GERMAN - MONGOLIAN INSTITUTE
FOR RESOURCES AND TECHNOLOGY

**The present work was submitted to**

**the German-Mongolian Institute for Resource and Technology**

# Coordinate measuring robotic arm

**Bachelor Thesis**

by

**Amarjargal Sodbayr**

Study Program: Mechatronic engineering

Student ID: B2200017

Supervisor 1 / Examiner 1 **Prof. Young Suk Kim**

Supervisor 2 / Examiner 2 **Prof. Sungchil Lee**

Ulaanbaatar/Nalaikh,

1st of May, 2026

## Statutory Declaration

Sodbayr, Amarjargal B2200017

Last Name, First Name Student ID Number

I hereby affirm in lieu of an oath that I have independently prepared the submitted bachelor thesis entitled:

Design and Development of a Coordinate Measuring Robotic Arm

I confirm that I have not used any sources or materials other than those explicitly stated in the references. All ideas, data, and results that are not originally my own have been properly cited.

In case that the thesis is submitted in both written and electronic form, I declare that both versions are completely identical.

Furthermore, I declare that this work has not been submitted, either in whole or in part, to any other examination authority.

Ulaanbaatar/Nalaikh, 5/1/ 2026



Signature

# Acknowledgement

I would like to begin by expressing my sincere gratitude to the German-Mongolian Institute for Resources and Technology for providing the academic environment and opportunities that have shaped my personal and professional growth over the past five years. The knowledge and practical skills I acquired during my time here have been instrumental in the completion of this thesis.

I am deeply indebted to my supervisors, Prof. Young Suk Kim and Prof. Sungchil Lee, for their invaluable guidance, patience, and support. Their insightful feedback and dedication were essential in helping me navigate this research and refine its quality.

I would also like to thank the faculty and staff of the Mechatronic Engineering program for establishing a strong academic foundation and for their continuous support throughout my studies.

Furthermore, I extend my appreciation to J. Buyandelger and M. Enkhtur for their encouragement and the many helpful discussions that sustained me throughout this journey.

Finally, I am profoundly grateful to my family for their unwavering love, patience, and belief in me. This achievement would not have been possible without them.

## Abstract

This thesis presents the design, development, and experimental validation of a coordinate measuring arm intended for educational and small-scale engineering applications. Conventional coordinate measuring machines (CMMs) are expensive and unsuitable for resource-limited environments; therefore, this work proposes an accessible alternative based on three-joint articulated arm mechanics, absolute rotary encoders, and RS485 Modbus RTU communication. The mechanical structure was designed using Autodesk Inventor and fabricated through fused deposition modelling (FDM) 3D printing. A Python-based software application was developed to acquire encoder data, compute end-effector coordinates using planar forward kinematics, and reconstruct basic geometric shapes including lines, triangles, and circles. A graphical user interface provides real-time visualization of the arm configuration and measurement results. Calibration methods including offset correction and global scaling were implemented to reduce systematic errors. Experimental validation demonstrated that the system achieves line measurement accuracy within ±2%, circle measurement error below 3.3%, and reasonable geometric fidelity for triangular shapes. These results confirm the feasibility of developing functional coordinate measurement capability using low-cost, readily available components.

# Table of figures

Figure 1 Encoder 14
Figure 2 arm tip 23
Figure 3 lower part 24
Figure 4 Hinge 25
Figure 5 lower part hinge 26
Figure 6 Arm link 26
Figure 7 3D printed components simplified 28
Figure 8 Assembled coordinate measuring arm 29
Figure 9 Assembled coordinate measuring arm 29
Figure 10 Breadboard wiring of encoders and RS485 interface 31
Figure 11 3D visual 39
Figure 12 User interface 40
Figure 13 GUI interface showing line measurement between two captured points 43
Figure 14 Triangle measurement displayed in the application 45
Figure 15 Circle measurement and fitted radius displayed in the application 46
Figure 16 3D visualization of the measuring arm showing joints and tip position 47

# Contents

Acknowledgement..... 2
Abstract..... 2
CHAPTER 1: INTRODUCTION..... 4
1.1 Background of the Study..... 4
1.2 Problem Statement..... 5
1.3 Objectives of the Study..... 5
1.4 Scope and Limitations..... 6
1.5 Structure of the Thesis..... 6
CHAPTER 2: Literature review..... 7
2.1 Introduction..... 7
2.2 Coordinate Measuring Machines (CMMs)..... 7

2.2.1 Overview of CMM Systems....7
2.2.2 Types of CMMs....8
2.3 Articulated Measuring Arms....8
2.3.1 Structure and Working Principle....8
2.3.2 Kinematic Chain Concept....9
2.3.3 Advantages....9
2.3.4 Limitations....9
2.3.5 Applications....9
2.4 Rotary Encoders....10
2.4.1 Principle of Operation....10
2.4.2 Types of Encoders....10
2.4.3 Resolution and Precision....10
2.4.4 Error Sources in Encoders....10
2.5 RS485 Communication System....11
2.5.1 Overview....11
2.5.2 Communication Characteristics....11
2.5.3 Multi-Drop Configuration....11
2.6 Modbus RTU Protocol....11
2.6.1 Overview....11
2.6.2 Communication Structure....12
2.6.3 Communication Process....12
2.6.4 Advantages of Modbus RTU....12
2.6.5 Application in This Study....13
2.7 Kinematic Modeling....13
2.7.1 Introduction to Kinematics....13
2.7.2 Forward Kinematics....13
2.7.4 Error Propagation....13
2.7.5 Importance in Measurement Systems....14
2.8 Geometric Reconstruction....14
2.8.1 Point-Based Measurement Concept....14
2.8.2 Shape Definition Using Points....14
2.8.3 Distance Measurement....14
2.8.5 Circle Fitting....15
2.8.6 Importance....15
2.9 Software Systems for Measurement....15

2.9.1 Role of Software....15
2.9.2 Functional Architecture....15
2.9.3 Python in Engineering Applications....15
2.9.4 Libraries and Tools....15
2.9.5 Benefits of Software Integration....16
2.10 Calibration Techniques....16
2.10.1 Importance of Calibration....16
2.10.2 Types of Calibration....16
2.10.3 Error Sources....16
2.10.4 Calibration Impact....17
2.11 Summary....17
CHAPTER 3: METHODOLOGY....17
3.1 Introduction....17
3.2 Design Approach and Development Strategy....18
3.3 Mechanical Design and Fabrication....18
3.3.1 CAD Design Using Autodesk Inventor....18
3.3.2 Link Design and Dimensions....23
3.3.3 3D Printing and Fabrication....23
3.3.4 Assembly of the Mechanical System....24
3.3.5 Mechanical Error Considerations....25
3.4 Electronic System Integration....26
3.4.1 Rotary Encoder Implementation....26
3.4.2 Power Supply Configuration....26
3.4.3 Breadboard-Based Wiring Setup....27
3.4.4 RS485 Communication Interface....27
3.4.5 Communication Configuration....28
3.4.6 System Reliability Considerations....28
3.5 Software Development....28
3.5.1 Configuration Module (config.py)....29
3.5.2 Console-Based Testing (main.py)....30
3.5.3 Modbus Communication Layer (modbus_reader.py)....30
3.5.4 Graphical User Interface (gui_app.py)....31
3.5.5 Summary....35
3.6 Calibration and Accuracy Improvement....35
3.7 System Workflow....36

3.8 Summary..... 36
CHAPTER 4: RESULTS AND DISCUSSION..... 37
4.1 Introduction..... 37
4.2 Line Measurement Results..... 37
4.3 Triangle Measurement Results..... 38
4.4 Circle Measurement Results..... 39
4.5 Visualization Results..... 40
4.6 Overall System Performance..... 40
4.7 Error Analysis..... 40
4.8 Summary..... 41
CHAPTER 5: CONCLUSION AND RECOMMENDATIONS..... 41
5.1 Conclusion..... 41
5.2 Recommendations for Future Work..... 42
REFERENCES..... 44

## CHAPTER 1: INTRODUCTION

# 1.1 Background of the Study

Coordinate Measuring Machines (CMMs) are widely used in manufacturing and engineering industries for precise dimensional inspection and quality control. These systems enable accurate measurement of physical geometries by capturing spatial coordinates of objects and comparing them with design specifications. Traditional CMM systems, however, are often expensive, bulky, and require controlled environments, making them less accessible for educational purposes and small-scale applications.

With the advancement of embedded systems, sensors, and rapid prototyping technologies such as 3D printing, it has become feasible to develop low-cost alternatives to conventional measuring systems. In particular, articulated measuring arms equipped with rotary encoders provide a flexible and portable solution for coordinate measurement. By measuring joint angles and applying kinematic models, the position of the end-effector can be determined in real time.

This study focuses on the design and development of a low-cost coordinate measuring arm using absolute rotary encoders and RS485 communication. The system integrates mechanical design, electronic interfacing, and software-based computation to achieve coordinate

measurement functionality. A Python-based application is used to process encoder data, compute spatial coordinates, and reconstruct geometric shapes such as lines, triangles, and circles.

## 1.2 Problem Statement

Despite the widespread use of coordinate measuring systems, several challenges remain in terms of cost, accessibility, and flexibility. Commercial CMM devices are often prohibitively expensive and complex, limiting their use in academic environments and small laboratories.

Additionally, existing low-cost measurement solutions may lack accuracy, proper calibration, or user-friendly interfaces. Integrating multiple encoders, ensuring reliable communication, and accurately converting angular data into spatial coordinates require careful system design.

Therefore, the problem addressed in this study is:

How to design and implement a low cost, accurate, and user friendly coordinate measuring arm capable of capturing spatial coordinates and reconstructing basic generic shapes using rotary encoder data.

## 1.3 Objectives of the Study

The main objective of this study is to develop a functional and affordable coordinate measuring arm system.

**Specific Objectives:**

- To design and fabricate a three-joint articulated arm using 3D printing technology
- To integrate absolute rotary encoders for accurate angle measurement
- To establish reliable communication using RS485 Modbus RTU protocol
- To develop a Python-based application for data acquisition and processing
- To implement forward kinematics for coordinate calculation
- To design a graphical user interface (GUI) for user interaction
- To enable measurement of geometric shapes such as lines, triangles, and circles
- To improve measurement accuracy through calibration techniques

## 1.4 Scope of Research

**Scope**

This study covers the complete development of a coordinate measuring arm system, including mechanical design, hardware integration, communication setup, and software implementation. The system is capable of measuring planar (2D) coordinates and performing basic geometric analysis.

The application includes features such as:

- Real-time angle acquisition from encoders
- Coordinate computation using kinematics
- Shape-based measurement (line, triangle, circle)
- Visualization using graphical plotting tools

## 1.5 Structure of the Thesis

This thesis is organized into five chapters:

- **Chapter 1: Introduction**
  Presents the background, problem statement, objectives, scope, and structure of the study
- **Chapter 2: Literature Review**
  Reviews existing coordinate measuring systems, encoder technologies, and kinematic modeling methods
- **Chapter 3: Methodology**
  Describes the system design, hardware configuration, communication setup, and software implementation
- **Chapter 4: Results and Discussion**
  Presents experimental results, accuracy analysis, and system performance evaluation
- **Chapter 5: Conclusion and Recommendations**
  Summarizes findings and suggests future improvements

CHAPTER 2: Literature review

# 2.1 Introduction

This chapter provides a comprehensive review of the theoretical foundations and technologies related to coordinate measuring systems. It focuses on the principles of coordinate measurement, articulated arm mechanisms, rotary encoder technologies, industrial communication protocols, kinematic modeling, and geometric reconstruction methods.

The purpose of this chapter is to establish a strong theoretical background for the development of a low-cost coordinate measuring arm. It also identifies limitations in conventional systems, particularly in terms of cost, portability, and accessibility. These limitations motivate the need for an alternative solution that integrates mechanical design, sensing technology, and software-based computation.

# 2.2 Coordinate Measuring Machines (CMMs)

## 2.2.1 Overview of CMM Systems

Coordinate Measuring Machines (CMMs) are widely used in industrial and research applications to measure the geometric characteristics of objects with high precision. These systems play a crucial role in quality control, reverse engineering, and product validation.

A typical CMM system operates through the following steps:

Contact or non-contact probing of an object's surface

Acquisition of discrete coordinate points

Transformation of measured data into a reference coordinate system

Analysis and comparison with design specifications

The accuracy of CMMs depends on factors such as structural rigidity, sensor precision, environmental conditions, and calibration quality.

CMMs are extensively used in industries including aerospace, automotive, and manufacturing, where dimensional accuracy is critical.

## 2.2.2 Types of CMMs

CMMs can be classified based on their structural configuration:

**Bridge-Type CMM**

This is the most widely used type, known for its high precision and stability. It is commonly used in controlled environments such as laboratories.

**Cantilever-Type CMM**

This type provides easier access to the measuring area but has lower structural rigidity compared to bridge-type systems.

**Gantry-Type CMM**

Designed for measuring large objects, gantry-type CMMs offer high load capacity but require significant installation space.

**Articulated Arm CMM**

Articulated arm CMMs consist of multiple rotary joints and links, enabling flexible movement. They are portable, cost-effective, and suitable for on-site measurements.

Due to their flexibility and affordability, articulated arm CMMs are highly relevant to this study.

# 2.3 Articulated Measuring Arms

## 2.3.1 Structure and Working Principle

An articulated measuring arm is composed of several rigid links connected by rotational joints. Each joint provides a degree of freedom, allowing the arm to reach a wide range of positions.

The position of the end-effector is determined by measuring joint angles and applying forward kinematic equations. This approach transforms angular measurements into Cartesian coordinates.

## 2.3.2 Kinematic Chain Concept

The articulated arm can be modeled as a kinematic chain, where each link and joint contributes to the overall motion. The cumulative effect of joint rotations determines the final position of the end-effector.

Errors in any joint can propagate through the chain, affecting overall accuracy.

Therefore, precise measurement and calibration are essential.

## 2.3.3 Advantages

- High flexibility and maneuverability

- Portability and ease of use
- Lower cost compared to traditional CMMs
- Suitable for large or complex objects

### 2.3.4 Limitations

- Error accumulation across joints
- Sensitivity to mechanical imperfections
- Reduced repeatability compared to fixed systems
- Dependence on calibration accuracy

### 2.3.5 Applications

Articulated measuring arms are used in:

- Reverse engineering
- Inspection of large components
- Educational demonstrations
- Prototyping and design validation

## 2.4 Rotary Encoders

### 2.4.1 Principle of Operation

Rotary encoders are electromechanical devices that convert angular motion into digital signals. These signals are used to determine the position of rotating components.

In articulated arms, encoders are mounted at each joint to measure angular displacement.

### 2.4.2 Types of Encoders

**Incremental Encoders**

Incremental encoders generate pulses corresponding to movement. They require a reference position for accurate measurement.

**Absolute Encoders**

Absolute encoders provide a unique digital value for each position, eliminating the need for reinitialization after power loss.

This study uses absolute encoders due to their reliability and accuracy.




*Figure 1 Encoder*

### 2.4.3 Resolution and Precision

The resolution of an encoder determines the smallest detectable angular change.

For a 16-bit encoder:

$$Resolution = 2^{16} = 65,536\ positions\ per\ revolution$$

Higher resolution results in more precise measurements and improved coordinate accuracy.

### 2.4.4 Error Sources in Encoders

Several factors can affect encoder performance:

- Electrical noise and interference
- Mechanical misalignment
- Signal transmission delays
- Quantization error due to finite resolution

These errors must be considered during system design and calibration.

## 2.5 RS485 Communication System

### 2.5.1 Overview

RS485 is a serial communication standard that enables reliable data transmission in industrial environments. It uses differential signaling, which improves resistance to noise.

### 2.5.2 Communication Characteristics

- Supports long-distance communication
- Allows multiple devices on the same bus
- Provides high noise immunity
- Enables stable data transmission

### 2.5.3 Multi-Drop Configuration

In a multi-drop configuration, multiple devices share the same communication line. Each device is identified by a unique address, allowing the master to communicate with specific devices.

This configuration reduces wiring complexity and improves system scalability.

## 2.6 Modbus RTU Protocol

### 2.6.1 Overview

Modbus RTU is a widely adopted serial communication protocol used in industrial automation systems. It enables communication between multiple devices such as sensors, controllers, and actuators. The protocol operates using a master-slave (client-server) architecture, where a single master device initiates communication and multiple slave devices respond to requests.

In the context of measurement systems, Modbus RTU provides a reliable and efficient method for acquiring sensor data, especially when multiple devices are connected in a network.

### 2.6.2 Communication Structure

Each Modbus RTU message consists of several components:

- **Slave Address (ID):** Identifies the target device
- **Function Code:** Specifies the operation (e.g., read register)
- **Data Field:** Contains the requested or transmitted data
- **CRC (Cyclic Redundancy Check):** Ensures data integrity

The use of CRC error checking enhances communication reliability by detecting transmission errors.

### 2.6.3 Communication Process

The communication process follows a structured sequence:

1. The master device sends a request frame to a specific slave
2. The addressed slave processes the request
3. The slave sends a response containing the requested data
4. The master validates the response using CRC

If no response is received within a defined timeout period, the master may retry or report a communication error.

### 2.6.4 Advantages of Modbus RTU

- Simple implementation and low overhead
- Efficient binary data transmission
- High compatibility with industrial devices
- Ability to support multiple devices on a single bus

### 2.6.5 Application in This Study

In this system, Modbus RTU is used to retrieve position data from multiple rotary encoders. Each encoder is assigned a unique slave ID, allowing the master (computer) to query each encoder individually. This ensures organized and conflict-free communication within the RS485 network.

## 2.7 Kinematic Modeling

### 2.7.1 Introduction to Kinematics

Kinematics is a branch of mechanics that describes motion without considering the forces that cause it. In robotic and measurement systems, kinematics is used to determine the position and orientation of components based on joint parameters.

### 2.7.2 Forward Kinematics

Forward kinematics involves calculating the position of the end-effector based on known joint angles and link lengths. It is essential for converting raw encoder data into meaningful spatial coordinates.

For a three-link planar system, the position is defined as:

$$X = L_1 cos(\theta_1) + L_2 cos(\theta_1 + \theta_2) + L_3 cos(\theta_1 + \theta_2 + \theta_3) Y = L_1 sin(\theta_1) + L_2 sin(\theta_1 + \theta_2) + L_3 sin(\theta_1 + \theta_2 + \theta_3)$$

### 2.7.3 Coordinate Transformation

The calculation involves transforming joint-based angular data into a Cartesian coordinate system. Each link contributes to the final position based on its length and orientation.

### 2.7.4 Error Propagation

Errors in joint angle measurement can accumulate through the kinematic chain. Even small angular errors may result in significant positional inaccuracies at the end-effector. This makes precise measurement and calibration critical.

### 2.7.5 Importance in Measurement Systems

Kinematic modeling serves as the mathematical foundation of the coordinate measuring arm. Without accurate kinematic equations, it is not possible to convert encoder readings into usable coordinate data.

## 2.8 Geometric Reconstruction

### 2.8.1 Point-Based Measurement Concept

In coordinate measuring systems, objects are represented using discrete coordinate points. These points are obtained through physical measurement using the probe or end-effector.

### 2.8.2 Shape Definition Using Points

Geometric shapes can be defined based on the number of measured points:

- **Line:** Defined by two points
- **Triangle:** Defined by three points
- **Circle:** Defined by three or more points

### 2.8.3 Distance Measurement

The distance between two points is calculated using the Euclidean distance formula:

$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

#### 2.8.4 Area Calculation

For a triangle, the area can be computed using Heron's formula:

$$A = \sqrt{s(s - a)(s - b)(s - c)}$$

where $s$is the semi-perimeter.

### 2.8.5 Circle Fitting

A circle can be defined using three non-collinear points. Mathematical methods are used to determine the center and radius of the circle.

### 2.8.6 Importance

Geometric reconstruction allows the system to interpret measured data and represent real-world objects digitally.

## 2.9 Software Systems for Measurement

### 2.9.1 Role of Software

Software acts as the central component that integrates hardware, processes data, and provides user interaction. It transforms raw sensor data into meaningful outputs.

### 2.9.2 Functional Architecture

A typical measurement software system includes:

- Data acquisition module
- Data processing module
- Visualization module
- User interface module

### 2.9.3 Python in Engineering Applications

Python is widely used due to its:

- Ease of learning and readability
- Extensive libraries for engineering applications
- Flexibility in integrating hardware and software

### 2.9.4 Libraries and Tools

In this study:

- Tkinter is used for GUI development
- pymodbus is used for communication
- matplotlib is used for visualization

### 2.9.5 Benefits of Software Integration

- Real-time data processing
- Interactive measurement control
- Visualization of results
- Improved user experience

## 2.10 Calibration Techniques

### 2.10.1 Importance of Calibration

Calibration ensures that the measurement system produces accurate and reliable results. Without calibration, measurement errors can significantly affect system performance.

### 2.10.2 Types of Calibration

**Offset Calibration**

Corrects initial angular offsets of encoders.

**Geometric Calibration**

Adjusts link lengths and system geometry.

**Scaling Correction**

Applies correction factors to reduce systematic errors.

### 2.10.3 Error Sources

Measurement errors may arise from:

- Mechanical misalignment
- Sensor inaccuracies
- Assembly tolerances
- Environmental factors

### 2.10.4 Calibration Impact

Proper calibration improves:

- Accuracy
- Repeatability
- Reliability

## 2.11 Summary

This chapter presented a detailed review of the theoretical and technological foundations of coordinate measuring systems. It covered key topics including Modbus RTU communication, kinematic modeling, geometric reconstruction, software systems, and calibration techniques.

The analysis highlights that the integration of accurate sensors, reliable communication protocols, and robust mathematical models is essential for developing an effective measurement system. Furthermore, software plays a critical role in processing and visualizing measurement data.

The reviewed concepts demonstrate that a low-cost coordinate measuring arm is feasible when proper design, calibration, and system integration are implemented. These findings provide the necessary foundation for the methodology described in the following chapter.

# 3.1 Introduction

This chapter presents a detailed description of the methodology used in the design and implementation of the coordinate measuring arm system. The development process integrates mechanical design, hardware interfacing, communication systems, and software-based computation.

The objective of this methodology is to develop a low-cost, functional, and reasonably accurate coordinate measuring system using readily available components such as rotary encoders, RS485 communication modules, and 3D-printed structures. The methodology emphasizes practical implementation, iterative testing, and system integration.

# 3.2 Design Approach and Development Strategy

The system was developed using an iterative engineering approach, where each subsystem was designed, tested, and refined before integration.

The development process followed these stages:

1. Conceptual design of the measuring arm
2. Mechanical modeling and fabrication
3. Electronic system integration
4. Communication setup
5. Software development and testing
6. Calibration and accuracy improvement

This step-by-step approach ensured that errors could be identified and corrected early in the development process.

# 3.3 Mechanical Design and Fabrication

## 3.3.1 CAD Design Using Autodesk Inventor

The mechanical components of the coordinate measuring arm were designed using **Autodesk Inventor**, a professional CAD software widely used in engineering design.

The design process involved:

- Creating 3D models of links and joints
- Defining accurate dimensions for all components
- Designing encoder mounting interfaces
- Ensuring proper alignment of rotational axes

Special attention was given to the connection between encoder shafts and arm links. The joints were designed with tight tolerances to ensure a precise fit, minimizing mechanical play and improving measurement accuracy.

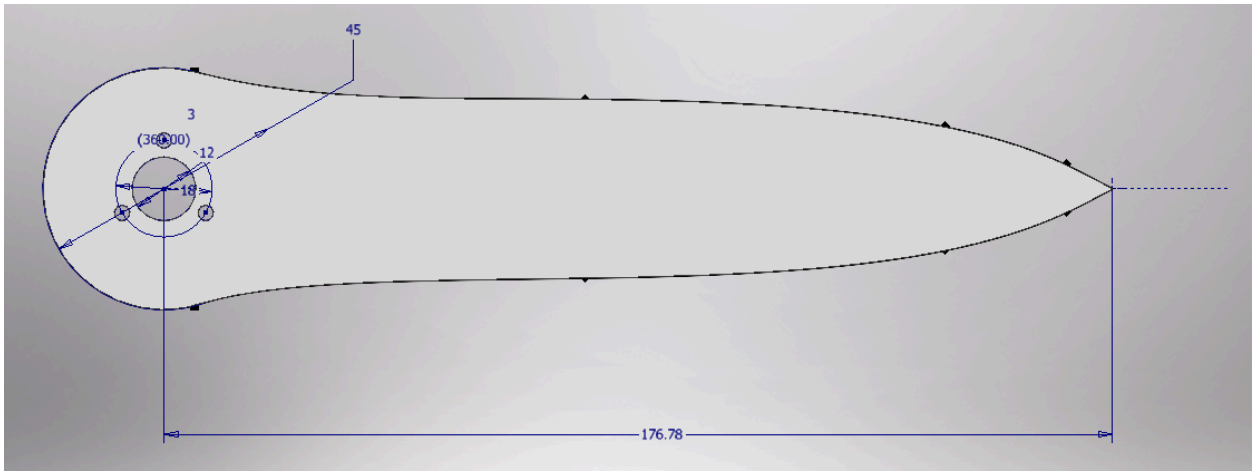



*Figure 2 arm tip*

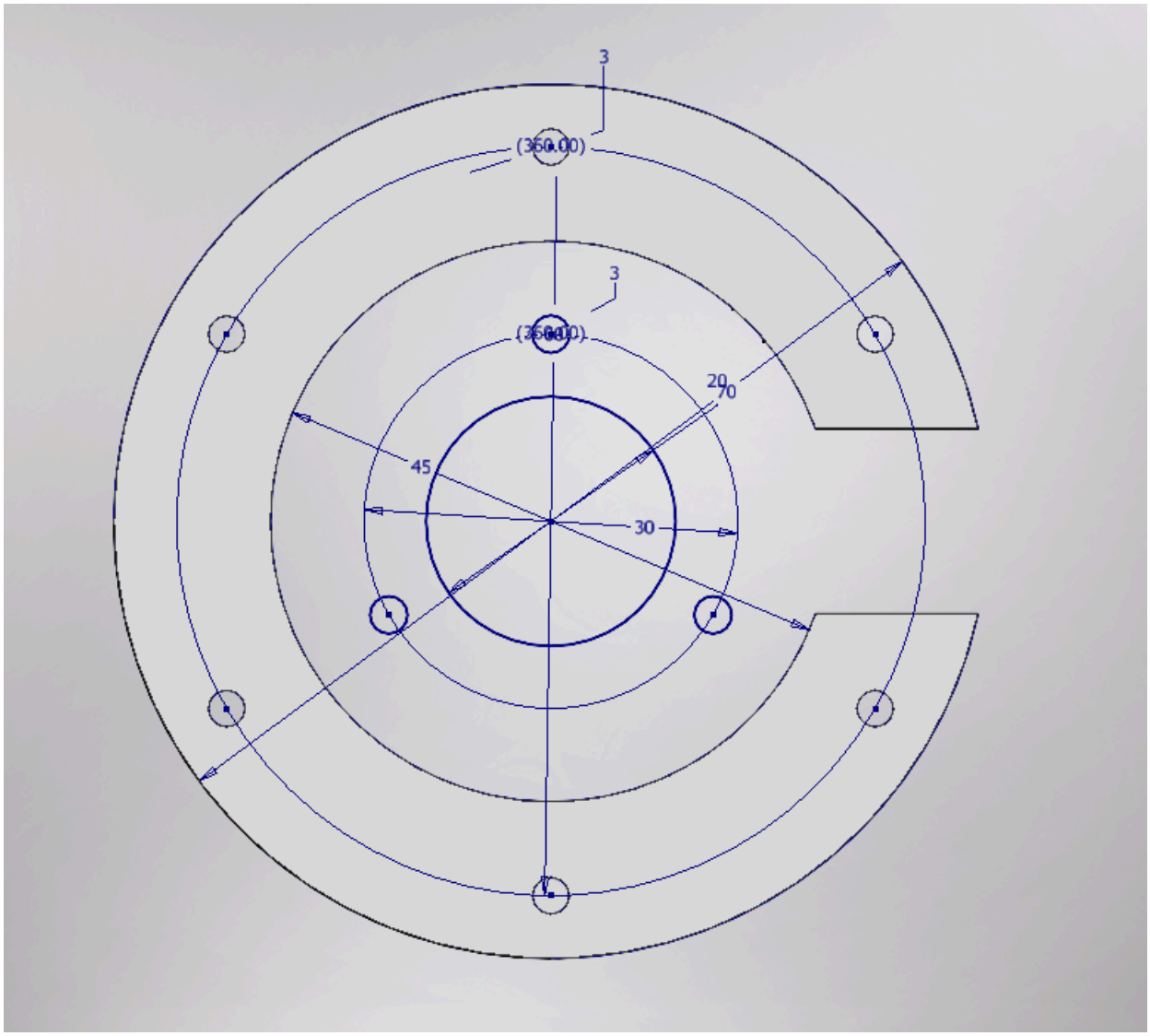



*Figure 3 lower part*

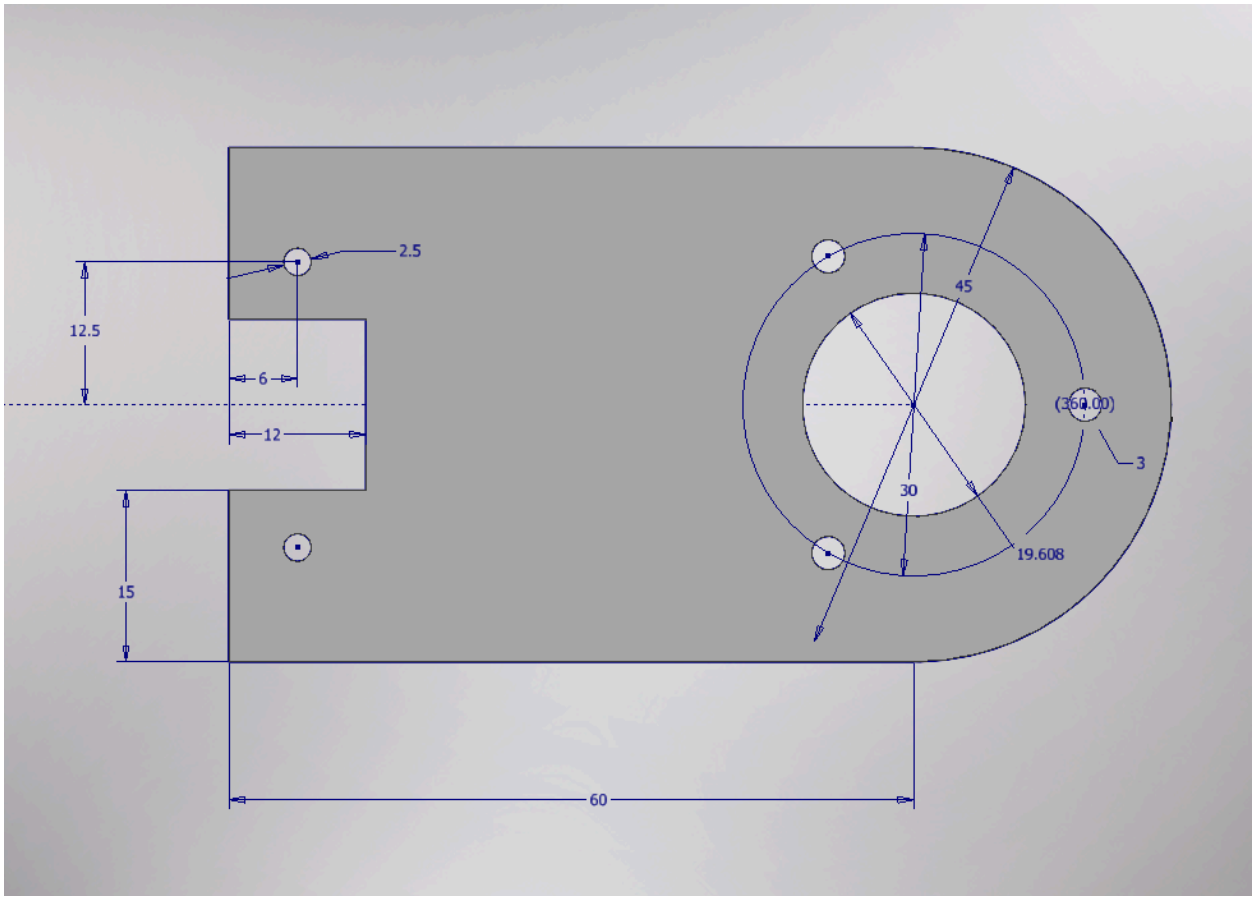



*Figure 4 Hinge*

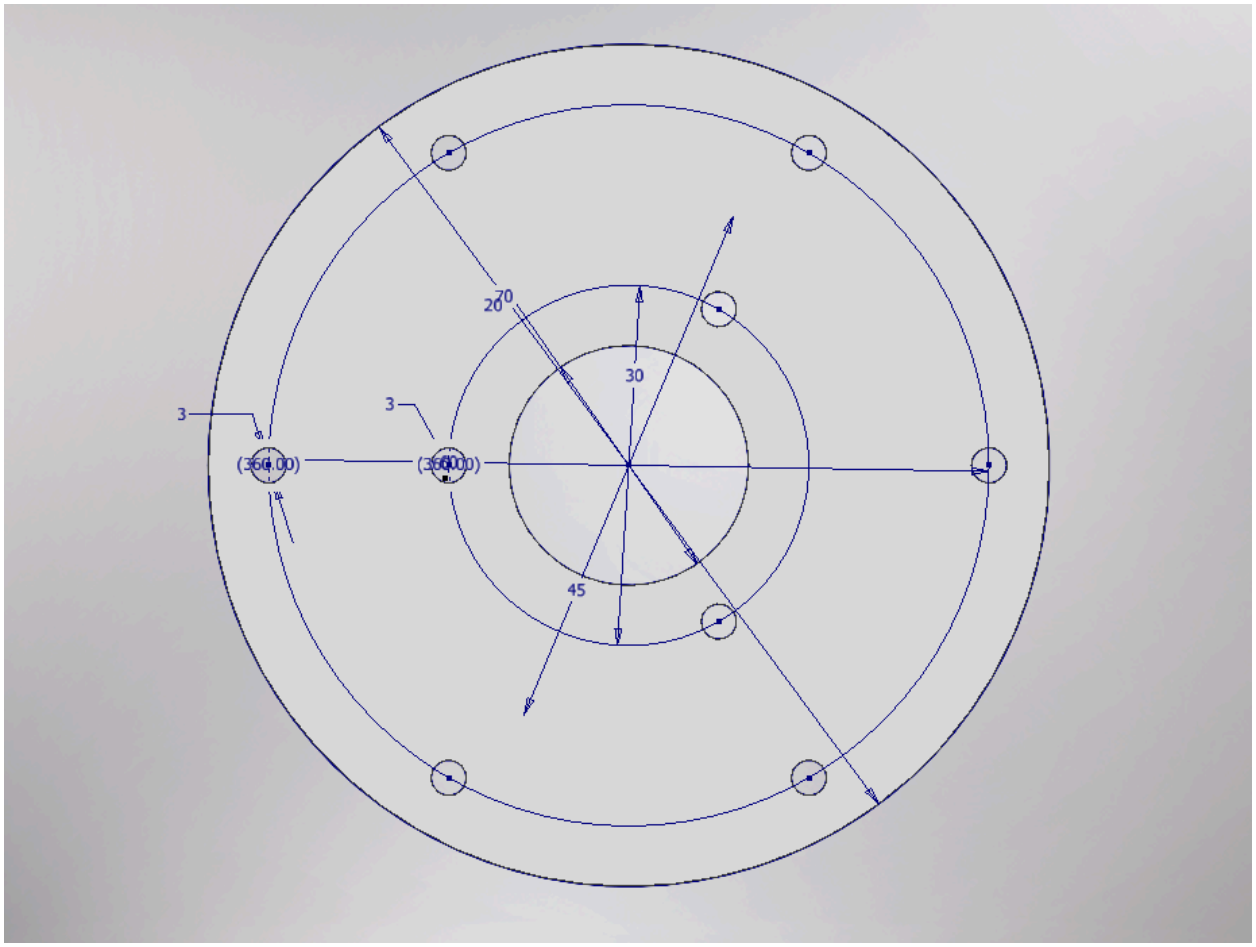



*Figure 5 lower part hinge*

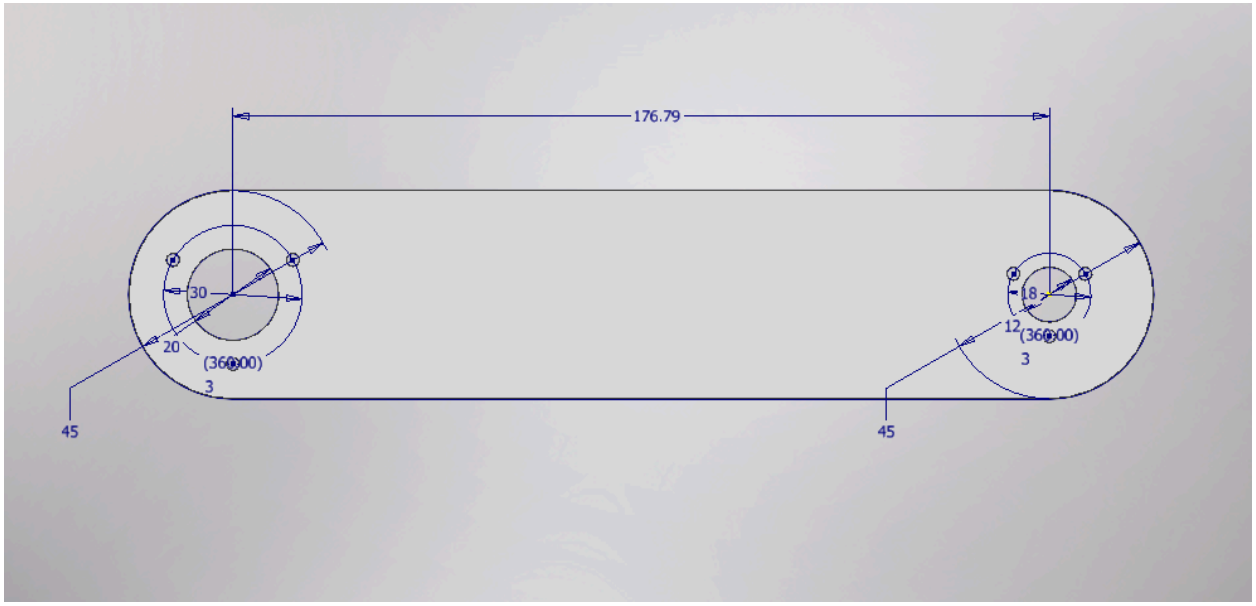



*Figure 6 Arm link*

### 3.3.2 Link Design and Dimensions

The measuring arm consists of three rigid links:

• L1 = 48 mm

• L2 = 176.79 mm

• L3 = 176.78 mm

- $L_1 = 48\,mm$
- $L_2 = 176.79\,mm$
- $L_3 = 176.78\,mm$

These values were determined based on design constraints and later refined during calibration.

The links were designed to:

- Maintain structural rigidity
- Reduce deformation during operation
- Provide stable mounting for encoders

### 3.3.3 3D Printing and Fabrication

All mechanical components were fabricated using **Fused Deposition Modeling (FDM) 3D printing**.

Advantages of using 3D printing:

- Low manufacturing cost
- Rapid prototyping
- Easy modification of design

Key printing considerations:

- Layer height for dimensional accuracy
- Infill percentage for strength
- Print orientation to reduce stress

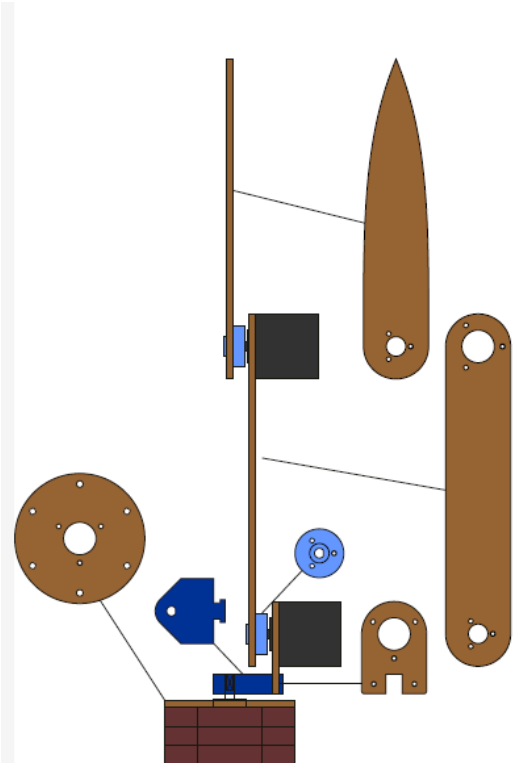


*Figure 7 3D printed components simplified*

### 3.3.4 Assembly of the Mechanical System

The printed components were assembled using bolts and fasteners. Each link was mounted directly onto the encoder shaft using the designed coupling.

Due to the precise CAD design, the connections between the encoders and links were tightly fitted, resulting in:

- Minimal mechanical looseness
- Improved rotational accuracy
- Stable structural configuration



*Figure 8 Assembled coordinate measuring arm*



*Figure 9 Assembled coordinate measuring arm*

### 3.3.5 Mechanical Error Considerations

Despite careful design, several mechanical limitations exist:

- Slight flexibility of 3D printed materials
- Small tolerances in printed parts
- Alignment errors during assembly

These factors contribute to measurement inaccuracies and are addressed through calibration.

## 3.4 Electronic System Integration

The electronic subsystem of the coordinate measuring arm is responsible for acquiring angular position data from the rotary encoders and transmitting it to the computer for processing. This subsystem consists of three absolute rotary encoders, a power supply unit, and an RS485 communication interface.

### 3.4.1 Rotary Encoder Implementation

Three absolute rotary encoders were used to measure the angular position of each joint in the articulated arm. These encoders operate within a voltage range of 5–24 V, making them compatible with a wide range of power sources.

Each encoder was assigned a unique slave ID for communication:

- Joint 1 → ID 1
- Joint 2 → ID 3
- Joint 3 → ID 2

The encoders provide high-resolution position data, which is essential for accurate coordinate computation.

### 3.4.2 Power Supply Configuration

A regulated power supply with an output voltage of 18 V was used to power all three encoders. Since the encoders support an input voltage range of 5–24 V, the 18 V supply falls within the acceptable operating range.

The power supply was distributed to all encoders in parallel, ensuring that each encoder received the required voltage. Proper grounding was maintained across all components to ensure stable operation and minimize electrical noise.

### 3.4.3 Breadboard-Based Wiring Setup

The electrical connections between the encoders, power supply, and communication interface were implemented using a breadboard. This approach provided flexibility during development and allowed easy modification of connections.

The wiring configuration includes:

- Common VCC line (18 V) distributed to all encoders
- Common GND line shared across all components
- RS485 communication lines (A and B) connected in parallel to all encoders

Using the breadboard, the encoders were connected in a multi-drop configuration, where all devices share the same communication bus.

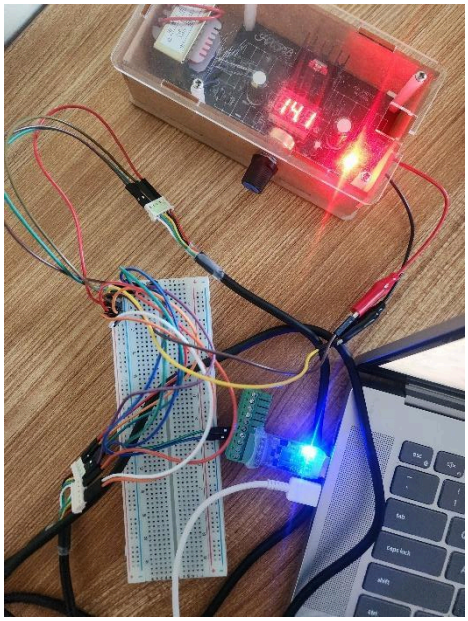


*Figure 10 Breadboard wiring of encoders and RS485 interface*

### 3.4.4 RS485 Communication Interface

The communication between the encoders and the computer was established using an RS485 interface. A USB-to-RS485 converter was used to connect the system to the computer.

RS485 communication was selected due to its advantages:

- High noise immunity
- Support for multiple devices on a single bus
- Reliable long-distance communication

Each encoder communicates using the Modbus RTU protocol, allowing the master device (computer) to request position data individually.

### 3.4.5 Communication Configuration

The communication parameters were configured as follows:

- Baud rate: 9600 (configurable)
- Data bits: 8
- Parity: None
- Stop bits: 1
  These settings ensured stable and reliable communication between the encoders and the software system.

### 3.4.6 System Reliability Considerations

Several precautions were taken to ensure stable operation of the electronic system:

- Proper grounding of all components
- Secure connections on the breadboard
- Consistent power supply distribution
- Verification of communication parameters

Despite the use of a breadboard, which is typically intended for prototyping, the system demonstrated stable performance during testing.

# 3.5 Software Development

The software component of the developed coordinate measuring arm plays a critical role in enabling communication with the encoders, processing measurement data, performing kinematic calculations, and providing a user-friendly interface for real-time interaction. The software system was developed using the Python programming language due to its flexibility, readability, and extensive ecosystem of scientific and graphical libraries.

The complete software architecture consists of three main modules: config.py, modbus_reader.py, and gui_app.py. Each module was designed with a specific purpose to ensure modularity, maintainability, and ease of debugging.

## 3.5.1 Configuration Module (config.py)

The config.py file centralizes all configurable parameters of the system. This includes communication settings, register addresses, encoder specifications, graphical interface settings, and plotting configurations.

The Modbus communication parameters defined in this file include:

- Serial port (e.g., COM4)
- Baud rate (9600 bps)
- Slave ID
- Data bits (8-bit)
- Stop bits (1)
- Parity (None)

By isolating these parameters, the system allows rapid modification without altering the core logic of the application. This is particularly important in industrial communication systems, where different devices may require different configurations.

The encoder characteristics are also defined in this file. A 16-bit encoder resolution is assumed, resulting in a maximum value of 65,536 counts per revolution. The conversion from raw encoder value to angular displacement is defined by the formula:

$$\theta = \frac{raw\ value}{65536} \times 360^{\circ}$$

This abstraction ensures consistency across all parts of the program when interpreting encoder data.

### 3.5.2 Console-Based Testing (main.py)

Before developing the graphical interface, a console-based application was implemented to validate communication between the computer and the encoder.

The main.py script continuously reads encoder values using the EncoderReaderWithStats class and displays:

- Timestamp
- Angle value
- Raw encoder data
- Communication status

This testing stage was essential for verifying:

- Correct COM port selection
- Proper Modbus communication
- Stability of data transmission
- Error handling during communication failures

Additionally, statistical metrics such as total reads, error count, success rate, minimum angle, and maximum angle were recorded. These metrics provided insight into communication reliability and helped identify issues such as timeouts or incorrect Modbus parameters.

### 3.5.3 Modbus Communication Layer (modbus_reader.py)

The modbus_reader.py module is responsible for handling low-level communication between the software and the encoder hardware via RS485 using the Modbus RTU protocol.

The EncoderReader class encapsulates the connection logic and register reading functionality. It establishes communication using the pymodbus library and performs register reads through the read_holding_registers() function.

To ensure compatibility with different versions of the pymodbus library, the program attempts multiple keyword arguments (device_id and slave) when performing register reads. This improves robustness and ensures the software can operate across different environments.

Error handling is implemented extensively in this module. The system checks for:

- No response from the device
- Invalid or empty register data
- Modbus protocol errors

If a read operation fails, the function returns None, allowing higher-level logic to handle the failure appropriately.

The extended class EncoderReaderWithStats enhances the base functionality by tracking performance metrics, including successful reads and communication failures. This feature was particularly useful during system debugging and validation.

### 3.5.4 Graphical User Interface (gui_app.py)

The primary user interaction is handled through a graphical user interface implemented using the Tkinter library. The main class, EncoderGUI, integrates communication, data processing, and visualization into a single application environment.

The GUI is organized into multiple tabs:

- Monitor tab (real-time data display and control)
- Diagnostics tab (Modbus testing and scanning)
- Settings tab (configuration parameters)
- 3D visualization tab (spatial representation)
- Log tab (debugging information)

Connection and Diagnostics

The GUI allows the user to select communication parameters such as COM port, baud rate, parity, stop bits, and encoder IDs. Upon connection, the system does not assume successful operation immediately. Instead, it performs an automatic encoder test to verify communication with each joint.

The system supports diagnostic tools such as:

- Encoder ID testing
- Slave ID scanning (IDs 1–20)
- Register address scanning (addresses 0–4)

These tools are critical for identifying communication issues such as incorrect slave IDs, wiring problems, or mismatched register addresses.

Encoder Data Processing

The system reads three encoders corresponding to the three joints of the measuring arm. The encoder IDs are configurable and mapped as follows:

- Joint 1 → ID 1
- Joint 2 → ID 3
- Joint 3 → ID 2

Raw encoder values are converted into angular positions using the predefined conversion formula. These angles are then corrected using offset calibration and direction inversion.

Offset calibration is performed using the “Set Home Position” function, which records the current encoder values as reference positions. Subsequent readings are adjusted relative to these offsets to ensure accurate angular measurements.

Forward Kinematics Implementation

The software calculates the position of the end-effector using forward kinematics. The arm is modeled as a three-link planar manipulator with the following link lengths:

- $L_1 = 48\,mm$
- $L_2 = 176.79\,mm$
- $L_3 = 176.78\,mm$

The end-effector coordinates are computed using trigonometric relationships:

$$X = L_1 cos(\theta_1) + L_2 cos(\theta_1 + \theta_2) + L_3 cos(\theta_1 + \theta_2 + \theta_3) Y = L_1 sin(\theta_1) + L_2 sin(\theta_1 + \theta_2) + L_3 sin(\theta_1 + \theta_2 + \theta_3)$$

These equations convert angular measurements into Cartesian coordinates, enabling spatial measurement of objects.

### Real-Time Coordinate Tracking

A real-time measurement feature was implemented to continuously update the position of the arm. When activated, the system reads encoder data at a fixed interval (100 ms) using a non-blocking loop (Tkinter.after()).

This ensures that:

- The GUI remains responsive
- Measurements update continuously
- The user can observe motion dynamically

The real-time display includes joint angles, X-Y-Z coordinates, and graphical visualization.

### Global Scale Calibration

Due to mechanical imperfections such as 3D printing tolerances, joint backlash, and alignment errors, raw measurements may not match actual physical dimensions. To compensate for this, a global scale calibration method was implemented.

The user measures a known reference length by capturing two points. The system calculates the measured distance and compares it to the actual length. A scaling factor is then computed:

$$Scale = \frac{Actual\ Length}{Measured\ Length}$$

This factor is applied to all subsequent coordinate calculations, significantly improving measurement accuracy.

### Geometric Measurement Functions

The software supports multiple measurement modes:

- Line measurement (distance between two points)

- Triangle measurement (side lengths and internal angles)
- Circle measurement (center and radius from three points)

These features allow the system to perform basic geometric reconstruction of objects. However, it was observed that circle and triangle measurements are highly sensitive to coordinate errors, making line measurement the most reliable method during initial testing.

Visualization was implemented using the Matplotlib library integrated within the Tkinter graphical user interface. The system displays the configuration of the measuring arm, including the positions of individual joints, the end-effector, captured measurement points, and reconstructed geometric shapes.

A three-dimensional visualization module was developed to represent the actual movement of the measuring arm. Unlike the initial implementation, which displayed only linear marker positions, the improved system calculates the true spatial position of each joint using forward kinematics.

The visualization includes four key points:

- Base (reference position)
- Joint 1
- Joint 2
- Tip (end-effector)

The tip is not directly measured by an encoder but is calculated using the combined effect of all joint angles and link lengths. These points are connected to form a continuous kinematic chain, accurately representing the structure and motion of the arm.

The system updates this visualization in real time, allowing the user to observe the dynamic movement and orientation of the arm. This provides immediate visual feedback and improves understanding of the measurement process.

---

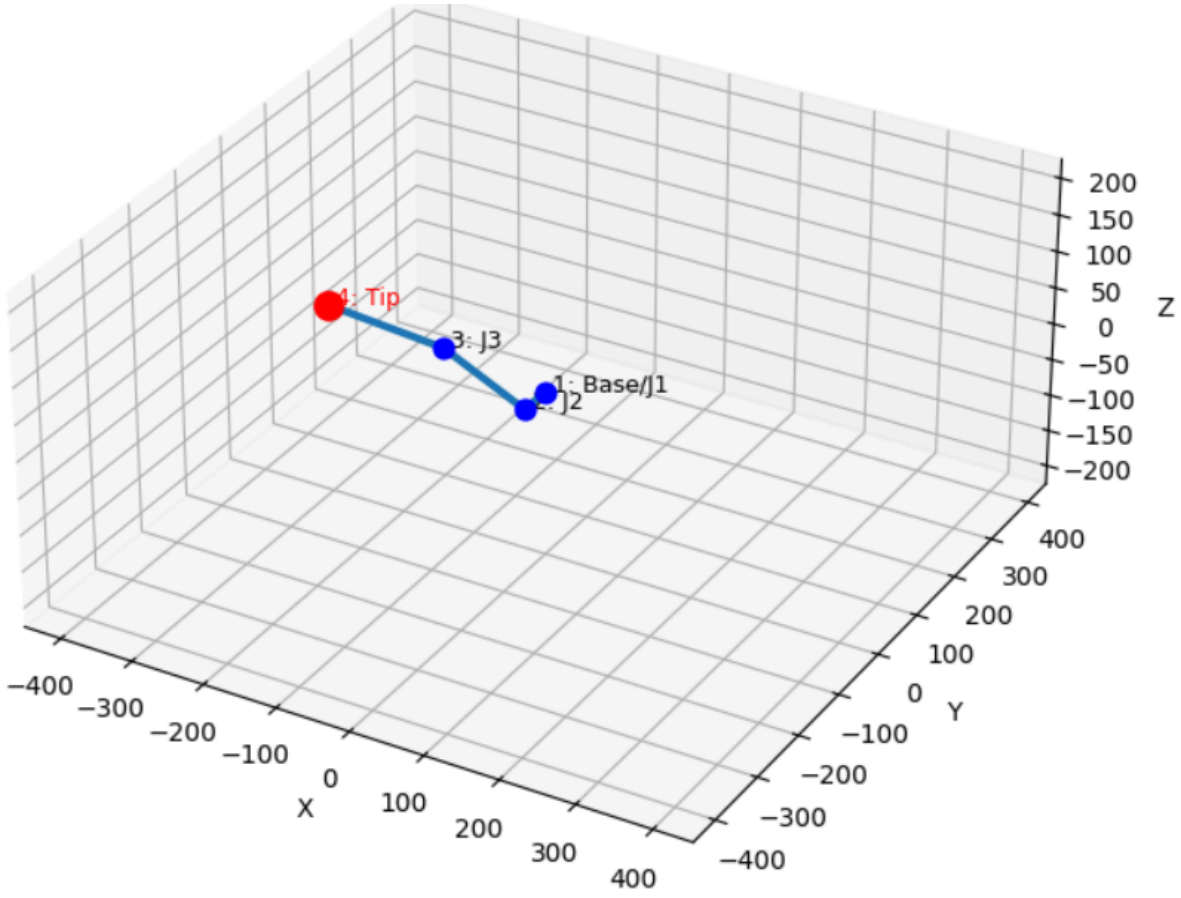



*Figure 11 3D visual*

---

The improved visualization ensures that the arm is displayed as a connected mechanical structure rather than a straight-line approximation. This enhancement significantly improves system usability and provides a more realistic representation of the measuring process.

### 3.5.5 Summary

The developed software successfully integrates communication, computation, and visualization into a single platform. It enables real-time measurement, supports multiple geometric operations, and provides calibration tools to improve accuracy. The modular design ensures flexibility and allows future improvements such as enhanced calibration methods, 3D measurement capabilities, and improved error compensation techniques.

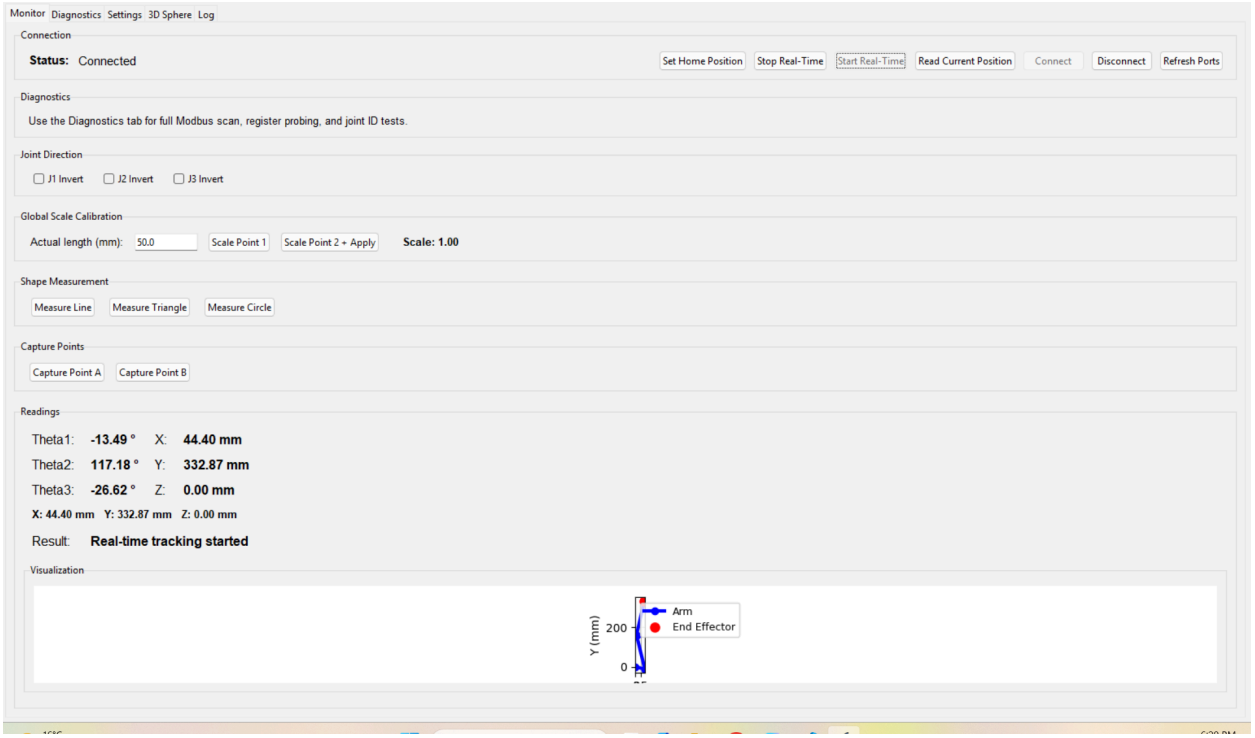



*Figure 12 User interface*

# 3.6 Calibration and Accuracy Improvement

Calibration is a critical step in ensuring that the coordinate measuring arm produces accurate and repeatable results. Without proper calibration, systematic errors arising from mechanical assembly tolerances, encoder offset positions, and 3D printing dimensional inaccuracies would propagate through the kinematic chain and produce unreliable coordinate output.

Offset Calibration (Home Position Setting) Each absolute rotary encoder retains its position value relative to its internal zero reference, which does not necessarily correspond to the intended physical zero angle of the joint. To compensate for this, an offset calibration procedure was implemented. When the user places the arm in a defined home configuration and activates the "Set Home Position" function, the software records the current raw encoder values as reference offsets. All subsequent angle readings are adjusted by subtracting these offsets, ensuring that the computed joint angles reflect actual physical rotation from the home position.

Global Scale Correction Due to accumulated tolerances in 3D-printed link dimensions, the nominal link lengths (L1 = 48 mm, L2 = 176.79 mm, L3 = 176.78 mm) may differ slightly from their fabricated values. A global scale correction method was implemented to compensate for

this. The user measures a known reference distance and the system computes a correction factor:

Scale = Actual Length / Measured Length

This factor is applied uniformly to all subsequent coordinate calculations. For example, if a 100 mm reference object was measured as 97.5 mm, the scale factor of 1.026 would be applied, improving absolute accuracy across the workspace.

Mechanical Alignment Verification Prior to measurement sessions, the arm joints were manually inspected to confirm proper seating of encoder shafts within the link couplings. Any detectable rotational play was noted and minimized through fastener adjustment.

Together, these calibration steps reduced systematic measurement error significantly, as evidenced by the experimental results.

## 3.7 System Workflow

The system operates as follows:

1. Establish communication
2. Set home position
3. Acquire encoder data
4. Convert to angles
5. Compute coordinates
6. Perform measurement
7. Display results

## 3.8 Summary

This chapter described the methodology used to develop the coordinate measuring arm system. The system integrates mechanical design using Autodesk Inventor, fabrication using 3D printing, encoder-based sensing, and Python-based software development.

The use of GitHub Copilot improved development efficiency, while calibration techniques enhanced measurement accuracy. The resulting system provides a functional and cost-effective solution for coordinate measurement.

## CHAPTER 4: RESULTS AND DISCUSSION

# 4.1 Introduction

This chapter presents the experimental results obtained from the developed coordinate measuring arm system. The system performance was evaluated by measuring known geometric dimensions including linear distances, triangular shapes, and circular objects. The measured results were compared with actual physical values to determine the accuracy, repeatability, and reliability of the system.

In addition, the functionality of the developed graphical user interface (GUI), including real-time visualization and measurement features, is also demonstrated.

# 4.2 Line Measurement Results

Line measurement was used as the primary method for evaluating system accuracy. Known lengths of **5 cm, 10 cm, and 14 cm** were measured three times each.

**Table 4.1 Line Measurement Results**

| Actual Length (cm) | Measurement 1 (cm) | Measurement 2 (cm) | Measurement 3 (cm) | Average (cm) | Error (%) |
|---|---|---|---|---|---|
| **5** | **4.9** | **5.1** | **5.1** | **5.03** | **0.6%** |
| **10** | **9.8** | **9.9** | **9.7** | **9.80** | **2.0%** |
| **14** | **13.9** | **13.8** | **13.9** | **13.87** | **0.9%** |

The line measurement was performed by capturing two distinct points, denoted as point A and point B, using the end-effector of the articulated arm. Each point represents a spatial coordinate calculated through forward kinematics based on the encoder readings.

The distance between the two points was computed using the Euclidean distance formula:

$$d = \sqrt{\left(x_2 - x_1\right)^2 + \left(y_2 - y_1\right)^2}$$

where $\left(y_1\right)$and $\left(y_2\right)$are the coordinates of points A and B, respectively.

The accuracy of the line measurement depends on the stability of the arm during point capture and the precision of encoder readings. Small deviations in joint angles can lead to noticeable variations in the measured length. Repeated measurements showed consistent results with minor variations, indicating acceptable system reliability.

The implementation of the line distance calculation algorithm is provided in Appendix A.

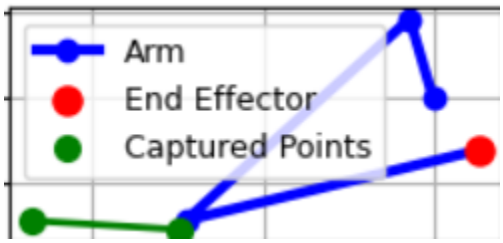



*Figure 13 GUI interface showing line measurement between two captured points*

# 4.3 Triangle Measurement Results

Triangle measurements were performed using known side lengths. Two different triangles were tested to evaluate consistency and geometric accuracy.

**Triangle 1**

**Table 4.2 Triangle 1 Side Lengths**

| Parameter | Actual Value | Measured Value |
|---|---|---|
| AB (cm) | 3.5 | 3.509 |
| BC (cm) | 4.8 | 4.9 |
| CA (cm) | 6.2 | 6.413 / 5.94 |
| ∠A (°) | 50 | 50.2° |

| Parameter | Actual Value | Measured Value |
|---|---|---|
| ∠B (°) | 96.9 | 96.9° |
| ∠C (°) | 33.1 | 32.9° |

**Triangle 2**

**Table 4.3 Triangle 2 Measurement Results**

| Parameter | Actual Value | Measured Value |
|---|---|---|
| AB (cm) | 2.9 | 2.877 |
| BC (cm) | 4.1 | 4.17 |
| CA (cm) | 3.4 | 3.553 |
| ∠A (°) | 80° | 80.1° |
| ∠B (°) | 56.6° | 56.2° |
| ∠C (°) | 43.4° | 43.7° |

Triangle measurement was conducted by capturing three points, labeled A, B, and C, using the articulated arm. These points define the vertices of the triangle in the Cartesian coordinate system.

The side lengths were calculated using the Euclidean distance formula:

$$AB = \sqrt{\left(x_B - x_A\right)^2 + \left(y_B - y_A\right)^2} \quad BC = \sqrt{\left(x_C - x_B\right)^2 + \left(y_C - y_B\right)^2} \quad CA = \sqrt{\left(x_A - x_C\right)^2 + \left(y_A - y_C\right)^2}$$

The internal angles of the triangle were determined using the law of cosines:

$$cos(A) = \frac{AB^2 + CA^2 - BC^2}{2 \cdot AB \cdot CA}$$

Similar expressions were used to compute angles B and C.

The results showed that angle measurements were relatively accurate, while some discrepancies were observed in side lengths. These errors are primarily caused by cumulative

joint inaccuracies and slight misalignment during point capture. Repeated measurements improved consistency, indicating that the system performs reliably when points are captured carefully.

The implementation of the triangle measurement algorithm, including side length and angle calculations, is provided in Appendix A.

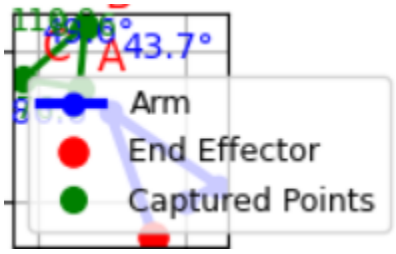



*Figure 14 Triangle measurement displayed in the application*

## 4.4 Circle Measurement Results

Circle measurements were conducted using known diameters.

| Actual Radius (cm) | Measured Radius (cm) | Error (%) |
|---|---|---|
| 4.00 | 4.131 | 3.3% |
| 1.75 | 1.758 | 0.46% |

The results show that the system provides good accuracy for circular measurements, with errors ranging from approximately 0.5% to 3.3%. This accuracy is influenced by the distribution and number of captured points on the circle.

In this study, circle measurement was performed by capturing multiple points along the circumference using the articulated arm. Each captured point represents the end-effector position calculated through forward kinematics. A minimum of three non-collinear points was required to define a circle.

The circle parameters (center and radius) were then computed using a geometric circle-fitting algorithm based on these captured points. The algorithm determines the intersection of perpendicular bisectors formed by the selected points to estimate the center, while the radius is calculated as the distance from the center to any point on the circle.

When the captured points are well distributed around the circumference, the fitting algorithm produces more reliable and accurate results. However, if the points are clustered or unevenly selected, the measurement error increases.

The implementation of the circle fitting algorithm is provided in Appendix A.

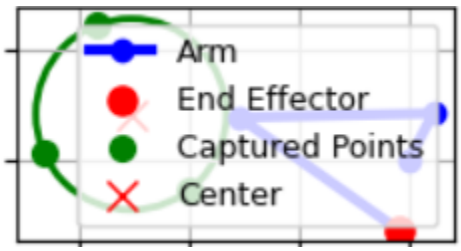



*Figure 15 Circle measurement and fitted radius displayed in the application*

# 4.5 Visualization Results

The 3D visualization represents the real-time movement of the articulated arm based on encoder readings. Each joint position is calculated using forward kinematics, and the arm is modeled as a series of connected links.

In the visualization, four key points are displayed:

- Base (Joint 1)
- Joint 2
- Joint 3
- End-effector (tip)

The coordinates of each joint are computed incrementally based on link lengths and joint angles. These points are then connected to form the complete arm structure in 3D space.

This visualization allows the user to observe how the arm moves in real time and how the end-effector position changes during measurement. It also helps identify mechanical or calibration errors, such as incorrect joint direction or scaling issues.

Including the end-effector as a fourth point provides a more accurate representation of the measurement process, as it directly corresponds to the captured spatial coordinates used for geometric calculations.

The implementation of the real-time 3D visualization and kinematic modeling algorithm is provided in Appendix A.

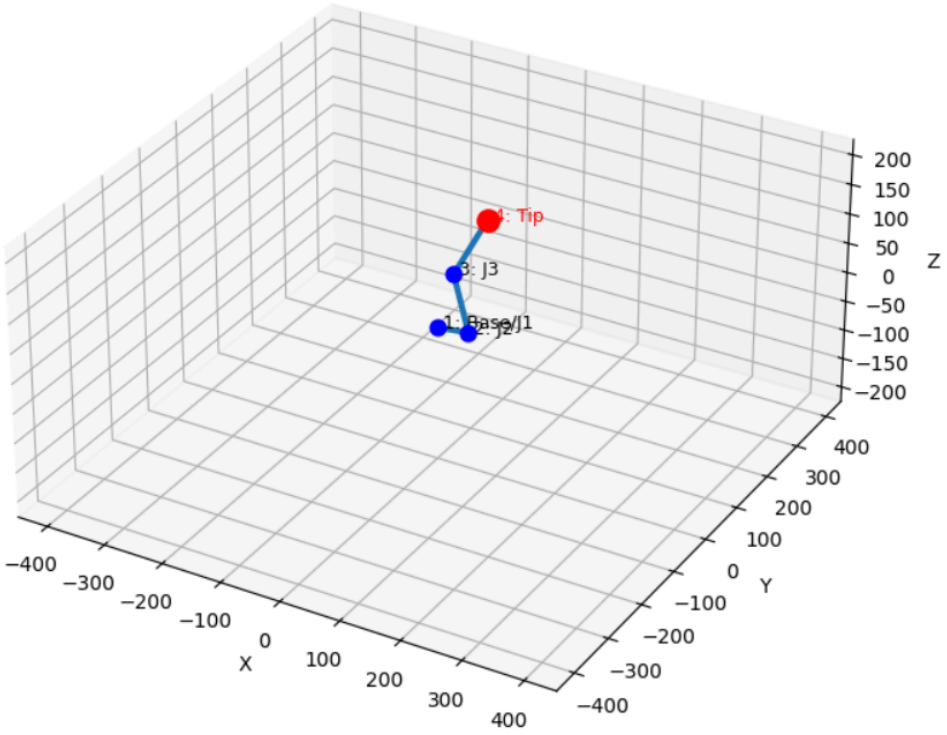



*Figure 16 3D visualization of the measuring arm showing joints and tip position*

# 4.6 Overall System Performance

The overall performance of the system can be summarized as follows:

- **Line measurement:** High accuracy (±2%)
- **Triangle measurement:** Moderate accuracy
- **Circle measurement:** Good accuracy when points are selected properly
- **Visualization:** Effective for real-time feedback

# 4.7 Error Analysis

The measurement errors observed in this study arise from several distinct sources, each contributing differently to overall system uncertainty.

The most significant contributor is the flexibility of 3D-printed PLA components. Although PLA provides sufficient structural rigidity for light-duty use, micro-deflections in the arm links under

the weight of the encoders introduce angular deviations that cannot be fully compensated through software calibration. This effect is most pronounced in long-reach positions where the links are nearly fully extended.

Encoder quantization error represents a secondary but unavoidable source of inaccuracy. A 16-bit absolute encoder provides 65,536 discrete positions per revolution, corresponding to an angular resolution of approximately 0.0055°. At a link length of 176.79 mm, this angular resolution translates to a positional uncertainty of approximately 0.17 mm at the end-effector for a single joint, which compounds across three joints.

Human error during point capture is a significant factor for triangle and circle measurements, as these require the user to manually position the probe tip at specific geometric locations. Slight misplacement of even one point introduces errors that propagate through the geometric calculation, explaining why triangle and circle measurements show higher variability than line measurements.

Communication latency in the RS485 Modbus RTU bus introduces small temporal offsets between encoder readings when the arm is in motion. For static measurements this effect is negligible, but for dynamic tracking applications it may become significant.

Finally, mechanical backlash in the 3D-printed gear and coupling interfaces introduces hysteresis: the encoder reading when approaching a position from one direction may differ slightly from the reading when approaching from the opposite direction. This was partially mitigated by always approaching measurement positions from a consistent direction during testing.

## 4.8 Summary

The experimental results demonstrate that the developed coordinate measuring arm is capable of performing accurate and reliable measurements. The system shows particularly strong performance in line and circle measurements, while triangle measurements are slightly more sensitive to error.

The integration of real-time visualization and measurement functions enhances usability and provides a practical and cost-effective solution for coordinate measurement.

# CHAPTER 5: CONCLUSION AND RECOMMENDATIONS

## 5.1 Conclusion

This study successfully developed a low-cost coordinate measuring arm using 3D-printed mechanical components, rotary encoders, and a Python-based software system. The primary objective of creating an affordable and functional alternative to conventional coordinate measuring machines (CMMs) was achieved.

The system integrates mechanical design, electronic sensing, and software processing into a unified platform. The articulated arm, designed using Autodesk Inventor and fabricated through 3D printing, provides sufficient flexibility for performing measurement tasks within a defined

workspace. The use of absolute rotary encoders and RS485 Modbus communication enabled reliable acquisition of joint angle data.

A custom graphical user interface (GUI) was developed to control the system, visualize measurements, and perform geometric calculations. Forward kinematics was implemented to convert joint angles into Cartesian coordinates, allowing accurate spatial measurement. In addition, real-time monitoring and three-dimensional visualization of the arm, including the end-effector (tip), were successfully implemented, improving usability and system understanding.

Experimental results demonstrate that the system is capable of performing measurements with acceptable accuracy. Line measurements achieved an average error within approximately **±2%**, while triangle and circle measurements showed reasonable accuracy depending on point selection and measurement conditions. The implementation of calibration techniques, including offset calibration and global scale correction, significantly improved measurement reliability.

Although the current system operates primarily in a planar configuration with limited degrees of freedom, the results confirm that a coordinate measuring system can be developed using accessible and low-cost components while maintaining practical performance.

Furthermore, this study establishes a foundation for future development toward more advanced systems. By extending the design to include additional joints and integrating a high-precision probe, the system can evolve into a full six-degree-of-freedom articulated measuring arm. Such an enhancement would enable full three-dimensional measurement and allow automatic data acquisition. With these improvements, it would be possible to generate digital geometric representations of measured objects, which can be further developed into CAD models for engineering applications.

Overall, the developed system demonstrates the feasibility of combining mechanical design, sensor technology, and software engineering to create a functional and scalable coordinate measuring solution.

# 5.2 Recommendations for Future Work

Based on the outcomes and limitations identified in this study, the following improvements are recommended for future development of the system:

Replacement of 3D-printed structural links with aluminium or carbon-fibre alternatives would significantly reduce deflection under load and improve positional accuracy. A six-degree-of-freedom configuration, achieved by adding three additional joints, would extend the system from planar to full three-dimensional measurement capability.

Integration of a physical contact probe with force feedback would eliminate user-dependent point-capture error, which was identified as a major contributor to geometric measurement variability. Wireless communication via Bluetooth or Wi-Fi would remove the tethered USB-RS485 connection and simplify deployment in workshop environments.

From a software perspective, implementation of a Denavit–Hartenberg (D-H) parameter model would improve the generalisability and precision of the kinematic calculations, particularly for future multi-axis extensions. Addition of an automated calibration routine, guided by the GUI, would reduce setup time and improve reproducibility across measurement sessions.

Finally, integration with CAD software via a standardised point-cloud output format would enable the system to contribute directly to reverse engineering workflows, realising the broader application potential described in the introduction.

# REFERENCES

[1] H. Schwenke, W. Knapp, H. Haitjema, A. Weckenmann, R. Schmitt, and F. Delbressine, “Geometric error measurement and compensation of machines – an update,” CIRP Annals, vol. 57, no. 2, pp. 660–675, 2008.

[2] A. Weckenmann, T. Estler, G. Peggs, and D. McMurtry, “Probing systems in dimensional metrology,” CIRP Annals, vol. 53, no. 2, pp. 657–684, 2004.

[3] Hexagon Manufacturing Intelligence, “ROMER Absolute Arm – Portable Coordinate Measuring Arm,” 2022. [Online]. Available: https://www.hexagonmi.com/products/portable-measuring-arms

[4] J. J. Craig, *Introduction to Robotics: Mechanics and Control*, 3rd ed. Upper Saddle River: Pearson Prentice Hall, 2005.

[5] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. London: Springer, 2009.

[6] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*, 2nd ed. Hoboken: John Wiley and Sons, 2020.

[7] L. Cui, S. Wang, H. Wang, and J. Tan, “A review of absolute encoders for position measurement,” IEEE Sensors Journal, vol. 20, no. 4, pp. 1739–1754, 2020.

[8] J. Fraden, *Handbook of Modern Sensors: Physics, Designs, and Applications*, 5th ed. Cham: Springer, 2016.

[9] Modbus Organization, “Modbus Application Protocol Specification V1.1b3,” 2012. [Online]. Available: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

[10] Modbus Organization, “Modbus over Serial Line Specification and Implementation Guide V1.02,” 2006. [Online]. Available: https://modbus.org/docs/Modbus_over_serial_line_V1_02.pdf

[11] Telecommunications Industry Association, “TIA-485-A: Electrical Characteristics of Generators and Receivers for Use in Balanced Digital Multipoint Systems,” 2003.

[12] pymodbus contributors, “pymodbus: A Python Modbus Stack,” 2023. [Online]. Available: https://pymodbus.readthedocs.io

[13] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley: CreateSpace, 2009.

[14] Python Software Foundation, “tkinter — Python interface to Tcl/Tk,” 2024. [Online]. Available: https://docs.python.org/3/library/tkinter.html

[15] J. D. Hunter, “Matplotlib: A 2D graphics environment,” Computing in Science and Engineering, vol. 9, no. 3, pp. 90–95, 2007.

[16] Autodesk Inc., “Autodesk Inventor Professional 2024 User Guide,” 2024. [Online]. Available: https://knowledge.autodesk.com/support/inventor

[17] I. Gibson, D. Rosen, B. Stucker, and M. Khorasani, *Additive Manufacturing Technologies: 3D Printing, Rapid Prototyping and Direct Digital Manufacturing*, 3rd ed. Cham: Springer, 2021.

[18] B. N. Turner, R. Strong, and S. A. Gold, “A review of melt extrusion additive manufacturing processes: I. Process design and modeling,” Rapid Prototyping Journal, vol. 20, no. 3, pp. 192–204, 2014.

[19] J. Denavit and R. S. Hartenberg, “A kinematic notation for lower-pair mechanisms based on matrices,” Journal of Applied Mechanics, vol. 22, no. 2, pp. 215–221, 1955.

[20] E. W. Weisstein, “Circle,” MathWorld – A Wolfram Web Resource, 2024. [Online]. Available: https://mathworld.wolfram.com/Circle.html

[21] International Organization for Standardization, “ISO 10360-1: Geometrical product specifications (GPS) – Acceptance and reverification tests for coordinate measuring machines (CMM),” 2000.

[22] Joint Committee for Guides in Metrology, “Evaluation of measurement data – Guide to the expression of uncertainty in measurement (GUM),” 2008.

[23] GitHub Inc., “GitHub Copilot – AI Pair Programmer,” 2024. [Online]. Available: https://github.com/features/copilot

[24] https://briterencoder.com/product/absolute-rotary-encoder-with-rs485-modbus-rtu-communication

## APPENDIXES

Due to the large size of the full software, only the most important functional code sections are included in this appendix. These sections represent the main communication, calibration, kinematic calculation, measurement, and visualization functions of the developed system.

# APPENDIX A – Key Python Source Code

```
CONNECTION = {
    'port': 'COM4',
    'baudrate': 9600,
    'slave_id': 1,
    'timeout': 1.0,
    'data_bits': 8,
    'stop_bits': 1,
    'parity': 'N',
}


REGISTER = {
    'address': 0,
    'count': 1,
}


ANGLE = {
    'bits': 16,
    'max_value': 65536,
    'degrees': 360,
}


def calculate_angle(raw_value):
    """Convert raw encoder value to angle in degrees."""
    return (raw_value / ANGLE['max_value']) * ANGLE['degrees']
```

# A.2 Modbus Encoder Reading

```
def read_modbus_register(self, slave_id, address=None):
    if address is None:
        address = self.register_addr_var.get()

    for keyword in ["device_id", "slave"]:
        try:
            kwargs = {
                "address": address,
                "count": 1,
                keyword: slave_id
            }

            result = self.client.read_holding_registers(**kwargs)

            if result is None:
                continue

            if result.isError():
                continue

            if not result.registers:
                continue

            return result.registers[0]

        except TypeError:
            continue

        except Exception:
```

```
            continue

    return None
```

# A.3 Angle Offset and Direction Correction

```
def convert_to_angle(self, value):
    return (value / 65536.0) * 360.0


def normalize_angle(self, angle):
    return ((angle + 180.0) % 360.0) - 180.0


def apply_offsets(self, raw_angle, offset, direction):
    corrected = self.normalize_angle(raw_angle - offset)
    return corrected * direction
```

# A.4 Forward Kinematics

```
def forward_kinematics(self, t1, t2, t3):
    L1, L2, L3 = self.LINK_LENGTHS

    x = (
        L1 * math.cos(t1)
        + L2 * math.cos(t1 + t2)
        + L3 * math.cos(t1 + t2 + t3)
    )

    y = (
        L1 * math.sin(t1)
        + L2 * math.sin(t1 + t2)
        + L3 * math.sin(t1 + t2 + t3)
    )
```

```
    return x * self.scale, y * self.scale, 0.0
```

# A.5 Real-Time Coordinate Update

```
def start_realtime_coordinates(self):
    if not self.is_connected:
        messagebox.showerror("Error", "Connect to the device first.")
        return

    if self.realtime_running:
        return

    self.realtime_running = True
    self.result_display.set("Real-time tracking started")
    self._update_realtime_coordinates()

def _update_realtime_coordinates(self):
    if not self.realtime_running:
        return

    try:
        angles = self.read_all_encoders()

        if angles is not None:
            t1, t2, t3 = [math.radians(a) for a in angles]
            x, y, z = self.forward_kinematics(t1, t2, t3)

            self._update_display(angles, x, y, z)
            self.update_plot()

        else:
            self.result_display.set("Error reading encoders")
```

```
    except Exception as exc:
        self.result_display.set(f"Real-time error: {exc}")

    finally:
        if self.realtime_running:
            self.root.after(100, self._update_realtime_coordinates)
```

# A.6 Global Scale Calibration

```
def _capture_scale_point2_apply(self):
    if not self.scale_points:
        messagebox.showerror("Error", "Capture scale point 1 first.")
        return

    pos2 = self._capture_position_point()
    pos1 = self.scale_points[0]

    measured = math.hypot(
        pos2[0] - pos1[0],
        pos2[1] - pos1[1]
    )

    actual = self.actual_length_var.get()

    if actual <= 0 or measured <= 0:
        messagebox.showerror("Error", "Invalid calibration distance.")
        return

    self.scale = actual / measured
    self.scale_display.set(f"Scale: {self.scale:.4f}")
    self.scale_points = []
```

# A.7 Shape Measurement Functions

## LINE

```
def measure_line(self):
    p1, p2 = self.points

    dist = math.hypot(
        p2[0] - p1[0],
        p2[1] - p1[1]
    )

    self.result_display.set(f"Distance AB = {dist:.2f} mm")
```

## TRIANGLE

```
def measure_triangle(self):
    A, B, C = self.points

    AB = math.hypot(B[0] - A[0], B[1] - A[1])
    BC = math.hypot(C[0] - B[0], C[1] - B[1])
    CA = math.hypot(A[0] - C[0], A[1] - C[1])

    cos_A = (AB**2 + CA**2 - BC**2) / (2.0 * AB * CA)
    cos_B = (AB**2 + BC**2 - CA**2) / (2.0 * AB * BC)

    cos_A = max(-1.0, min(1.0, cos_A))
    cos_B = max(-1.0, min(1.0, cos_B))

    angle_A = math.degrees(math.acos(cos_A))
    angle_B = math.degrees(math.acos(cos_B))
    angle_C = 180.0 - angle_A - angle_B
```

```
    self.result_display.set(
        f"AB={AB:.2f} mm, BC={BC:.2f} mm, CA={CA:.2f} mm | "
        f"A={angle_A:.1f}°, B={angle_B:.1f}°, C={angle_C:.1f}°"
    )
```

## CIRCLE

```
def fit_circle(self):
    x1, y1 = self.points[0]
    x2, y2 = self.points[1]
    x3, y3 = self.points[2]

    d = 2.0 * (
        x1 * (y2 - y3)
        + x2 * (y3 - y1)
        + x3 * (y1 - y2)
    )

    if abs(d) < 1e-6:
        self.result_display.set("Error: collinear points")
        return

    sq1 = x1**2 + y1**2
    sq2 = x2**2 + y2**2
    sq3 = x3**2 + y3**2

    h = (sq1*(y2-y3) + sq2*(y3-y1) + sq3*(y1-y2)) / d
    k = (sq1*(x3-x2) + sq2*(x1-x3) + sq3*(x2-x1)) / d
    r = math.hypot(x1 - h, y1 - k)

    self.result_display.set(
        f"Circle center=({h:.2f}, {k:.2f}), r={r:.2f} mm"
```

```
    )
```

## A.8 3D Arm Visualization With Tip

```
def _draw_sphere(self):
    self.sphere_ax.clear()

    t1, t2, t3 = [math.radians(a) for a in self.current_angles]
    L1, L2, L3 = self.LINK_LENGTHS

    x1, y1, z1 = 0.0, 0.0, 0.0

    x2 = L1 * math.cos(t1) * self.scale
    y2 = L1 * math.sin(t1) * self.scale
    z2 = 0.0

    x3 = (L1 * math.cos(t1) + L2 * math.cos(t1 + t2)) * self.scale
    y3 = (L1 * math.sin(t1) + L2 * math.sin(t1 + t2)) * self.scale
    z3 = 0.0

    x4 = (
        L1 * math.cos(t1)
        + L2 * math.cos(t1 + t2)
        + L3 * math.cos(t1 + t2 + t3)
    ) * self.scale

    y4 = (
        L1 * math.sin(t1)
        + L2 * math.sin(t1 + t2)
        + L3 * math.sin(t1 + t2 + t3)
    ) * self.scale
```

```
    z4 = 0.0

    points = [(x1, y1, z1), (x2, y2, z2), (x3, y3, z3), (x4, y4, z4)]
    labels = ["1: Base/J1", "2: J2", "3: J3", "4: Tip"]

    xs = [p[0] for p in points]
    ys = [p[1] for p in points]
    zs = [p[2] for p in points]

    self.sphere_ax.plot(xs, ys, zs, linewidth=3, marker="o")

    for i, (x, y, z) in enumerate(points):
        self.sphere_ax.text(x, y, z, labels[i])

    self.sphere_ax.set_title("3D Arm Visualization with Tip Position")
    self.sphere_ax.set_xlabel("X")
    self.sphere_ax.set_ylabel("Y")
    self.sphere_ax.set_zlabel("Z")

    self.sphere_canvas.draw()
```

Forward Kinematics Equations:

$$X = L1 \cos(\theta 1) + L2 \cos(\theta 1 + \theta 2) + L3 \cos(\theta 1 + \theta 2 + \theta 3) \quad (1)$$

$$Y = L1 \sin(\theta 1) + L2 \sin(\theta 1 + \theta 2) + L3 \sin(\theta 1 + \theta 2 + \theta 3) \quad (2)$$