802.11n wireless LAN and Bluetooth 4.0, enabling seamless connectivity. ArduCam (5MP): High-resolution camera module capturing images at 2592 x 1944 pixels. Strategically positioned around the hand to capture images from multiple angles, ensuring full coverage.

- **Data Transmission Module** Purpose: Manages the network configuration for multiple Raspberry Pis, allowing them to operate as Compound Pi servers for data acquisition and communication. Components: Compound Pi Software: Configured on a single Pi and cloned for others, this software operates over broadcast UDP, enabling local network communication between Pis. Raspberry Pi Zero Wireless (512MB RAM): Acts as a server to listen for and execute network commands. Efficient in LAN-based operations due to its UDP broadcast capability. Setup: Each Pi listens for orders from a network client, executing and sending back results similar to a web server but optimized for LAN. Reference: Compound Pi Documentation

- **Data Processing Module** Purpose: Functions as the control center for the scanning process and post-processes the captured data for 3D model reconstruction. Components: Laptop/Computer: Orchestrates the scanning process and handles photogrammetry for 3D reconstruction. Requires sufficient processing power for complex 3D tasks. Note: This module is adaptable, allowing substitution with any computer capable of handling intensive data processing.

- **Lighting Module** Purpose: Provides consistent illumination, enhancing image quality and enabling quicker image acquisition. Components: LED Strips (12V): Three white LED strips wrapped around the scanner serve as the primary light source. This setup improves accuracy by reducing motion blur with faster shutter speeds.

- **Power Supply Module** Purpose: Ensures stable and continuous power for the entire system, including Raspberry Pis, networking components, and LED lighting. Components: 4S Lithium-Ion Battery or 12V Power Supply: Supplies power to Raspberry Pis and LED strips, enabling reliable operation for all modules.

3.1.1 System Interactions and Use Case Diagram

The following use cases define user interactions with the scanner:

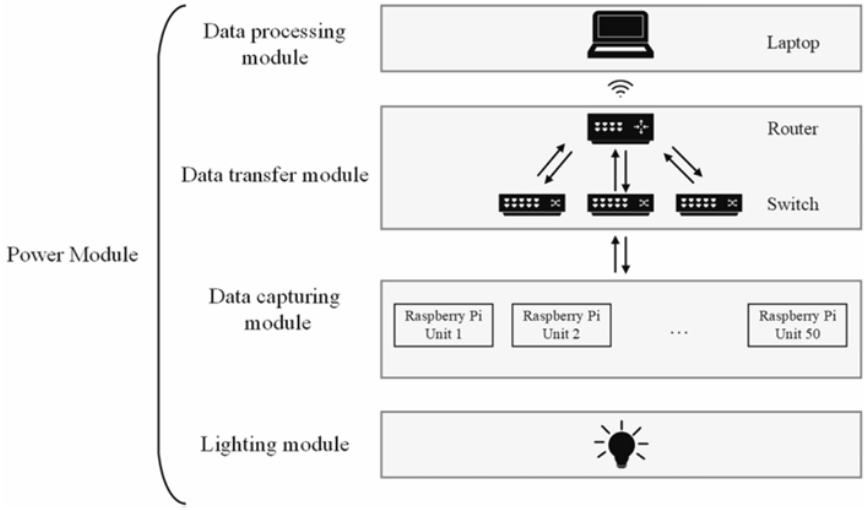

Figure 3.1: Proposed System Design

- **Initiate Scanning**: The user commands the system to begin synchronized image capture.
- **Monitor Progress**: The system provides real-time feedback on scanning progress.
- **3D Reconstruction**: The user initiates processing of the captured images to reconstruct a 3D model.

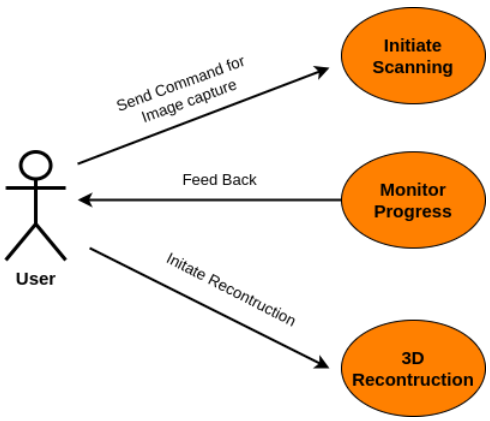

Figure 3.2: User Case Diagram

3.2 Design Considerations

3.2.1 Constraints

- Limited budget
- Fixed camera positions

- Modular assembly
- Lightning

3.2.2 System Environment

The system environment requires a Wi-Fi-enabled setup where Raspberry Pi units connect to a central laptop. This setup provides flexibility in positioning and easy control over the scanning process, ensuring effective wireless communication between all devices.

3.2.3 Design Methodology

The modular design approach allows each component, such as Raspberry Pi units, LED lighting, and power supplies, to be assembled independently and tested in stages. Each module's feedback can lead to iterative improvements in calibration, synchronization, and image quality.

3.3 Data Design

3.3.1 Interaction Workflow

A sequence of commands is issued from the laptop, which initiates synchronization, captures images, collects data, and sends it for processing. Upon completion, users can view the generated 3D model.

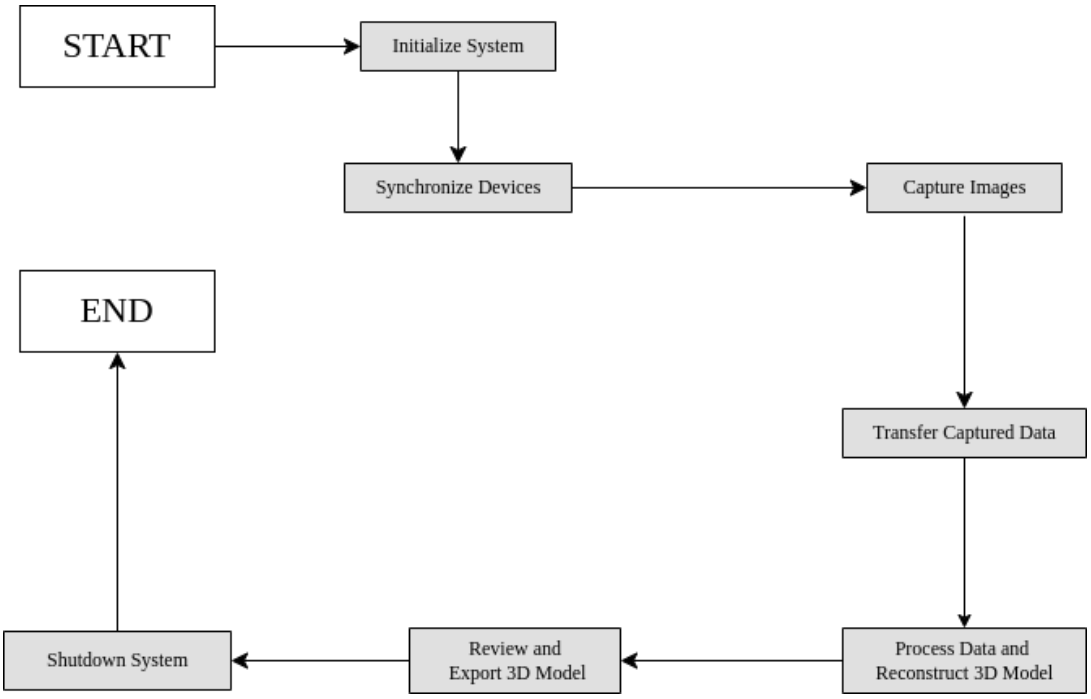

Figure 3.3: Workflow Diagram

3.4 Testing and Security Considerations

3.4.1 Testing

Testing focuses on:

- **Synchronization Accuracy**: Ensuring all Raspberry Pi units capture images within 80 ms.
- **Data Transfer Reliability**: Testing for uninterrupted data transmission.
- **Model Accuracy**: Verifying the 3D reconstruction by comparing scans to a reference model.

3.4.2 Security

To ensure data integrity and protect the system, the Wi-Fi network is encrypted, and access is limited to authorized devices. Commands sent to Raspberry Pi units are also secured to prevent unauthorized access.

3.5 Deployment

3.5.1 System Setup

The system requires Python-based software on each Raspberry Pi to handle image capture and data transmission. A central control system, connected to the Raspberry Pi network, manages synchronization and controls data flow.

3.5.2 Operational Flow

Once deployed, the operational flow includes initiating the system, capturing data, processing images, and reconstructing the 3D model. The system provides feedback at each stage, ensuring smooth operation.

4 Prototypes and Results

4.1 Ni3D Zero

We have successfully built the initial prototype of the scanner system. This prototype includes:

- A single Raspberry Pi Noir V2 camera, chosen for its 48.8° field of view, which is ideal for capturing focused, high-quality images at close distances. This camera is affixed at a height above the rotating platform, enabling it to capture images at each rotational interval.
- A Raspberry Pi 3B module controls the camera and image capture intervals, offering a user-friendly interface for managing the device. The platform is controlled via an Arduino Mega, which drives a 4-wire stepper motor using an L298 motor driver. This setup enables precise rotation and synchronized captures, essential for consistent, repeatable scans.

The platform rotates at 30° increments every 5 seconds, which aligns with the camera's capture timing. This interval ensures that the captured images cover a full 360° view, allowing for detailed photogrammetry processing of the hand model.

4.1.1 Rotating Platform Setup

The rotating platform, controlled by the Arduino Mega, provides a stable and precise rotation for image capture. The use of the L298 motor driver, coupled with a 4-wire stepper motor, enables smooth rotation with precise control. This level of control is crucial, as consistent positioning and timing between the camera and platform rotation are vital for effective photogrammetric reconstruction. Adjustments were made to fine-tune the motor's speed, ensuring each rotation aligns perfectly with the camera's capture interval.

3. System Modules and Interaction

To streamline the scanning process, the project system is divided into specific modules, each responsible for handling different functionalities:

- **Data Capturing Module:** The Raspberry Pi camera captures images at predefined intervals. Every 5 seconds, as the platform rotates 30°, an image is captured. This approach allows for a full set of hand images to be obtained in a short duration.
- **Control and Synchronization Module:** The Arduino Mega, connected to the motor driver and stepper motor, handles platform rotation. The synchronization between the Arduino and Raspberry Pi ensures that image capture occurs at every desired angle, with the Raspberry Pi sending a trigger signal to capture each frame. This approach provides both reliability and flexibility in the scanning process.

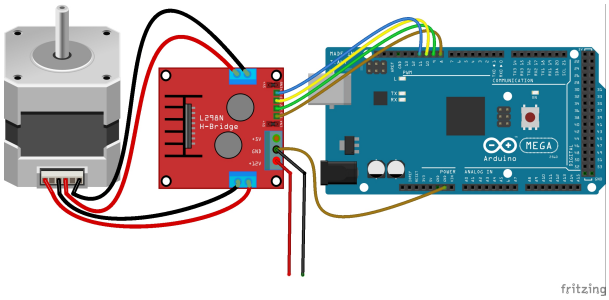

Figure 4.1: Rotating platform setup

- **Lighting Module (Planned):** The current prototype does not include additional lighting, which could affect image quality. To address this, an LED lighting system will be installed around the platform to provide uniform illumination and reduce shadows. This enhancement will improve image quality and accuracy, which are crucial for high-resolution 3D reconstruction.

4.1.2 Challenges and Solutions

This project has faced a few challenges in achieving consistent and synchronized performance from the prototype components. Below are the key challenges and their respective solutions:

- **Synchronization Issues:** Synchronizing the camera with the rotating platform was essential for capturing high-quality images at each angle. Initial tests showed slight timing mismatches, but these were resolved by fine-tuning the Arduino and Raspberry Pi interface codes to improve communication and ensure capture precision.
- **Image Quality under Variable Lighting:** During initial trials, inconsistencies in ambient lighting conditions affected image quality, potentially compromising the accuracy of 3D reconstruction. The planned LED lighting module will provide consistent and uniform light, mitigating this issue. The LEDs will reduce shadows and enhance contrast, enabling the system to capture clearer, more detailed images across all rotational positions.
- **Platform Stability:** To ensure minimal vibration and smooth rotation, we adjusted motor speeds and used cushioning to stabilize the platform, resulting in improved image capture consistency.

Testing has been a critical aspect of the project to ensure the prototype meets expected performance standards: The initial prototype was tested to confirm the reliability of image captures at each 30° rotation interval. This test helped validate the synchronization between the platform rotation and camera capture, ensuring all components operated in harmony.

4.2 Ni3D 1.0

4.2.1 Overview of Ni3D 1.0

The Ni3D 1.0 represents a significant advancement from our initial prototype, Ni3D Zero. In this iteration, we have expanded the data capturing module to include multiple Raspberry Pi Zero units, each equipped with a V2 Camera Module. This configuration allows for simultaneous image capture from multiple angles, greatly improving the efficiency and accuracy of the 3D scanning process.

4.2.2 Hardware Components

The primary hardware components of the Ni3D 1.0 system include:

- **Raspberry Pi Zero**: A compact and cost-effective single-board computer that provides the necessary processing power and connectivity for each camera unit. Its small form factor and low power consumption make it ideal for distributed imaging setups.
- **Raspberry Pi Camera Module V2**: An 8-megapixel camera capable of capturing high-resolution images (3280 x 2464 pixels). The camera's quality is essential for detailed photogrammetric reconstruction.
- **3D Printed Ring Frame**: A custom-designed frame made from PLA material with an inner diameter of 18 cm and an outer diameter of 20 cm. The frame securely holds the Raspberry Pi Zeros and camera modules in precise positions around the scanning area.

4.2.3 System Architecture

The Ni3D 1.0 system architecture is designed for scalability and synchronization. The multiple Raspberry Pi Zero units act as individual imaging nodes, all controlled by a central processing unit (a laptop or server). The central unit issues commands to synchronize the image capture across all nodes.

Data Capture Process

The data capture process involves the following steps:

1. **Initialization**: The central unit initializes communication with all Raspberry Pi Zero nodes over the network.
2. **Synchronization**: A synchronization protocol ensures that all camera modules capture images simultaneously or within an acceptable time window (e.g., within 80 ms).
3. **Image Capture**: Upon receiving the capture command, each camera module captures a high-resolution image of the object from its unique angle.
4. **Data Transfer**: Captured images are transferred back to the central unit for processing.

4.2.4 Software Implementation

The software implementation includes custom scripts running on the Raspberry Pi Zeros and the central unit:

- **Raspberry Pi Software**: A lightweight Python script runs on each Raspberry Pi Zero, listening for commands from the central unit. Upon receiving a capture command, it triggers the camera module and sends the image back to the central unit.
- **Central Control Software**: The central unit runs a control script that manages the synchronization and data collection from all nodes. It also handles the pre-processing of images before 3D reconstruction.
- **Networking**: Communication between the central unit and the Raspberry Pi nodes is handled over a wired Ethernet network, using TCP/IP protocols for reliable data transfer.

4.2.5 Testing and Results

Extensive testing was conducted to validate the performance of Ni3D 1.0:

Synchronization Accuracy

Tests confirmed that the synchronization protocol effectively ensures simultaneous image capture across all nodes. Time discrepancies were measured to be less than 50 ms, which is acceptable for photogrammetry.

Image Quality

The integration of the LED lighting system resulted in high-quality images with consistent lighting conditions. This improvement significantly enhanced the details captured in the images, which is crucial for accurate 3D reconstruction.

3D Reconstruction

Using the captured images, we performed 3D reconstructions using photogrammetry software (e.g., Agisoft Metashape or OpenMVG). The resulting 3D models exhibited high levels of detail and accuracy, validating the effectiveness of the Ni3D 1.0 system.

4.2.6 Challenges and Solutions

During the development and testing of Ni3D 1.0, we encountered several challenges:

- **Network Latency**: Initial tests showed that network latency could affect synchronization. We addressed this by optimizing the network configuration and using wired connections instead of wireless to reduce latency.
- **Power Management**: With multiple Raspberry Pi units and LEDs, power consumption was higher than anticipated. We implemented a centralized power management system with adequate capacity and proper voltage regulation.

- **Data Handling**: Transferring large image files from multiple nodes simultaneously required efficient data handling. We optimized the data transfer protocols and implemented a queuing system to manage bandwidth usage.

4.2.7 Future Enhancements

Building on the success of Ni3D 1.0, future enhancements may include:

- **Increased Number of Cameras**: Adding more camera modules to capture images from more angles, improving the completeness and detail of the 3D models.
- **Automated Calibration**: Implementing software routines for automatic calibration of camera positions and orientations to streamline setup.
- **Real-time Reconstruction**: Exploring methods for real-time or near-real-time 3D reconstruction to provide immediate feedback to users.
- **Wireless Communication**: Investigating the use of high-speed wireless communication protocols to eliminate the need for wired connections while maintaining synchronization.

Figure 4.2: Ni3D 1.0 Prototype Setup

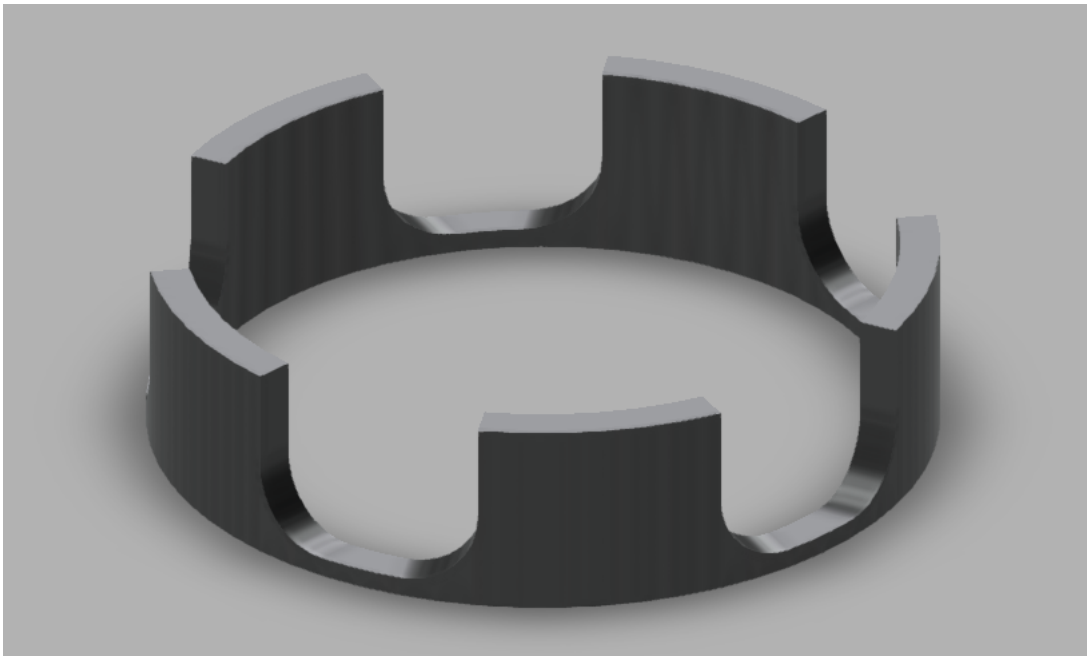

Figure 4.3: 3D Printed Frame for Camera Modules

4.3 Ni3D 2.0

Figure 4.4: Ni3D 2.0 Prototype Setup

4.3.1 Overview of Ni3D 2.0

Building upon the foundational architecture of Ni3D 1.0, Ni3D 2.0 represents a significant leap in reliability, robustness, and system design. One of the major issues encountered in Ni3D 1.0 was data loss during image retrieval , primarily due to simultaneous and uncoordinated transmission attempts from multiple Raspberry Pi nodes. Ni3D 2.0 solves this critical bottleneck through the implementation of a queue-based sequential data transfer system , ensuring each Raspberry Pi sends data to the central server in a strictly controlled, non-overlapping manner.

4.3.2 Key Improvements Over Ni3D 1.0

- **Sequential Data Transfer Using Queues**: To address the image data loss observed in Ni3D 1.0, a queuing mechanism was introduced. In Ni3D 2.0, the central controller maintains a queue and sends explicit retrieval requests to each Raspberry Pi one at a time. Only the node at the front of the queue is allowed to initiate data transfer. Once it completes, the next node is triggered. This serialized communication protocol guarantees zero collision and significantly improves data integrity.
- **Enhanced Server-Client Protocol**: The command-response structure was redesigned to ensure acknowledgements at every step. Each Pi sends status messages like `CAPTURE_COMPLETE`, `SEND_DONE`, or `RESET_COMPLETE` to the server, which in turn triggers the next step, preventing race conditions.
- **Improved Error Handling**: The new system includes checksums and timeout-based fallback routines to detect and handle incomplete transmissions, further boosting reliability.
- **Optimized Socket Management**: Persistent socket pooling and controlled connection lifetimes were implemented to reduce handshake overhead and increase throughput during batch transfers.
- **Dynamic Node Discovery**: Ni3D 2.0 supports automatic discovery of connected Raspberry Pi nodes on the network, simplifying setup and allowing dynamic scalability.

4.3.3 Revised Data Capture and Retrieval Workflow

1. **System Initialization**: The server scans for available Pi nodes and assigns them unique queue positions.
2. **Parallel Image Capture**: All Pi nodes receive a `CAPTURE:N` command and capture images in parallel, storing them locally.
3. **Queued Data Retrieval**: The server begins dequeuing nodes one by one. It sends a `GET` command to the Pi at the front of the queue. This Pi transmits its images back to the server. Once the server confirms reception with an `ACK`, the next Pi in the queue is activated.

4. **Post-transfer Reset**: After all images are retrieved, the server optionally sends a `RESET` command to each Pi to clean their storage, ensuring readiness for the next cycle.

After integrating the queuing-based protocol in Ni3D 2.0, no image loss was observed across extensive test cases involving concurrent captures from 8 Raspberry Pi Zero units. The sequential transfer model maintained the integrity of every JPEG file, and transfer speeds remained within acceptable bounds due to reduced congestion.
Ni3D 2.0 demonstrates how thoughtful system-level coordination can solve deeply rooted data-handling problems in distributed embedded setups. By mitigating the concurrency-induced image loss present in Ni3D 1.0, the system now provides a stable and deterministic platform suitable for high-accuracy 3D photogrammetry applications (More details on the Code can be found in Appendix).

4.4 Image Acquisition and Reconstruction Output

Figure 4.5 displays a set of input images captured using multiple Raspberry Pi camera modules deployed in the Ni3D 2.0 scanning system. Each image represents a unique perspective of the scanned object, acquired simultaneously from different nodes within the distributed camera cluster. The synchronization protocol employed ensures that all cameras trigger image capture nearly simultaneously, minimizing parallax distortion due to object motion. These high-resolution images form the basis of the photogrammetric reconstruction pipeline, providing dense multi-view data coverage of the target geometry.

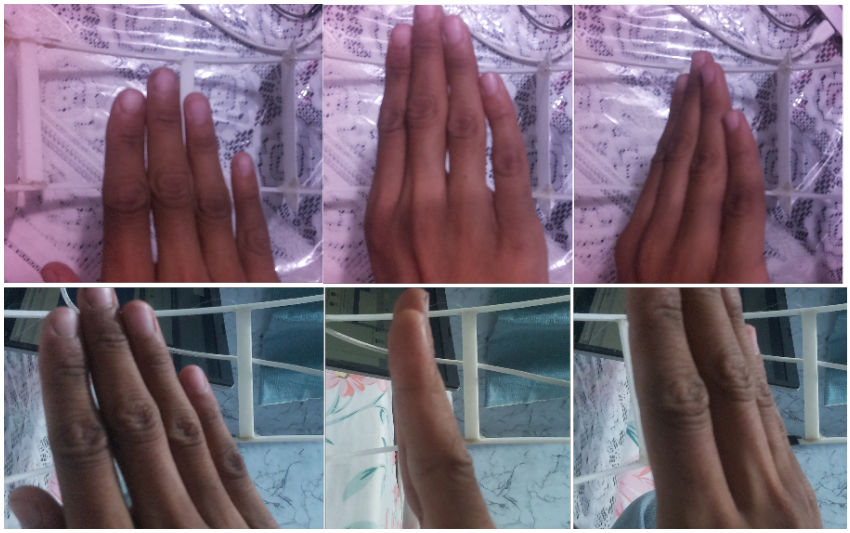

Figure 4.5: Input images captured by Raspberry Pi camera modules during a single scan session using Ni3D 2.0. Each image corresponds to a different viewpoint for multi-view stereo reconstruction.

Figure 4.6 showcases the reconstructed 3D models generated from the above input images. The two images illustrate the same mesh under varying lighting and background rendering conditions—one with a blue background and diffuse lighting, and the other with a white background and directional lighting for contrast enhancement. The reconstructions clearly demonstrate the spatial fidelity and surface detail retained through the

Ni3D 2.0 pipeline. Minor surface noise and mesh artifacts, visible in shadowed or textureless regions, suggest the need for further fine-tuning in both the calibration phase and the image acquisition parameters.

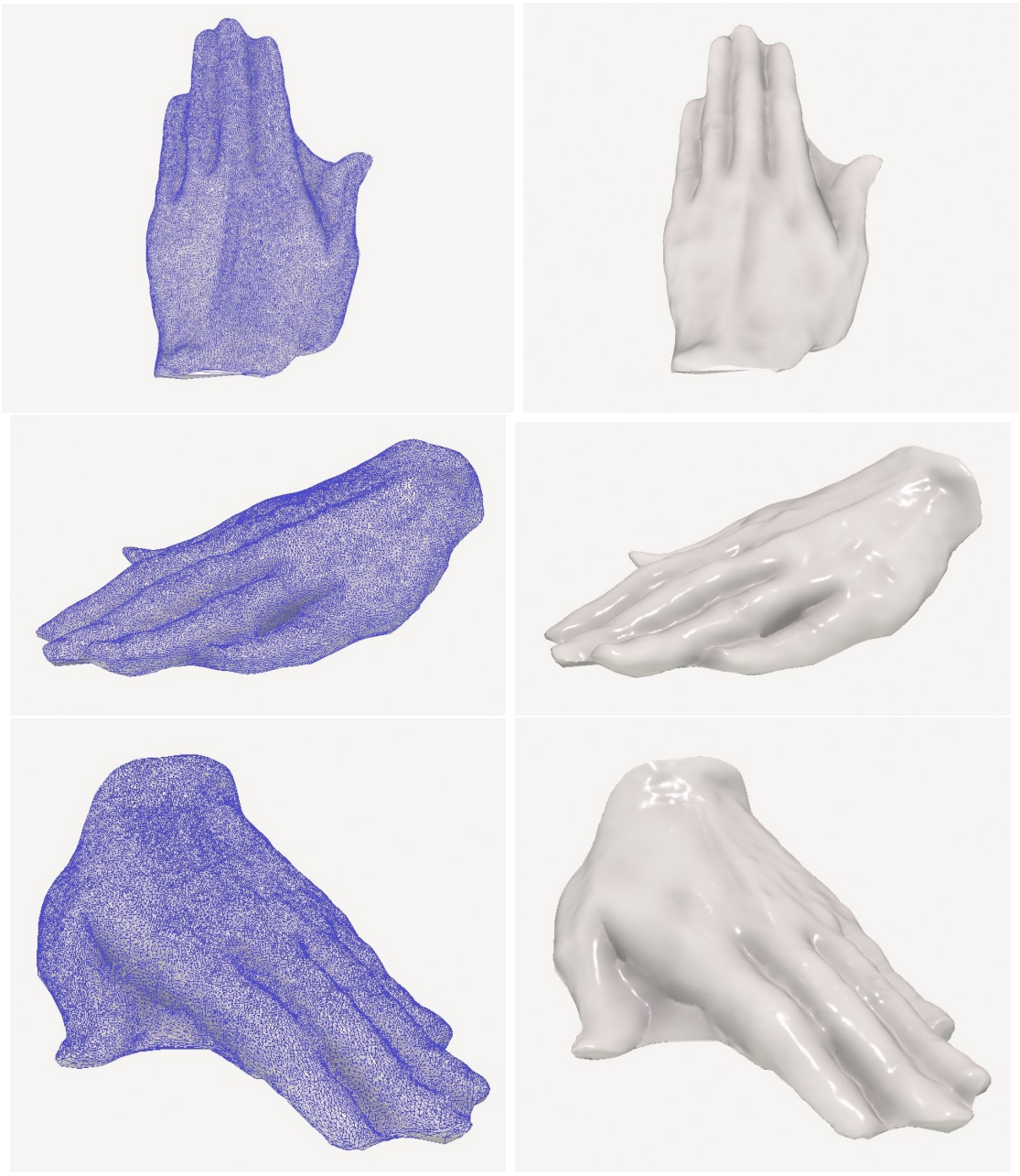

Figure 4.6: 3D reconstruction outputs generated from the Ni3D 2.0 image pipeline.

These results validate the effectiveness of the Ni3D 2.0 system in producing high-quality 3D reconstructions using an embedded, low-cost multi-camera architecture. Further improvements in calibration accuracy, lighting uniformity, and viewpoint density are expected to enhance reconstruction smoothness and reduce surface artifacts in future iterations.

5 Conclusion and Future Works

The journey of developing a cost-effective and robust 3D scanning platform began with the creation of the Ni3D Zero prototype. This initial system laid the groundwork for exploring photogrammetric 3D reconstruction using minimal hardware. Through this design, we achieved synchronized captures at 30° rotational intervals, enabling full 360° data acquisition. However, several challenges emerged, such as unstable lighting, motor-induced vibrations, and synchronization mismatches, which were addressed through iterative improvements and modular redesign. The development of Ni3D 2.0 marks a substantial advancement over the Ni3D 1.0 system in both architectural robustness and functional reliability. While Ni3D 1.0 successfully demonstrated a low-cost, scalable 3D scanning platform using Raspberry Pi Zeros and high-resolution camera modules, it faced significant challenges related to data integrity, particularly during image retrieval from multiple distributed nodes. In Ni3D 1.0, the simultaneous transmission of images from multiple Raspberry Pi units led to data collisions and partial file transfers, thereby degrading the quality of the 3D reconstructions. Ni3D 2.0 addresses this critical issue through a queue-based sequential communication protocol , ensuring that only one node transmits at a time. This serialized approach has completely eliminated image loss , leading to consistent and reliable data acquisition across all connected nodes. Additionally, socket communication was optimized, and a modular control interface was implemented for command-response synchronization. These architectural changes ensure that the system scales well with increasing node count while maintaining low latency and high reliability. However, current reconstructions show the presence of extraneous grid artifacts —likely a result of internal calibration errors, environmental inconsistency, or minor misalignments in the optical paths. This highlights the need for further fine-tuning of camera calibration parameters and reconstruction algorithms.

5.1 Future Works

5.1.1 Integration of Advanced Lighting Module

Controlled illumination is vital for reducing shadow artifacts and enhancing surface feature capture. Future versions will integrate a tunable LED lighting system with programmable intensity and spectrum adjustments. This will ensure uniform lighting regardless of object texture or ambient light conditions.

5.1.2 Enhanced 3D Reconstruction Algorithms

We aim to incorporate advanced photogrammetric techniques, including structure-from-motion (SfM) with AI-based depth inference. These upgrades will improve geometric accuracy and reduce noise, especially in occluded or reflective regions. Integration of open-source tools like COLMAP or OpenMVG with post-processing pipelines tailored to embedded systems is under exploration.

5.1.3 Scaling the Camera Array

Increasing the number of cameras in the Ni3D ring setup is a key focus for improving 3D model completeness. This scaling will help mitigate occlusion zones and boost depth map fidelity. Alongside this, geometric optimization of camera positions and inter-camera angle will be conducted for balanced spatial coverage.

5.1.4 Wireless Communication Over Dynamic IP Networks

To increase deployment flexibility, especially in dynamic or mobile setups, we aim to transition from wired Ethernet to secure wireless protocols . Recognizing the unreliability of standard IP-based methods in dynamic networks, we plan to implement MAC-based device binding for secure identification and command routing. Protocols optimized for embedded communication such as MQTT or CoAP will be evaluated, ensuring robust low-latency transmission under fluctuating network conditions.

5.1.5 Cloud-Accelerated Processing and Secure Offloading

For large-scale reconstructions, the computational demand can exceed the capabilities of local systems. Integration with cloud processing pipelines , using containerized reconstruction environments (e.g., Dockerized COLMAP), will enable near-real-time reconstruction. We will ensure secure data transmission and storage using encrypted protocols like TLS over MQTT.

5.1.6 Real-world Scenario Testing and Adaptability

Ni3D 2.0 will be tested across a range of application domains including industrial inspection , archaeological preservation , and biometric scanning . Field deployment will uncover application-specific constraints and allow us to evolve the system towards a more user-configurable and adaptive platform.

5.1.7 Automated Calibration and Artifact Reduction

To address the observed grid patterns in 3D reconstructions, we will develop automated calibration routines using fiducial markers and structured light patterns. These tools will optimize intrinsic and extrinsic parameters dynamically, ensuring minimal geometric distortion in the output mesh.

5.2 Final Remarks

Ni3D 2.0 not only resolves the critical issue of image loss faced in its predecessor but also lays the foundation for a scalable, secure, and high-fidelity 3D scanning platform . With sequential data queuing, planned wireless integration, and algorithmic improvements, it stands poised for broader adoption in low-cost, high-impact 3D vision applications. Continued development will focus on real-time reconstruction, edge deployment, and deeper integration with AI-based vision systems, making Ni3D a powerful tool in embedded 3D imaging.

6 References

1. Y. Liu et al, A Novel Cloud-Based Framework for the Elderly Healthcare Services Using Digital Twin, IEEE Access 7 (2019) 49088–49101.

2. G. Elkoura, K. Singh, Handrix: animating the human hand, in: ACM Eurographics/SIGGRAPH Symp. Comput. Animat., 2003, pp. 110–119.

3. H. Endo, K. Kawahara, Gender differences in hand stability of normal young people assessed at low force levels, Ergonomics 54 (3) (2011) 273–281.

4. R. Garsthagen, "An Open Source, Low-Cost, Multi Camera Full-Body 3D Scanner," in 5th International Conference on 3D Body Scanning Technologies, pp. 174–183.

5. 3dMD, "3dMD hand system," 2020. [Online]. Available: https://3dmd.com/products/!/hand.

6. T.C. ten Harkel, C.M. Speksnijder, F. van der Heijden, C.H.G. Beurskens, K.J.A.O. Ingels, T.J.J. Maal, Depth accuracy of the RealSense F200: Low-cost4D facial imaging, Sci. Rep. 7 (1) (2017) 1–8.

7. Microsoft., "Azure Kinect," 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/kinect-dk/hardware-specification.

8. A. Leipner, R. Baumeister, M.J. Thali, M. Braun, E. Dobler, L.C. Ebert, Multi-camera system for 3D forensic documentation, Forensic Sci. Int. 261 (2016) 123–128.

9. A. Yu, K.L. Yick, S.P. Ng, J. Yip, 2D and 3D anatomical analyses of hand dimensions for custom-made gloves, Appl. Ergon. 44 (3) (2013) 381–392.

10. G. Harih, B. Dolšak, Tool-handle design based on a digital human hand model, Int. J. Ind. Ergon. 43 (4) (2013) 288–295.

11. Y. Yang, H. Zhou, Y.u. Song, P. Vink, Identify dominant dimensions of 3D hand shapes using statistical shape model and deep neural network Applied Ergonomics, Appl. Ergon. 96 (2021) 103462, https://doi.org/10.1016/j.apergo.2021.103462.

12. X. Mouy et al, FishCam: A low-cost open source autonomous camera for aquatic research, HardwareX 8 (00110) (2020).

13. Y. Yang, W.S. Elkhuizen, T. Hou, T.T.W. Essers, Y. Song, Optimal Camera Configuration for 3D Scanning of Human Hand, International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, 2019, p. V001T02A047.

14. Raspberry Pi., "Raspberry Pi OS," 2020. [Online]. Available: https://www.raspberrypi.org/softw

15. Python, "Python For Beginners," 2020. [Online]. Available: https://www.python.org/about/getti

16. Agisfot, 3D model reconstruction tutorial [Online]. Available 2020 https://agisoft.freshdesk.com/s

17. Caroline A. Grant, Melissa Johnston, Clayton J. Adam, J. Paige Little, Accuracy of 3D surface scanners for clinical torso and spinal deformity assessment, Med. Eng. Phys. 63 (2019) 63–71.

18. Capturingreality," 2021. [Online]. Available: https://www.capturingreality.com/.

19. J.L. Schönberger, E. Zheng, J.-M. Frahm, M. Pollefeys, Pixelwise view selection for unstructured multi-view stereo, European Conference on Computer Vision (2016) 501–518.

20. J.L. Schonberger, J.M. Frahm, Structure-from-Motion Revisited, Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. vol. 2016-Decem (2016) 4104–4113.

21. Siti Hana Nasir, Olga Troynikov, Influence of hand movement on skin deformation: A therapeutic glove design perspective, Appl. Ergon. 60 (2017) 154 162.

22. R.T. Yunardi, A. Imandiri, Design of the 3D Surface Scanning System for Human Wrist Contour Using Laser Line Imaging, in: Proc.- 2018 5th Int. Conf. Inf. Technol. Comput. Electr. Eng. ICITACEE, 2018, pp. 19–23.

23. Miyeon Lee, Dong Il Yoo, Sungmin Kim, Development of low cost three-dimensional body scanner using depth perception camera, Int. J. Cloth. Sci. Technol. 29 (6) (2017) 857–867.

24. J. Straub, S. Kerlin, Development of a Large, Low-Cost, Instant 3D Scanner, Technologies 2 (2) (2014) 76–95.

25. Y. Yang, T. Yuan, T. Huysmans, W.S. Elkhuizen, F. Tajdari, Y. Song, Posture-invariant 3D Human Hand Statistical Shape Model, ASME J. Comput. Inf. Sci. Eng. (2020) 1–31

7 Appendix

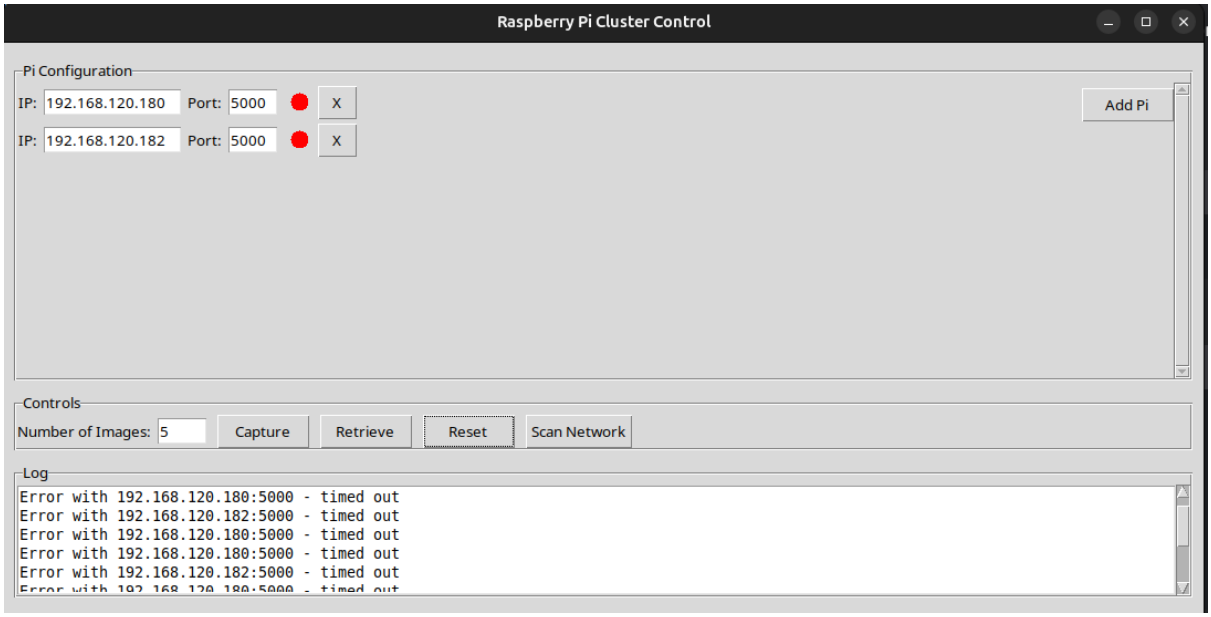

Figure 7.1: GUI

System Description

This section describes the functionality of a Python-based GUI application designed for managing a distributed Raspberry Pi camera cluster. Built using the Tkinter library, the interface provides centralized controls for capturing, retrieving, and resetting image data from multiple Raspberry Pi devices configured as remote nodes. When launched, the GUI presents a window titled **"Raspberry Pi Cluster Control"** (Figure 7.1). This window is divided into three primary sections: **Pi Configuration**, **Controls**, and **Log**. At its core, this application allows users to dynamically add, remove, and manage multiple Raspberry Pis, each specified by their IP address and port number.

Pi Configuration Section

This section features a scrollable list where the user can enter:

- **IP address** of the Raspberry Pi.
- **Port number**, with a default value of 5000.
- A colored circular status indicator (green for connected, red for disconnected).

Each Pi entry also includes a small **"X"** button, which can be used to remove the specific Pi from the list. A button labeled **"Add Pi"** adds a new row to this list, allowing the user to manage multiple Pis in parallel.

Controls Section

This section allows the user to issue commands to the connected Raspberry Pis. It contains:

- A numeric entry box for specifying the number of images to capture (default: 5).
- **"Capture"** button: sends a `CAPTURE:<num>` command to all listed Pis and the Pis start capturing the images simultaneously.
- **"Retrieve"** button: sends a `GET` command and downloads the captured images.

  The GET command triggers image reception over TCP sockets. Images are saved under a directory structure like `./server_images/<Pi_IP>/`. Each image is preceded by a JSON header specifying the filename and size. Transmission concludes with an `END` message.
- **"Reset"** button: sends a `RESET` command to clear the state/data on Pis.
- **"Scan Network"** button: executes an `arp-scan` on the local network and lists active devices.

  When a network scan is completed, a new window opens listing all detected IPs. The user may select one and click **"Use Selected IP"** to populate a new Pi entry automatically.

Each command (CAPTURE, GET, RESET) is sent using a separate thread for each Pi node to ensure GUI responsiveness. Socket communication includes connection timeout handling, real-time status updates (red/green indicators), and error feedback.

Log Section

A scrollable log pane displays runtime messages such as:

- Command dispatch confirmations
- Transfer status for image retrieval
- Socket communication errors

Logs are appended in chronological order and auto-scroll to always display the latest message.

Raspberry Pi Client Code

When the Raspberry Pi boots and a Python script is executed, it opens a TCP socket on all interfaces (`0.0.0.0`) at port `5000`, marking itself as ready to receive incoming connections. It waits patiently in a loop, listening for the server (i.e., the GUI controller on the main machine) to initiate a connection.

Once a connection is established, the Pi logs the IP of the connecting server and delegates the processing to the `handle_command()` function. This function acts as the command dispatcher—it waits to receive a text-based instruction from the server. Each command is expected to be a single keyword or a keyword with a parameter, like `CAPTURE:5`.

CAPTURE Command: Image Acquisition

When the server sends a command like `CAPTURE:5`, it means "capture 5 images." The Pi immediately invokes the `capture_images(num_images)` function. Inside this function, it first ensures that the image storage directory (`./pi_images`) exists. Then, for each image to be captured, it constructs a unique filename using a high-resolution timestamp to prevent filename collisions. It calls the `libcamera-jpeg` utility to capture a full-resolution (1920×1080) image and stores it in the specified folder. After each capture, it waits for one second (to avoid sensor lag or overlapping exposures).

Once all requested images are captured, the Pi sends a short confirmation message—`CAPTURE_COMPLETE`—back to the server over the same TCP connection.

GET Command: Image Transmission

If the server sends the `GET` command, the Pi responds by transmitting all the `.jpg` files currently stored in the `./pi_images` folder. The process is handled by `send_images(conn)`. For every image file, it first reads the file size, constructs a header of the form `filename:size`, and sends this metadata to the server. Then, it opens the image file in binary mode and transmits it chunk-by-chunk using a buffer of 4096 bytes until the entire file has been sent. Once all images are transferred, the Pi sends an `END` message, indicating that no more data remains to be sent.

RESET Command: Image Cleanup

If the `RESET` command is issued by the server, the Pi interprets this as an instruction to delete all previously captured images. The `reset_storage()` function is triggered, which iterates through the `./pi_images` directory and removes every `.jpg` file. Upon completion, the Pi sends back a `RESET_COMPLETE` message to the server to confirm successful cleanup.

UNKNOWN Command: Fallback

In case an unrecognized command is received, the Pi simply replies with `UNKNOWN_COMMAND`, serving as a failsafe mechanism to notify the server that the request was malformed or unsupported.

Connection Lifecycle

After each command is handled and the appropriate response sent, the TCP connection is closed, and the Pi returns to its listening state, ready for the next request. This design ensures that each operation—whether it's capturing, transmitting, or resetting—is handled in a self-contained and stateless manner, with full feedback loops for server confirmation.

8 Appendix-II

Server Control Code base

```
import tkinter as tk
from tkinter import ttk, scrolledtext
import socket
import threading
import os
import json

class PiClusterGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Raspberry Pi Cluster Control")
        self.server_image_base = './server_images'
        os.makedirs(self.server_image_base, exist_ok=True)

        # GUI Configuration
        self.pi_entries = []
        self.create_widgets()

    def create_widgets(self):
        main_frame = ttk.Frame(self.root, padding="10")
        main_frame.pack(fill=tk.BOTH, expand=True)

        # Pi Configuration Section
        config_frame = ttk.LabelFrame(main_frame, text="Pi
            Configuration")
        config_frame.pack(fill=tk.X, pady=5)

        # Scrollable Pi List
        self.canvas = tk.Canvas(config_frame)
        scrollbar = ttk.Scrollbar(config_frame, orient=tk.
            VERTICAL, command=self.canvas.yview)
        self.scrollable_frame = ttk.Frame(self.canvas)

        self.scrollable_frame.bind("<Configure>", lambda e: self.
            canvas.configure(scrollregion=self.canvas.bbox("all"))
            )
        self.canvas.create_window((0, 0), window=self.
            scrollable_frame, anchor="nw")
        self.canvas.configure(yscrollcommand=scrollbar.set)

        self.canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
```

```
        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

        add_btn = ttk.Button(config_frame, text="Add Pi", command
            =self.add_pi)
        add_btn.pack(pady=5)

        # Control Section
        control_frame = ttk.LabelFrame(main_frame, text="Controls
            ")
        control_frame.pack(fill=tk.X, pady=5)

        ttk.Label(control_frame, text="Number of Images:").pack(
            side=tk.LEFT)
        self.num_images = ttk.Entry(control_frame, width=5)
        self.num_images.insert(0, "5")
        self.num_images.pack(side=tk.LEFT, padx=5)

        ttk.Button(control_frame, text="Capture", command=self.
            capture_images).pack(side=tk.LEFT, padx=5)
        ttk.Button(control_frame, text="Retrieve", command=self.
            get_images).pack(side=tk.LEFT, padx=5)
        ttk.Button(control_frame, text="Reset", command=self.
            reset_pis).pack(side=tk.LEFT, padx=5)

        # Log Section
        log_frame = ttk.LabelFrame(main_frame, text="Log")
        log_frame.pack(fill=tk.BOTH, expand=True, pady=5)

        self.log = scrolledtext.ScrolledText(log_frame, state='
            disabled', height=10)
        self.log.pack(fill=tk.BOTH, expand=True)

        self.add_pi()

    def add_pi(self):
        frame = ttk.Frame(self.scrollable_frame)
        frame.pack(fill=tk.X, pady=2)

        ttk.Label(frame, text="IP:").pack(side=tk.LEFT)
        ip_entry = ttk.Entry(frame, width=15)
        ip_entry.pack(side=tk.LEFT, padx=5)

        ttk.Label(frame, text="Port:").pack(side=tk.LEFT)
        port_entry = ttk.Entry(frame, width=5)
        port_entry.insert(0, "5000")
        port_entry.pack(side=tk.LEFT, padx=5)

        status = tk.Canvas(frame, width=20, height=20)
        self.draw_status(status, 'red')
        status.pack(side=tk.LEFT, padx=5)
```

```
        ttk.Button(frame, text="X", width=3,
                   command=lambda: self.remove_pi(frame)).pack(side
                      =tk.LEFT)

        self.pi_entries.append({
            'frame': frame,
            'ip': ip_entry,
            'port': port_entry,
            'status': status
        })

    def remove_pi(self, frame):
        for entry in self.pi_entries:
            if entry['frame'] == frame:
                self.pi_entries.remove(entry)
                frame.destroy()
                break

    def draw_status(self, canvas, color):
        canvas.delete("all")
        canvas.create_oval(2, 2, 18, 18, fill=color, outline='')

    def log_message(self, message):
        self.log.configure(state='normal')
        self.log.insert(tk.END, message + "\n")
        self.log.configure(state='disabled')
        self.log.see(tk.END)

    def capture_images(self):
        try:
            num = int(self.num_images.get())
        except ValueError:
            self.log_message("Invalid number of images")
            return

        for entry in self.pi_entries:
            threading.Thread(target=self.send_command, args=(
                entry['ip'].get(),
                int(entry['port'].get()),
                'CAPTURE',
                num,
                entry['status']
            )).start()

    def get_images(self):
        for entry in self.pi_entries:
            threading.Thread(target=self.send_command, args=(
                entry['ip'].get(),
                int(entry['port'].get()),
                'GET',
                None,
```

```
                entry['status']
            )).start()

    def reset_pis(self):
        for entry in self.pi_entries:
            threading.Thread(target=self.send_command, args=(
                entry['ip'].get(),
                int(entry['port'].get()),
                'RESET',
                None,
                entry['status']
            )).start()

    def send_command(self, ip, port, command, arg, status_canvas)
        :
        try:
            with socket.socket(socket.AF_INET, socket.SOCK_STREAM
                ) as s:
                s.settimeout(5)
                s.connect((ip, port))
                self.update_status(status_canvas, 'green')

                if command == 'CAPTURE':
                    message = f"CAPTURE:{arg}"
                else:
                    message = command

                s.sendall(message.encode())
                self.log_message(f"Sent {command} to {ip}:{port}"
                    )

                if command == 'GET':
                    self.receive_images(s, ip)

        except Exception as e:
            self.log_message(f"Error with {ip}:{port} - {str(e)}"
                )
            self.update_status(status_canvas, 'red')

    def receive_images(self, sock, pi_name):
        save_dir = os.path.join(self.server_image_base, pi_name)
        os.makedirs(save_dir, exist_ok=True)

        try:
            while True:
                header_len = int.from_bytes(sock.recv(4),
                    byteorder='big')
                header_data = sock.recv(header_len).decode()

                if header_data == "END":
                    break
```

```
                header = json.loads(header_data)
                filename = header['name']
                filesize = header['size']

                with open(os.path.join(save_dir, filename), 'wb')
                    as f:
                    remaining = filesize
                    while remaining > 0:
                        chunk = sock.recv(min(4096, remaining))
                        f.write(chunk)
                        remaining -= len(chunk)

                self.log_message(f"Received {filename} from {
                   pi_name}")

        except Exception as e:
            self.log_message(f"Transfer error with {pi_name}: {
               str(e)}")

    def update_status(self, canvas, color):
        self.root.after(0, self.draw_status, canvas, color)

if __name__ == '__main__':
    root = tk.Tk()
    app = PiClusterGUI(root)
    root.mainloop()
```

Raspberry Pi Client Code

```
    import socket
import os
import time
from datetime import datetime
import subprocess  # For camera capture - modify as needed

# Configuration
HOST = '0.0.0.0'
PORT = 5000
IMAGE_DIR = './pi_images'
NUM_CAMERAS = 1  # Change if using multiple cameras

def capture_images(num_images):
    os.makedirs(IMAGE_DIR, exist_ok=True)

    for i in range(num_images):
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S_%f")
        filename = os.path.join(IMAGE_DIR, f'img_{timestamp}_{i}.
           jpg')
```

```
        # Replace with your actual capture command
        # Example using libcamera-jpeg:
        subprocess.run([
            'libcamera-jpeg',
            '-o', filename,
            '--width', '1920',
            '--height', '1080',
            '--nopreview'
        ])

        time.sleep(1)  # Adjust delay between captures as needed

def send_images(conn):
    images = [f for f in os.listdir(IMAGE_DIR) if f.endswith('.
       jpg')]

    for img in images:
        path = os.path.join(IMAGE_DIR, img)
        filesize = os.path.getsize(path)

        # Send header with filename and size
        conn.sendall(f"{img}:{filesize}".encode())

        # Send image data
        with open(path, 'rb') as f:
            while True:
                bytes_read = f.read(4096)
                if not bytes_read:
                    break
                conn.sendall(bytes_read)

    conn.sendall("END".encode())

def reset_storage():
    for f in os.listdir(IMAGE_DIR):
        if f.endswith('.jpg'):
            os.remove(os.path.join(IMAGE_DIR, f))

def handle_command(conn):
    data = conn.recv(1024).decode()

    if data.startswith('CAPTURE'):
        _, num_images = data.split(':')
        capture_images(int(num_images))
        conn.sendall("CAPTURE_COMPLETE".encode())
    elif data == 'GET':
        send_images(conn)
    elif data == 'RESET':
        reset_storage()
        conn.sendall("RESET_COMPLETE".encode())
```

```
    else:
        conn.sendall("UNKNOWN_COMMAND".encode())

def main():
    os.makedirs(IMAGE_DIR, exist_ok=True)

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
        s.listen()
        print(f"Pi client listening on {HOST}:{PORT}")

        while True:
            conn, addr = s.accept()
            print(f"Connection from {addr}")
            handle_command(conn)
            conn.close()

if __name__ == '__main__':
    main()
```