

Université de technologie de Compiègne

**Study of a 3D Object Position and Orientation Detection System with
Embedded AI Processing**

Master ISCF - Project Workshop

Project Supervisors: Erwan Dupont, Hani Al Hajjar, Laurent Petit

Team Members:

Andri Halili

Syed Muhammad Abubakar

Housseem Abbes

Wang Biyao

Youssef el Gouri

Compiègne, 13 January 2026

Abstract

This project presents a 6-degree-of-freedom (6DOF) pose estimation system for 3D objects using LiDAR-based point cloud acquisition and embedded deep learning. Conducted at UTC's Roberval Laboratory, the work addresses real-time object localization with resource-constrained embedded intelligence. The system employs a YDLidar G2 scanner mounted vertically with a conveyor belt mechanism to capture successive 2D vertical slices at 1cm intervals, reconstructing a 2.5D-to-3D point cloud. A PyTorch-based deep learning regression model predicts six pose parameters: translational coordinates (x , y , z) and rotational angles (roll, pitch, yaw). The model is trained on real and synthetically augmented point cloud data and optimized for deployment on an NVIDIA Jetson Orin Nano platform.

Data acquisition is managed through ROS2 integration for seamless hardware-software communication. The project initially explored MEMS mirror-based laser triangulation but pivoted to the LiDAR solution due to hardware constraints. Key contributions include a synthetic data generation pipeline, vertical scanning methodology for 3D reconstruction, and embedded AI processing for real-time inference.

The system demonstrates the feasibility of combining low-cost scanning hardware with modern deep learning for applications in quality control, robotic manipulation, and autonomous systems.

Keywords: 6D pose estimation, point cloud processing, embedded AI, LiDAR scanning, deep learning, edge computing, NVIDIA Jetson, ROS2

Table of Contents

Abstract	2
Table of Contents	3
Introduction	5
1.1 Project Context and Motivation	5
1.2 Problem Statement	5
1.3 Objectives	6
State of the Art	7
2.1 3D Object Detection and Scanning Mode	7
2.1.1 Laser Triangulation Principles	7
2.1.2 LiDAR-Based Scanning Techniques	8
2.2 6D Pose Estimation Deep Learning Approaches	9
2.1.1 Multilayer Perceptrons with Engineered Features	9
2.1.2 PointNet	10
2.1.3 PointNet++	11
2.1.4 Other Relevant Techniques	12
2.3 Embedded AI for Real-Time Processing	13
2.3.1 TinyML and Edge AI Overview	14
2.4 Positioning of Our Approach	15
System Architecture & Methodology	17
3.1 Overall System Design	17
3.2 Hardware Components	19
3.2.3 Line Laser and Camera Setup	23
3.2.4 NVIDIA Jetson Orin Nano	25
3.3 Initial Approach with MEMS Mirror	26
LiDAR Scanning Process	27
4.1 LiDAR Scanning Process	27
4.1.1 Vertical Slice Method	27
4.1.2 ROS2 Integration	28
4.2 Point Cloud Generation & Simulation	30
4.2.1 Real Point Cloud Acquisition	30
4.2.2 Synthetic Data Generation	31
4.2.3 Dataset Preparation	32
AI Model Development	35
5.1 Model Architecture Selection	35

	4
5.2 Training Process	38
5.3 Model Optimization for Embedded Deployment	42
Experimental Setup & Integration	45
6.1 Physical System Assembly	45
6.2 System Integration Status	46
Results & Discussion	48
7.1 AI Model Performance	48
7.2 System Performance	49
7.3 Final Project Visualization	50
Conclusions & Future Work	53
8.1 Project Summary	53
8.2 Future Improvements	54
8.3 Broader Applications	55
8.4 Closing Remarks	55
References	56

Introduction

1.1 Project Context and Motivation

In modern industrial and robotic systems, accurate 6DOF pose estimation, determining an object's position and orientation in 3D space, is essential for automated assembly, quality control, and autonomous manipulation. Traditional approaches rely on centralized processing systems, introducing latency and bandwidth constraints that limit real-time performance. The emergence of embedded artificial intelligence, particularly TinyML and Edge AI, enables the deployment of machine learning algorithms directly on resource-constrained devices at the sensor edge, eliminating communication delays while enhancing system robustness and efficiency. This project explores the integration of LiDAR-based 3D scanning with embedded deep learning for real-time pose estimation, addressing the challenge of bringing intelligent processing capabilities to mechatronic measurement systems.

1.2 Problem Statement

Implementing accurate 6DOF pose estimation on embedded platforms presents significant technical challenges. Point cloud data from LiDAR sensors is inherently sparse, noisy, and varies in density depending on scanning geometry. Deep learning models capable of processing such data typically require substantial computational resources, while embedded platforms like the NVIDIA Jetson Orin Nano offer limited processing power and memory compared to cloud-based systems. The core challenge lies in balancing model accuracy with real-time inference constraints while maintaining robustness to imperfect sensor data. Additionally, the scarcity of labeled 3D training data necessitates effective synthetic data generation strategies to achieve reliable pose prediction.

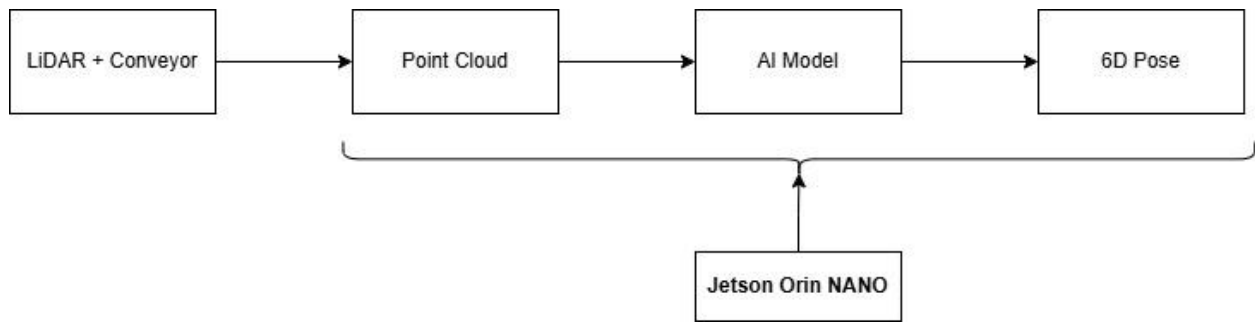


Figure 1.1 System Overview Diagram

1.3 Objectives

The primary objective of this project is to develop a functional 6DOF pose estimation system integrating LiDAR-based point cloud acquisition with embedded deep learning processing. Specific objectives include:

- Design and implement a 3D scanning system using a vertical LiDAR configuration and conveyor mechanism to reconstruct object point clouds
- Generate synthetic training data to augment real sensor measurements and improve model generalization
- Develop and train a deep learning regression model to predict translational and rotational pose parameters from point cloud inputs
- Optimize the trained model for deployment on the NVIDIA Jetson Orin Nano embedded platform
- Integrate hardware components through ROS2 for coordinated data acquisition and processing
- Validate system performance through experimental testing with known object poses

State of the Art

This section reviews the existing literature relevant to 3D object pose estimation using point cloud data and embedded artificial intelligence. The review is organized into four main areas: first, an overview of 3D scanning and point cloud acquisition methods; second, a survey of traditional and deep learning approaches for 6DOF pose estimation; third, an examination of embedded AI platforms and optimization techniques; and finally, a positioning of the present work within the current state of research. This review establishes the technical foundation for the methodological choices made in this project.

2.1 3D Object Detection and Scanning Mode

Three-dimensional object detection requires accurate acquisition of spatial geometry, typically achieved through various scanning technologies. Among these, optical methods based on structured light, stereo vision, and laser-based techniques have become prevalent in industrial and research applications due to their non-contact nature and high precision capabilities.

2.1.1 Laser Triangulation Principles

Laser triangulation is a well-established optical measurement technique based on geometric principles. The method projects a laser line or point onto an object surface, while a camera positioned at a known baseline distance captures the reflected light. By analyzing the displacement of the laser projection in the camera image, the distance to surface points can be calculated using triangulation geometry [1]. This technique offers millimeter-level accuracy and is widely used in quality inspection, reverse engineering, and dimensional measurement systems [2]. The primary advantages include high precision, simple calibration procedures, and cost-effectiveness. However, limitations arise with highly reflective or absorptive surfaces, and complete 3D geometry capture requires either mechanical movement of the laser-camera assembly or object manipulation.

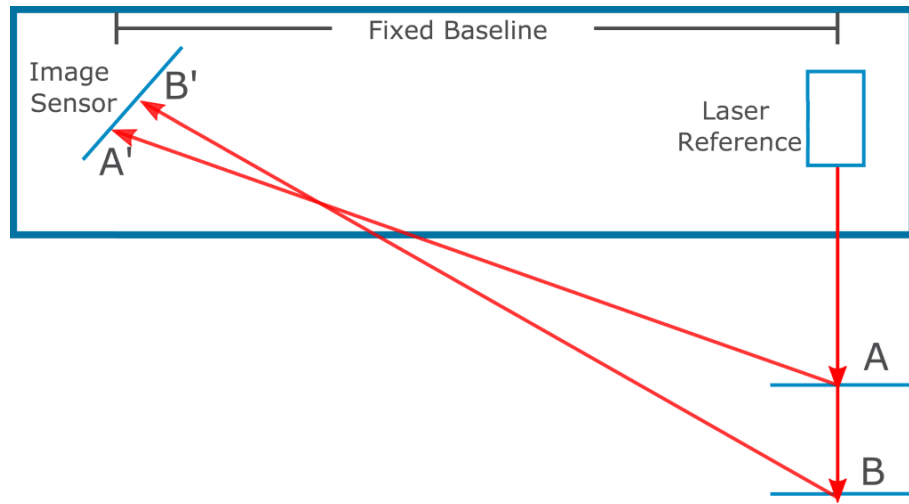


Figure 2.1 Laser Triangulation Method

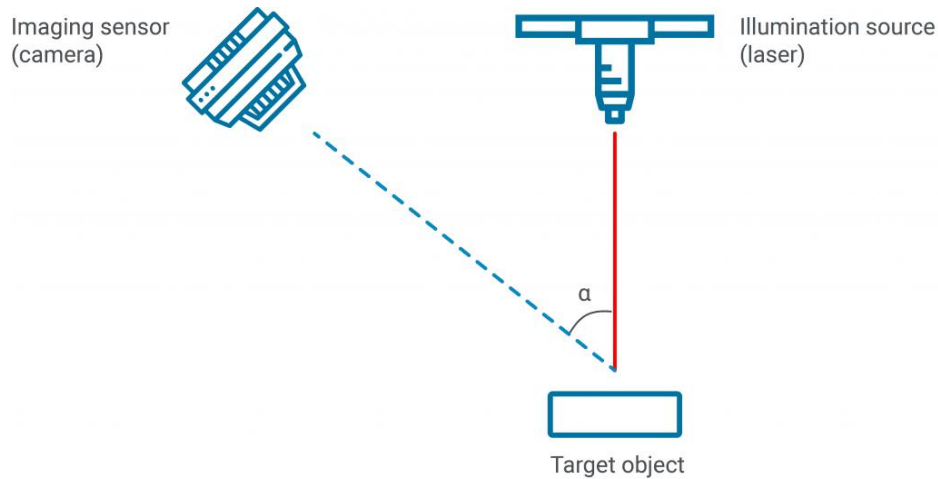


Figure 2.2 Typical 3D Scanner Setup

2.1.2 LiDAR-Based Scanning Techniques

Light Detection and Ranging (LiDAR) technology extends laser-based measurement principles to enable rapid acquisition of dense point clouds. LiDAR systems emit laser pulses and measure time-of-flight or phase shift of returned signals to determine distances. Commercial LiDAR sensors are classified into 2D and 3D configurations [3]. Two-dimensional LiDAR sensors perform planar scanning by rotating a single laser beam to

capture distance measurements along a circular arc. While 2D LiDAR units provide only planar slices, they can be adapted for 3D reconstruction through mechanical scanning strategies. One effective approach involves mounting the sensor in a fixed orientation and translating the target object linearly, capturing successive parallel slices that are subsequently stacked to form volumetric point cloud representations [4]. This slice-stacking method offers a cost-effective alternative to expensive 3D LiDAR systems while maintaining adequate spatial resolution for pose estimation applications.

2.2 6D Pose Estimation Deep Learning Approaches

Deep learning has become the dominant paradigm for 6DOF pose estimation from 3D point cloud data. The literature presents a spectrum of approaches ranging from sophisticated architectures that process raw point clouds directly to simpler methods that combine handcrafted feature extraction with standard neural network regressors. This subsection reviews the principal deep learning approaches relevant to point cloud-based pose estimation, examining their theoretical foundations, architectural designs, and relative strengths and limitations.

2.1.1 Multilayer Perceptrons with Engineered Features

Prior to end-to-end deep learning, 3D pose estimation relied on geometric algorithms and handcrafted features. Iterative Closest Point (ICP) iteratively minimizes distances between point clouds to align objects, requiring good initial alignment and suffering from local minima [5]. Feature-based methods extract descriptors such as Fast Point Feature Histograms (FPFH) or surface normals for matching between template and scene [6].

While deterministic and requiring no training data, these approaches struggle with real-time performance on embedded hardware and are sensitive to noise and partial occlusions.

Feature-based methods combined with multilayer perceptrons (MLPs) offer a middle ground suited to constrained deployment. Handcrafted statistics such as minimum, maximum, mean, and standard deviation computed from spatial regions drastically reduce dimensionality while capturing essential geometric properties. For structured scanning processes like vertical slicing, region-based features provide fixed-length representations regardless of point count, eliminating variable input size issues. MLPs trained on these features require modest training data since domain knowledge is pre-encoded, execute efficiently through simple matrix operations, and provide interpretable inputs for debugging. The primary limitation is reduced generalizability, as fixed features may miss subtle variations that learned representations capture. Nevertheless, for known objects under controlled conditions with strict computational constraints, engineered features with MLP regression provide an effective and pragmatic solution.

2.1.2 PointNet

PointNet addresses the fundamental challenge of processing unordered point sets, which lack the regular grid structure of images. Point clouds are inherently orderless, as the same 3D points can be presented in any permutation without altering geometry [7].

PointNet achieves permutation invariance through a symmetric architecture that processes each point independently via shared multilayer perceptrons, producing per-point features. These features are aggregated using max-pooling, selecting maximum values across all points for each dimension to produce a fixed-length global descriptor. For pose estimation, this global feature passes through fully connected layers regressing the six pose parameters. An optional spatial transformer network can learn canonical alignments, improving robustness to rotations.

PointNet's key innovation demonstrated that symmetric functions enable effective unordered set processing, establishing a paradigm balancing local feature extraction with global reasoning. The architecture handles variable point counts without modification, providing flexibility for applications with varying scan density. However, global max-pooling discards local geometric structure information, as each point contributes independently without capturing neighboring relationships. This sacrifices the ability to model fine details such as edges, corners, and surface curvature, potentially impacting accuracy when such features are discriminative for pose estimation. Additionally, training PointNet from scratch requires substantial labeled data to learn effective representations.

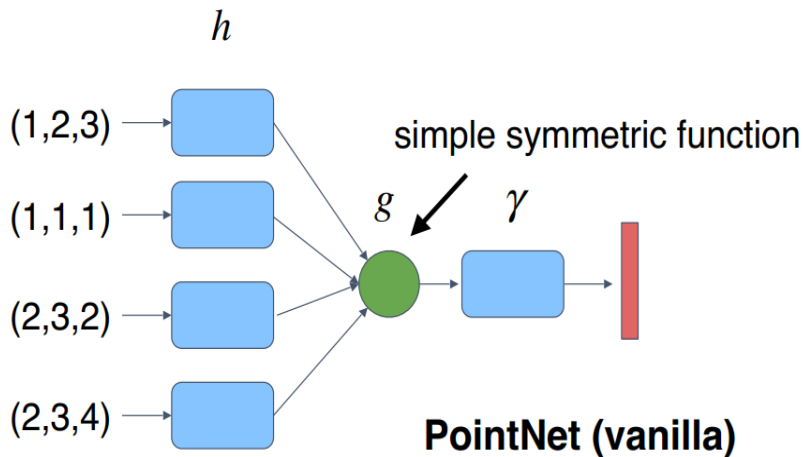


Figure 2.1 PointNet simple architecture

2.1.3 PointNet++

PointNet++ extends the original architecture to capture local geometric structures through hierarchical feature learning. The key innovation is the set abstraction layer, which applies PointNet locally within neighborhoods of points rather than globally across the entire point cloud. This hierarchical approach enables the network to learn features at multiple spatial scales, from fine local details to coarse global structure [8].

For pose estimation, PointNet++ offers improved accuracy compared to PointNet, particularly for objects with distinctive local features. The ability to capture edges, corners, and surface variations enables more precise pose discrimination. The multi-scale feature hierarchy provides robustness to partial occlusions, as local features from visible regions can still inform the pose prediction even when parts of the object are missing. These properties align well with the characteristics of LiDAR-scanned point clouds, where surface details and edge responses carry important geometric information.

The enhanced capabilities of PointNet++ come at the cost of increased computational complexity. The neighborhood search operations required for grouping add overhead compared to PointNet's purely pointwise processing. The number of hierarchical levels, neighborhood sizes, and feature dimensions must be carefully tuned to balance accuracy against inference time. For embedded deployment on resource-constrained platforms, these considerations motivate either aggressive model compression or alternative approaches that achieve comparable results with lower complexity.

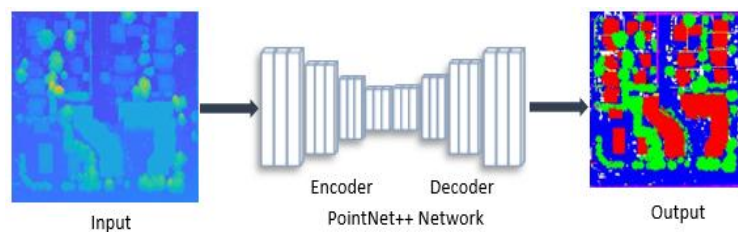


Figure 2.2 Pointnet++ Architecture

2.1.4 Other Relevant Techniques

Beyond the approaches detailed above, several other techniques provide relevant context. Dynamic Graph Convolutional Neural Networks (DGCNN) extend point cloud processing through graph neural networks, constructing k-nearest neighbor graphs in feature space to learn edge features encoding relationships between neighboring points [9]. While achieving

strong performance on shape analysis tasks, the dynamic graph construction overhead presents challenges for real-time embedded deployment. Keypoint-based methods decompose pose estimation into predicting correspondences between observed points and predefined object keypoints, followed by geometric solving using Perspective-n-Point (PnP) or singular value decomposition algorithms. PVNet exemplifies this approach, predicting direction vectors toward object keypoints and recovering pose through RANSAC-based voting, providing robustness to outliers and partial visibility [10]. Similar concepts can be applied to point cloud data by predicting offsets from observed points to canonical keypoints on CAD models. However, these advanced architectures require substantial computational resources and training data, reinforcing the practical advantages of feature-based approaches with compact neural networks for resource-constrained embedded applications.

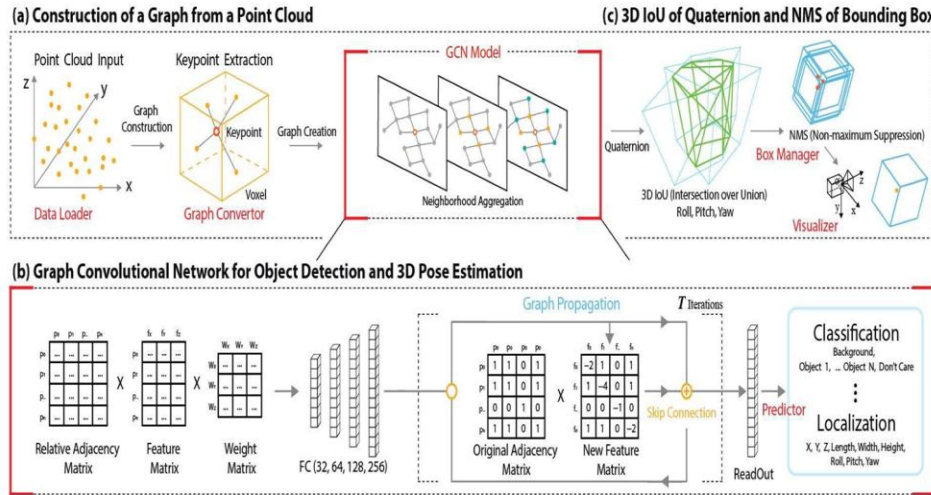


Figure 2.3 Architecture of GNN for pose estimation

2.3 Embedded AI for Real-Time Processing

The deployment of deep learning models on resource-constrained embedded devices represents a critical challenge for real-time applications. While cloud-based inference offers virtually unlimited computational resources, edge computing solutions eliminate network

latency, reduce bandwidth requirements, and enhance data privacy. This subsection examines the embedded AI landscape, focusing on hardware platforms and optimization techniques relevant to real-time pose estimation systems.

2.3.1 TinyML and Edge AI Overview

TinyML refers to machine learning deployment on ultra-low-power microcontrollers operating within milliwatt power budgets, enabling intelligent processing directly at the sensor level [11]. Edge AI encompasses a broader category including more capable embedded processors such as GPUs and neural processing units that execute complex deep learning models locally while maintaining real-time performance. The transition from cloud-based to edge-based inference is driven by practical requirements: latency-critical applications in robotics and autonomous systems cannot tolerate cloud communication delays, bandwidth limitations make continuous high-dimensional sensor data streaming impractical, privacy considerations favor local processing, and edge deployment enables operation in environments with unreliable network connectivity. For pose estimation in industrial and robotic applications, these factors strongly favor embedded solutions processing sensor data locally.

Deploying deep learning models on embedded platforms requires model optimization techniques. Quantization reduces numerical precision from 32-bit floating point to 8-bit integers, decreasing memory footprint and accelerating inference on integer arithmetic hardware [12]. Pruning removes redundant weights contributing minimally to accuracy, reducing computational requirements. Knowledge distillation trains compact student networks to mimic larger teacher models, transferring representational capacity to embedded-suitable architectures. Framework-specific optimizations such as NVIDIA TensorRT provide automated graph optimization, layer fusion, and precision calibration for target hardware platforms.

2.3.2 Hardware Platforms for Embedded AI

The embedded AI hardware landscape spans from microcontroller-based systems to GPU-equipped platforms, with selection driven by computational demands, power constraints, and application requirements. Microcontroller units such as ARM Cortex-M series provide basic inference using TensorFlow Lite Micro but remain limited to simple tasks due to restricted memory and processing power [13]. These platforms are unsuitable for point cloud processing requiring substantial memory for 3D data. AI accelerators like Google's Coral Edge TPU (4 TOPS, 2 watts) and Intel's Neural Compute Stick 2 offer efficient neural network execution [14]. While excelling at image tasks, they lack flexibility for custom point cloud pipelines and general-purpose computation.

GPU-equipped embedded platforms represent the most capable solution for complex deep learning workloads. The NVIDIA Jetson family, with the Jetson Orin Nano employed in this project, offers optimal performance-power balance. Native CUDA, cuDNN, and TensorRT support enables GPU-accelerated computation and inference optimization, while ROS2 integration facilitates robotic system incorporation. PyTorch models convert to TensorRT engines with FP16 and INT8 precision modes. GPIO interfaces enable direct sensor integration, making the Jetson family well-suited for mechatronic applications requiring embedded intelligence and real-time geometric data processing.

2.4 Positioning of Our Approach

The literature review reveals that while substantial progress has been made in 6D pose estimation and embedded AI independently, the integration of cost-effective LiDAR-based scanning with deep learning on resource-constrained platforms remains underexplored. This project addresses this gap through a unique combination of vertical 2D LiDAR scanning

with conveyor-based translation, deep learning regression for pose prediction, and deployment targeting the NVIDIA Jetson Orin Nano platform.

In contrast to RGB-D approaches such as DenseFusion that require synchronized color and depth cameras, our system relies exclusively on geometric point cloud data from an affordable 2D LiDAR sensor, simplifying hardware requirements and reducing cost. While traditional methods like ICP offer deterministic solutions, they lack the speed necessary for real-time embedded applications; conversely, our deep learning approach provides rapid inference once optimized for edge deployment. A critical innovation lies in addressing training data scarcity through synthetic point cloud generation and augmentation, enabling model training without extensive real-world data collection. The vertical slice-stacking methodology transforms a 2D LiDAR into an effective 3D scanner, demonstrating that accurate pose estimation can be achieved within strict computational and budgetary constraints relevant to industrial applications.

System Architecture & Methodology

3.1 Overall System Design

This project develops a compact embedded mechatronic system for real-time 6D pose estimation, integrating LiDAR scanning with on-device computation. The system architecture comprises four main functional subsystems operating within a deterministic data processing pipeline.

System Architecture

The sensing and scanning subsystem acquire three-dimensional object information through a vertically mounted LiDAR sensor that captures depth measurements as the object translates through the sensing area via a conveyor mechanism. This configuration reconstructs dense 3D point clouds by aggregating successive vertical scans, eliminating mechanically complex rotating components while providing reliable and repeatable acquisition. A line laser structured light source enhances geometric feature visibility to support accurate spatial reconstruction when required.

The embedded computing subsystem centers on the NVIDIA Jetson Orin Nano, selected after evaluating alternatives including Raspberry Pi 5, Arduino, and Jetson Xavier NX. The Jetson Orin Nano provides an Ampere architecture GPU supporting CUDA and TensorRT acceleration for real-time camera input processing and AI inference, alongside sufficient interfaces (CSI, USB 3.2, UART, I²C) for sensor and laser control system integration. All computation executes locally on the embedded platform, eliminating cloud dependency and communication latency to satisfy real-time constraints.

The artificial intelligence and pose estimation module implements the core methodology through a deep learning regression model trained to predict full 6DOF pose parameters, including translational and rotational components, directly from reconstructed point cloud inputs.

System integration and data management leverage ROS2 to enable modular communication between hardware drivers, processing nodes, and inference modules. An e-con Jetson camera provides native Jetson compatibility and low-latency data transfer for the imaging subsystem.

Data Flow Pipeline

The system follows a deterministic processing sequence: the LiDAR acquires depth measurements as the object advances along the conveyor; successive scans combine to reconstruct a 3D point cloud; the point cloud transmits to the AI model executing on the Jetson Orin Nano; the deep learning model infers 6DOF pose in real time; and estimated pose outputs feed visualization, logging, or downstream robotic applications.

Mechanical Architecture

The mechanical design provides a stable, modular, and reconfigurable physical platform addressing the critical dependence of 3D perception accuracy on sensor positioning and structural rigidity. Design objectives include precise and repeatable alignment between LiDAR, line laser, and camera; sufficient structural rigidity to prevent vibrations and misalignment; flexible reconfiguration during calibration and experimental iterations; and rapid prototyping support through additive manufacturing.

All mechanical components were designed in CATIA, with full system-level virtual assembly performed to reduce fabrication errors and minimize physical prototyping iterations. Custom support structures for each sensing component were fabricated via 3D printing, ensuring rigid mounting on the PMMA base plate with accurate and repeatable positioning. These supports function as mechanical adapters between commercial sensor housings and standardized M6 mounting patterns, enabling integration without modifying off-the-shelf components. Additive manufacturing facilitates rapid redesign and replacement, ensuring adaptability to configuration changes throughout iterative system development.

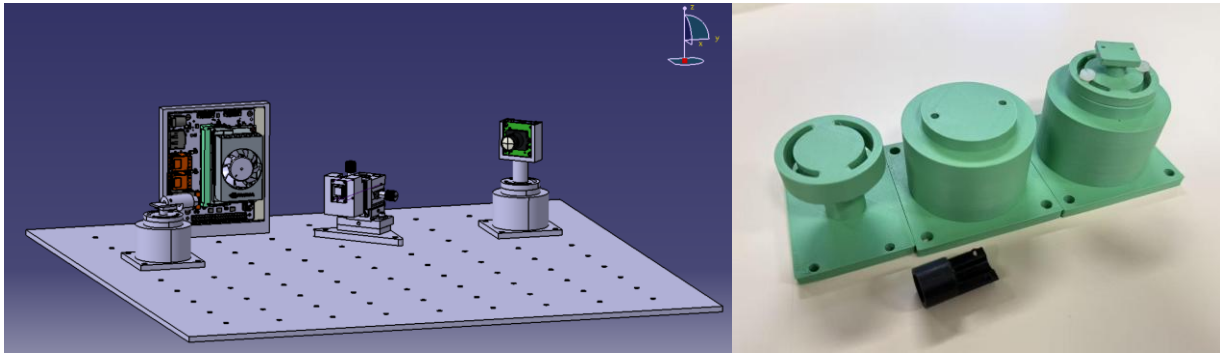


Figure 3.1 3D Printed Components and CAD

3.2 Hardware Components

3.2.1 YDLidar G2 Specifications and Setup

The YDLiDAR G2 is a compact 2D LiDAR sensor commonly used in robotics and embedded perception applications due to its low cost, lightweight design, and sufficient measurement accuracy for indoor environments.

Technical Specifications Overview

The main characteristics of the YDLiDAR G2 relevant to this project are:

- Scanning principle: Time-of-Flight (ToF)
- Scanning type: 2D planar scanning
- Angular scanning range: up to 360°
- Angular resolution: dependent on rotation speed
- Measurement range: suitable for short- to medium-range indoor scanning
- Output data: distance measurements associated with angular positions
- Interface: serial communication with the embedded processing unit

These specifications make the YDLiDAR G2 appropriate for layered scanning approaches where multiple 2D scans are combined to reconstruct a 3D representation.

Sensor Orientation and Mounting Configuration

In the proposed system, the YDLiDAR G2 is mounted vertically, rather than in its conventional horizontal configuration. This orientation allows the sensor to perform scans in a vertical plane, capturing height information of the object.

The scanning angle is predefined and limited to a specific angular sector instead of using the full 360° rotation. This constraint reduces unnecessary data acquisition and focuses the scanning process on the region of interest where the object is located.

The LiDAR remains fixed in space, while the object moves in front of it on a conveyor belt.

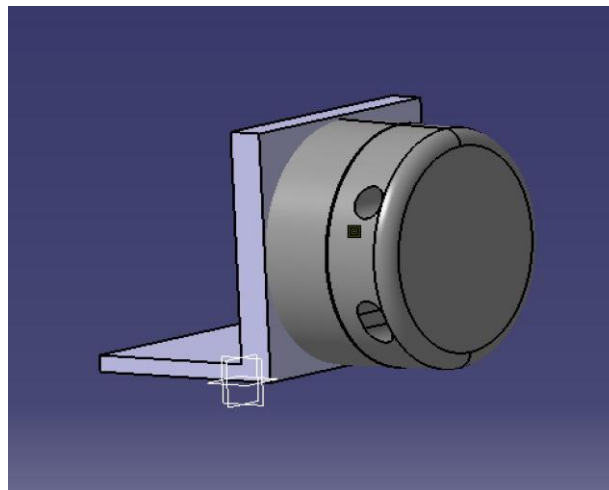


Figure 3.2 The vertical setup of the Lidar

3.2.2 Conveyor System

The conveyor system provides controlled linear translation of the target object through the LiDAR scanning plane, enabling the vertical slice-stacking methodology described in Section 4.1. The system architecture evolved through two design iterations, ultimately converging on a solution optimized for the project's functional requirements.

Initial Translation Stage Evaluation

Initial efforts focused on commissioning a Newport CONEX-CC controller integrated with an MFA-CC motorized linear stage. This ultra-compact DC servo motor system offers exceptional precision with a maximum travel range of 25 mm, minimum incremental motion of 100 nm, and position repeatability of $\pm 2.5 \mu\text{m}$. Following software installation, USB driver configuration, and successful communication validation, a complete initialization and calibration phase was performed, including homing using optical limit switches and incremental motion testing. The system demonstrated excellent performance and compliance with manufacturer specifications.

However, functional analysis revealed a critical limitation: the 25 mm travel range was insufficient for translating the LiDAR sensor across the distances required for complete object scanning. This constraint would have severely limited experimental flexibility and scanning capability. Consequently, an architectural decision was made to invert the motion strategy by maintaining the LiDAR in a fixed position and instead translating the object under inspection using a conveyor-based system. This approach provides substantially greater effective travel distance, continuous and repeatable motion, improved adaptability to various measurement scenarios, and simplified mechanical integration.

Conveyor Servo Motor Configuration

The conveyor is driven by a Lynxmotion Smart Servo (LSS) configured and controlled through the LSS Config software interface. Commissioning began by establishing USB-to-serial communication via a dedicated LSS adapter board, with the correct COM port identified through the Windows Device Manager. Following connection establishment, an automated bus scan procedure detected the servomotor by systematically querying servo identifiers (IDs 0-250) across supported baud rates. The scan confirmed successful detection without ID conflicts, indicated by a green LED status.

Firmware verification ensured compatibility and operational stability prior to system integration. Manual control testing through interactive position sliders validated movement fluidity, absence of erratic behavior, and consistency between commanded and actual mechanical response. Advanced parameter configuration included setting the servo identifier, baud rate, origin offset, angular operating range, rotation direction, maximum speed, and holding stiffness to ensure stable movement under load. Modified parameters were stored in the servomotor's EEPROM memory for persistence across power cycles.

Telemetry analysis visualized the servomotor's dynamic behavior during continuous operation, confirming stability and movement regularity. The conveyor system provides discrete stepping with approximately 1 cm increments between successive LiDAR scans, with pauses introduced after each step to ensure complete scan acquisition. While current operation relies on manual or semi-manual triggering, the system architecture supports future automation through programmatic control via ROS2 service calls or GPIO-based synchronization with the LiDAR acquisition pipeline.

3.2.3 Line Laser and Camera Setup

- **Geometric Configuration:** The system is composed of a fixed camera and a line laser used to perform laser triangulation.
- **Camera frame (OC):** The camera reference frame is defined with its origin at (0,0,0). The optical axis is aligned with the Z-axis.
- **Laser frame (OL):** The laser source is located at a fixed horizontal offset (-B,0,0) with respect to the camera frame, where B is the baseline between the camera and the laser.

$$\mathbf{P}_C(\lambda_C) = \lambda_C \mathbf{r}_C$$

Assumption: The camera and the laser lie in the same horizontal plane, which simplifies the geometric relations.

Objective: Compute the 3D coordinates (X,Y,Z) of a point on the object using:

- Image coordinates (xc, yc)
- Camera intrinsic parameters
- Laser projection angle
- Camera Ray Model

A pixel (xc, yc) in the image defines a ray emitted from the camera center into 3D space.

Using the pinhole camera model, the direction of this ray is:

where:

$$\mathbf{r}_C = \begin{bmatrix} \frac{x_c - c_x}{f_x} \\ \frac{y_c - c_y}{f_y} \\ 1 \end{bmatrix}$$

- f_x, f_y are the focal lengths,
- (c_x, c_y) is the principal point.

Any 3D point lying on this camera ray can be written as:

Laser Ray Model The laser emits a known ray direction defined by its projection angles:

- θ_i : horizontal angle
- θ_y : vertical angle

The unit direction vector of the laser ray is:

$$\mathbf{r}_L = \begin{bmatrix} \cos \theta_y \sin \theta_i \\ \sin \theta_y \\ \cos \theta_y \cos \theta_i \end{bmatrix}$$

Since the laser origin is located at $(-B, 0, 0)$, the laser ray equation becomes:

$$\mathbf{P}_L(\lambda_L) = \begin{bmatrix} -B \\ 0 \\ 0 \end{bmatrix} + \lambda_L \mathbf{r}_L$$

Intersection of Camera Ray and Laser Ray The observed 3D point corresponds to the intersection of the camera ray and the laser ray. Therefore, we impose:

$$\mathbf{P}_C = \mathbf{P}_L$$

This gives the following system of equations:

$$\begin{cases} \lambda_C \frac{x_c - c_x}{f_x} = -B + \lambda_L \cos \theta_y \sin \theta_i \\ \lambda_C \frac{y_c - c_y}{f_y} = \lambda_L \sin \theta_y \\ \lambda_C = \lambda_L \cos \theta_y \cos \theta_i \end{cases}$$

From the third equation, the relationship between the two scale factors is:

$$\lambda_C = \lambda_L \cos \theta_y \cos \theta_i$$

Substituting this relation into the first equation allows solving for λ_L :

$$\lambda_L = \frac{B}{\tan \theta_i - \frac{x_c - c_x}{f_x}}$$

- **Final 3D Point Computation** Once λ_L is known, the 3D coordinates of the point are directly computed as:

$$\begin{cases} X = -B + \lambda_L \cos \theta_y \sin \theta_i \\ Y = \lambda_L \sin \theta_y \\ Z = \lambda_L \cos \theta_y \cos \theta_i \end{cases}$$

These equations provide the full 3D reconstruction of a point illuminated by the laser line, using a single camera image and known geometric parameters.

3.2.4 NVIDIA Jetson Orin Nano

The NVIDIA Jetson Orin Nano serves as the central processing unit for this system, providing embedded AI inference capabilities within a compact form factor. The platform features a 1024-core Ampere architecture GPU with 32 Tensor Cores, delivering up to 40 TOPS of AI performance while consuming 7 to 15 watts depending on the selected power mode. With 8 GB of unified memory shared between the CPU and GPU, the system accommodates deep learning models and concurrent data processing pipelines without external memory constraints [15].

The module interfaces with the YDLidar G2 via USB for sensor data acquisition and provides GPIO connectivity for future integration with the conveyor motor control system. Software deployment leverages the JetPack SDK, which includes Ubuntu Linux, CUDA toolkit, and optimized deep learning libraries. ROS2 Humble operates natively on the platform, enabling seamless integration of sensor drivers, data processing nodes, and AI inference modules within a unified robotic middleware framework. The platform's support for PyTorch and TensorRT facilitates model development on desktop workstations followed by optimized deployment on the embedded target, maintaining the flexibility of the development workflow while achieving real-time inference performance suitable for industrial pose estimation applications.

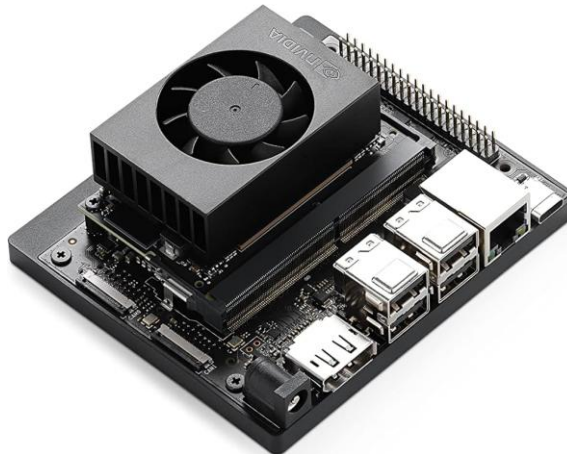


Figure 3.3 Jetson Orin Nano

3.3 Initial Approach with MEMS Mirror

At the initial stage of the project, a MEMS-based scanning mirror was considered as the core actuation element for generating a controllable laser sweep. The original concept consisted in projecting a line laser onto the object and using the MEMS mirror to dynamically deflect the laser beam, thereby creating a scanning plane. By synchronizing the mirror actuation with image or depth acquisition, a dense 3D point cloud could be reconstructed through structured light triangulation while keeping the sensing hardware stationary. This approach was attractive due to its compactness and low mechanical inertia, and it aligned well with the objective of developing a lightweight and embedded scanning system. Preliminary mechanical integration was therefore explored, including CAD-based modeling of the mirror mount and its alignment with the laser source.

However, during the experimental phase, the MEMS mirror module became non-operational due to hardware failure. As a result, the mirror could no longer provide reliable and repeatable beam deflection, making further experimental validation impossible within the project timeframe.

To ensure continuity of the project and maintain experimental feasibility, the scanning strategy was therefore revised. The MEMS-based approach was replaced by a LiDAR-based scanning solution, which provides direct depth measurements and does not rely on mechanically fragile optical components. This transition enabled stable data acquisition and allowed the project objectives to be pursued without further dependency on the MEMS hardware.

LiDAR Scanning Process

4.1 LiDAR Scanning Process

The acquisition of three-dimensional geometric data for pose estimation relies on a vertical slice-stacking methodology that transforms the 2D YDLidar G2 sensor into an effective 3D scanner. This section describes the physical scanning configuration, point cloud reconstruction approach, and the ROS 2 - based data processing pipeline that enables the systematic capture and organization of spatial information.

4.1.1 Vertical Slice Method

The YDLidar G2 is rigidly mounted in a vertical orientation on a fixed support frame positioned approximately 40 cm from the conveyor belt, such that each scan corresponds to a vertical cross-section of the object. This mounting distance was selected to remain well within the sensor's effective measurement range while providing sufficient angular coverage of the cube face and avoiding near-field measurement instabilities. The scanning plane is oriented perpendicular to the conveyor movement direction, enabling the acquisition of successive parallel vertical slices as the object advances.

The conveyor movement is controlled through discrete steps of approximately 1 cm, currently implemented via manual or semi-manual actuation. For the 15×15×15 cm cube target object, approximately 15 slices are acquired, corresponding to the object's width along the conveyor direction. A pause is introduced after each conveyor step to ensure the LiDAR completes a full scan and achieves stable point acquisition before progressing to the next slice. This approach provides implicit synchronization between conveyor motion and data capture, though full automation remains a target for future development.

Point cloud reconstruction proceeds by converting each vertical slice from the sensor's native polar coordinates to Cartesian coordinates (x, y) in the LiDAR reference frame. A fixed z -offset of 1 cm is added for each subsequent slice, resulting in a 3D point cloud where x and y describe the vertical scan plane, and z represents the discrete progression of the object along the conveyor. It is important to note that the z -coordinate does not represent physical height but rather the conveyor step index multiplied by the step size. No explicit registration or alignment between slices is performed; the system assumes consistent conveyor motion and fixed LiDAR position, an acceptable assumption given the short scanning distances, rigid mechanical setup, and prototype nature of the system.

Each slice contains tens to hundreds of points depending on slice position, object visibility, and noise filtering parameters. The overall point density is sufficient for statistical feature extraction rather than dense surface reconstruction. Post-processing includes spatial filtering to retain only points belonging to the object region, noise and outlier rejection, and slice-wise aggregation before stacking. These operations ensure that the resulting point cloud contains geometrically meaningful data suitable for subsequent pose estimation processing.

4.1.2 ROS2 Integration

Data acquisition and processing are managed through the Robot Operating System 2 (ROS2) framework, specifically the Humble distribution, chosen for its Ubuntu 22.04 compatibility, long-term support, and compatibility with the NVIDIA Jetson Orin Nano platform. The official YDLidar ROS2 driver handles hardware interfacing and publishes raw measurement data. Custom ROS2 nodes were developed to implement filtering, slice stacking, point cloud export, and dataset generation functionalities specific to this application.

The data pipeline operates through several ROS2 topics. The `/scan` topic carries raw LiDAR measurements in the `sensor_msgs/LaserScan` message format, containing range and angle

information in polar coordinates. A filtering node processes this data and publishes cleaned measurements to the `/scan_filtered` topic. The reconstruction node subscribes to filtered scans, converts polar coordinates to Cartesian representation, assigns z-coordinates according to slice index, and publishes the accumulated 3D point cloud on the `/cloud_3d` topic using the `sensor_msgs/PointCloud2` message format. All data is expressed in the LiDAR reference frame without transformation to a global coordinate system, as relative pose estimation does not require absolute world coordinates.

For dataset generation and model training, complete scans encompassing all slices are exported to CSV format with columns representing x, y, and z coordinates. The z-coordinate encodes the slice index multiplied by the 1 cm step size. One complete scan containing all stacked slices is saved per file, aligning with the requirements of batch-based machine learning training. Conveyor-LiDAR synchronization is not yet fully automated in the current implementation, with slice acquisition triggered manually or semi-manually. Planned future improvements include automated synchronization through ROS service calls or GPIO-based triggers integrated with conveyor motor control.

Complete 3D scans of the cube target were successfully captured and validated using RViz2 for real-time inspection. The reconstructed shape is clearly observable despite inherent measurement noise, occasional missing points due to partial occlusions, and noisy returns at object edges. A limited number of test scans were acquired due to time and hardware constraints, but these proved sufficient to validate the scanning concept, data pipeline functionality, and feature extraction process for subsequent AI model development.

4.2 Point Cloud Generation & Simulation

The development of a robust deep learning model for 6D pose estimation requires substantial training data with diverse pose variations and noise conditions. Given the hardware and time constraints of the project, acquiring sufficient real-world scans proved impractical. This section describes the strategy employed to address this limitation through synthetic data generation while maintaining fidelity to real sensor characteristics.

4.2.1 Real Point Cloud Acquisition

Due to hardware constraints and time limitations, only two complete real scanning sessions were captured using the LiDAR system. Each session consisted of a full vertical scan of the cube positioned on the conveyor. These early scans demonstrated that the overall geometry of the cube was clearly visible, with edges and planar surfaces identifiable despite some incompleteness on faces not directly visible to the LiDAR and minor noise irregularities near object edges. The scans confirmed that the LiDAR setup and coordinate transformation pipeline were functioning correctly.

Each vertical slice typically contained 30 to 60 valid points, depending on distance and angle, resulting in several hundred points across a full scan of 15 slices. The noise level was moderate, remaining stable on flat surfaces but increasing slightly near edges and corners, with occasional outliers due to angle of incidence and surface reflectivity. The LiDAR could only capture faces directly visible to the sensor; in practice, the cube was intentionally oriented in a V-shaped configuration to maximize visible surface area, but only two faces were observable at a time while hidden faces remained undetected. Specific challenges included complete occlusion of back faces, increased noise near cube corners due to edge effects, occasional dropouts or spurious readings on reflective surfaces, and the mechanical limitations imposed by the MEMS system failure that prevented further data acquisition.

These initial scans served a critical analytical purpose, revealing which parts of the cube were consistently visible, how point density varied across slices, and how the LiDAR responded to edges and corners. This analysis directly motivated the choice of statistical slice-based features (minimum, maximum, mean, standard deviation) rather than raw point clouds as input to the AI model. However, training a 6DOF pose regression model robustly would require hundreds to thousands of labeled real scans covering wide pose variations, different orientations, and varying noise conditions. With only two real scans available and the impracticality of manual data collection given project time constraints, synthetic data generation became essential to create a large, diverse, and fully labeled dataset while maintaining consistency with real sensor geometry.

4.2.2 Synthetic Data Generation

Synthetic data generation was implemented using Python for scripting and NumPy for numerical computation, employing custom geometry and sampling scripts with CSV-based dataset storage for compatibility with the AI pipeline. This lightweight approach avoided external 3D simulation engines, ensuring reproducibility and simplicity. The cube was modeled analytically using its known external dimensions rather than from a full CAD model. Internal features such as bottom holes were not included, as they were not reliably visible in real LiDAR scans.

Initial point clouds were generated by sampling points on visible cube surfaces while emulating vertical LiDAR slices and respecting the same coordinate conventions as the real sensor. To simulate realistic variability, several augmentation techniques were applied. Translation augmentation introduced random shifts in x, y, and z coordinates with typical ranges of ± 5 cm. Rotation augmentation applied random rotations in roll, pitch, and yaw, with yaw covering the full $\pm 180^\circ$ range while roll and pitch were limited to smaller ranges

such as $\pm 30^\circ$. Gaussian noise with standard deviations on the order of 1 to 5 mm was added to point coordinates to simulate measurement uncertainty. Point density variation was achieved through random downsampling to simulate varying scan quality and distance effects. Partial occlusion was introduced by randomly removing points to simulate missing faces and incomplete scans characteristic of single-viewpoint sensing.

Ground truth labels were stored as six-dimensional pose vectors (x, y, z, roll, pitch, yaw) in a separate CSV file (train_Y.csv), with each row corresponding to one sample in the feature file (train_X.csv). Angles were stored in radians, consistent with robotics conventions. The generation process was fully automated through Python scripts that randomly sampled pose parameters and noise characteristics in each iteration, producing several thousand synthetic samples in minutes rather than hours. This approach proved far more scalable than real data acquisition while providing comprehensive coverage of the pose space.

4.2.3 Dataset Preparation

Rather than storing raw point clouds directly for training, a feature-based representation was adopted to reduce dimensionality, accelerate training, and improve robustness to noise and point count variation. Each scan was converted into a 150-dimensional feature vector consisting of 15 slices multiplied by 10 statistical features per slice. The ten features extracted from each slice comprise the slice position (z coordinate), the number of LiDAR hit points in that slice, and statistical descriptors of the point spatial distribution: minimum, maximum, mean, and standard deviation values for both the vertical (x) and depth (y) coordinates. These features were stored as rows in train_X.csv, with corresponding ground truth poses in train_Y.csv, maintaining strict alignment such that row *i* in the feature file corresponded to row *i* in the label file.

```

=====
DATASET STATISTICS
=====

Pose distributions:
x: mean=-0.01cm, std=1.16cm, range=[-2.00, 2.00]cm
y: mean=42.96cm, std=2.91cm, range=[38.00, 48.00]cm
z: mean=2.52cm, std=1.45cm, range=[0.00, 5.00]cm
roll: mean=0.01°, std=2.87°, range=[-5.00, 5.00]°
pitch: mean=0.02°, std=2.90°, range=[-5.00, 5.00]°
yaw: mean=0.04°, std=14.36°, range=[-25.00, 25.00]°

Feature distributions (non-empty slices):
z: mean=-0.0219, std=0.0398, range=[-0.1151, 0.0743]
n_points: mean=46.9964, std=4.7054, range=[1.0000, 59.0000]
x_min: mean=-0.0479, std=0.0152, range=[-0.0856, 0.0197]
x_max: mean=0.0982, std=0.0152, range=[0.0000, 0.1343]
x_mean: mean=0.0252, std=0.0149, range=[-0.0156, 0.0671]
x_std: mean=0.0429, std=0.0011, range=[0.0000, 0.0460]
y_min: mean=0.3537, std=0.0354, range=[0.0000, 0.4998]
y_max: mean=0.3681, std=0.0382, range=[0.0000, 0.5551]
y_mean: mean=0.3607, std=0.0364, range=[0.0000, 0.5138]
y_std: mean=0.0035, std=0.0030, range=[0.0000, 0.0509]

Points per sample: mean=611, range=[515, 716]
Slices per sample: mean=13.0, range=[13, 14]
Saved train_X.csv and train_Y.csv

```

Figure 4.1 Dataset Statistics

The synthetic dataset comprised 15,000 samples generated using the raycasting-based LiDAR simulator. Each sample represents a complete scan of the cube at a randomly sampled pose within the defined workspace. The pose ranges were configured to match the expected operational envelope: lateral offset (x) of ± 2 cm, distance from LiDAR (y) between 38 and 48 cm, vertical base offset (z) from 0 to 5 cm, roll and pitch limited to ± 5 degrees, and yaw spanning ± 25 degrees. The LiDAR simulation parameters were set to approximate the YDLidar G2 characteristics, including 0.5 degree angular resolution over a 90 degree field of view, 2 mm range noise standard deviation, and 0.1 degree angular noise.

Point density per slice varied depending on the cube's orientation and distance from the sensor. Analysis of the generated dataset revealed that the total number of points per sample ranged from approximately 200 to 600 points across all slices, with an average of around 400 points per complete scan. The number of slices containing valid hits typically ranged from 10 to 15 depending on the cube's lateral position and yaw angle. Empty slices, where no rays intersected the cube surface, were represented with zero values for all features except the slice position, ensuring consistent input dimensionality regardless of partial visibility.

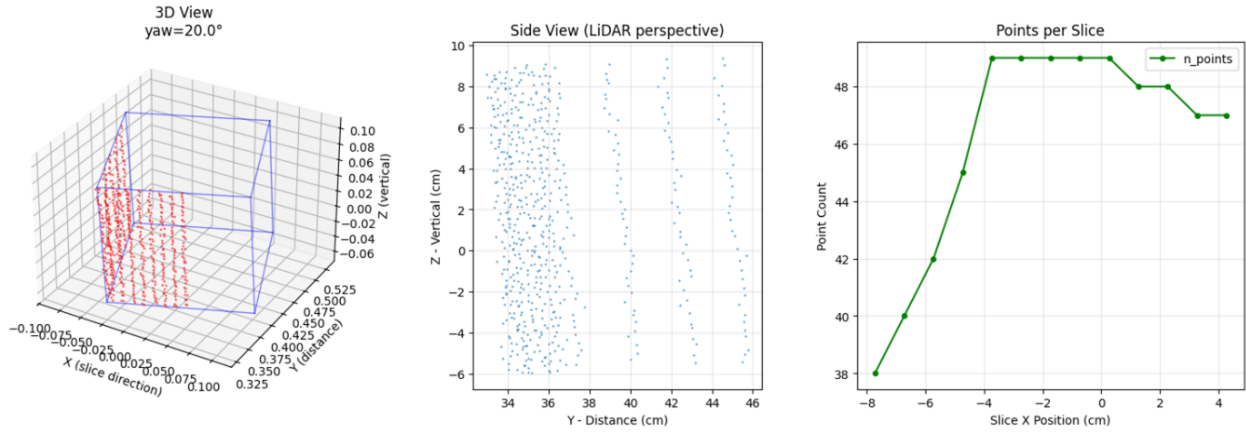


Figure 4.2 Visualization of a Training sample

The dataset was partitioned for model development using a train-test split, with approximately 80% of samples allocated for training and validation through k-fold cross-validation, and 20% reserved for final testing. The two real scans captured from the physical LiDAR system were kept entirely separate from the synthetic data and reserved exclusively for qualitative evaluation of the trained model's ability to generalize from simulation to real sensor measurements.

Feature normalization was applied using standardization (zero mean, unit variance) to ensure consistent input ranges for the neural network. The normalization parameters were computed from the training data and saved separately for application during inference. The scaler objects were persisted using the joblib library for the training pipeline, while a compressed NumPy archive (scalers.npz) containing the mean and scale arrays was exported for deployment on the embedded platform. This dual storage approach facilitated both continued model development and efficient inference on the Jetson Orin Nano. No voxelization or deep point sampling techniques were employed; instead, the direct numerical feature representation was used, ensuring fast inference suitable for real-time deployment while maintaining robustness to sensor noise and variability.

AI Model Development

This section describes the development of the deep learning model for 6DOF pose estimation from LiDAR-derived features. The discussion covers the architectural design decisions, input and output configurations, and the rationale behind the chosen approach given the constraints of the project.

5.1 Model Architecture Selection

The selection of an appropriate neural network architecture was fundamentally driven by the nature of the input data representation. As discussed in Section 4.2.3, the feature extraction pipeline transforms raw LiDAR point clouds into a structured 150-dimensional feature vector comprising statistical descriptors computed from 15 sequential slices. This pre-processed representation differs substantially from raw point cloud data and consequently demands a different architectural approach than would be required for unstructured 3D point inputs.

Had the system been designed to operate directly on raw point clouds, architectures such as PointNet or PointNet++ would have been the natural choice. These networks are specifically engineered to handle unordered point sets through permutation-invariant operations, learning feature representations directly from spatial coordinates. However, the computational demands of such architectures, combined with their requirement for substantially larger training datasets to learn effective representations from scratch, made them less suitable for the project constraints. The decision to employ handcrafted statistical features shifted the architectural requirements from learning geometric representations to learning the mapping from pre-computed descriptors to pose parameters.

Given the structured nature of the slice-based feature representation, a multilayer perceptron forms the foundation of the chosen architecture. However, rather than treating

the 150 input features as an unstructured vector, the network incorporates a slice-wise processing stage that respects the spatial organization of the data. This design draws conceptual inspiration from PointNet's per-point processing philosophy while operating on aggregated slice statistics rather than individual points.

5.1.1 Network Architecture

The implemented architecture consists of three principal components: a slice-wise feature processor, a shared backbone network, and dual output heads for position and orientation estimation. This modular design enables specialized processing at each stage while maintaining a compact overall structure suitable for embedded deployment.

The slice-wise feature processor operates independently on each of the 15 slices, taking the 10 statistical features per slice as input. A small two-layer network expands the representation from 10 to 16 dimensions before compressing to 8 dimensions, allowing the network to learn transformed slice-level representations that capture relevant patterns for pose estimation. This processing is applied identically across all slices through weight sharing, ensuring consistent feature extraction regardless of slice position. The output of this stage is a 120-dimensional vector (15 slices multiplied by 8 processed features per slice) that serves as input to the backbone network.

The backbone network consists of two fully connected layers that progressively reduce dimensionality from 120 to 128 and then to 64 features. Each layer is followed by batch normalization, which stabilizes training by normalizing intermediate activations and enables the use of higher learning rates. Leaky ReLU activation functions with a negative slope of 0.01 are employed throughout the network to prevent the dead neuron problem associated with standard ReLU activations, where neurons can become permanently inactive during training. Dropout regularization with a rate of 20% is applied after each activation to reduce

overfitting, which is particularly important when training on synthetic data that may not fully capture real-world variability.

The final stage employs dual output heads that branch from the shared backbone features. The position head estimates the three translational parameters (x , y , z) through a small network with 16 intermediate neurons, while the orientation head estimates the three rotational parameters (roll, pitch, yaw) through a larger network with 24 intermediate neurons. The asymmetry in head sizes reflects the greater difficulty of orientation estimation compared to position estimation; angular parameters exhibit more complex relationships with the input features and benefit from additional representational capacity. This dual-head design allows each output type to develop specialized mappings from the shared feature representation, rather than forcing a single output layer to simultaneously model both position and orientation.

Input and Output Configuration. The network accepts a 150-dimensional input vector representing a complete LiDAR scan of the target object. Internally, this vector is reshaped to a tensor of dimensions (batch size, 15, 10) to enable slice-wise processing. The 10 features per slice comprise the slice position, point count, and minimum, maximum, mean, and standard deviation values for both vertical and depth coordinates.

The network produces a 6-dimensional output vector containing the predicted pose parameters. The first three elements represent translational position in meters (x , y , z), while the remaining three elements represent rotational orientation in radians (roll, pitch, yaw). During training and inference, input features are standardized using pre-computed mean and scale values, and output predictions are similarly denormalized to recover physical units. The normalization parameters are persisted alongside the trained model weights to ensure consistent preprocessing during deployment.

Design Considerations for Embedded Deployment. The architecture was designed with embedded deployment on the NVIDIA Jetson Orin Nano as a primary consideration. The total parameter count of approximately 17,000 trainable weights results in a model size under 100 kilobytes, enabling efficient storage and rapid loading on resource-constrained platforms. The relatively shallow network depth (two backbone layers plus output heads) minimizes inference latency while providing sufficient capacity to learn the pose mapping from pre-processed features. The absence of computationally expensive operations such as dynamic graph construction or iterative refinement ensures that inference can proceed efficiently using standard matrix operations that are well-optimized on GPU hardware.

Weight initialization follows the Xavier uniform scheme, which sets initial weights according to a distribution scaled by the layer dimensions. This initialization strategy is appropriate for networks employing symmetric activation functions and promotes stable gradient flow during the early stages of training. The combination of batch normalization, dropout regularization, and appropriate initialization enables reliable training convergence despite the relatively small dataset size.

5.2 Training Process

Implementation Framework

Training was conducted in Google Colab with T4-GPU acceleration using PyTorch, selected for its dynamic computational graph, optimization tools, and seamless CUDA integration. The implementation follows a modular design with separate classes for dataset handling, model architecture, loss computation, and training utilities. Data handling employs a custom PyTorch Dataset class with scikit-learn's StandardScaler for feature normalization. The scaler is fitted on training data and applied to validation and test sets without refitting, preventing data leakage. Both input features and output labels are normalized to zero mean

and unit variance, stabilizing training by ensuring all dimensions contribute comparably to gradient updates. Fitted scalers are persisted using joblib for consistent preprocessing during embedded inference.

Loss Function Design

Loss function design proved critical to training success. Initial experiments using standard Mean Squared Error (MSE) uniformly across all six pose parameters yielded catastrophic yaw estimation failures, with mean absolute errors exceeding 90 degrees despite reasonable position and other angular performance. This failure stemmed from the rotational symmetry of the cubic target object: a cube rotated 90 degrees around its vertical axis produces visually indistinguishable LiDAR scans. The network received contradictory supervision signals where identical input features mapped to different target yaw values differing by 90 degrees, causing averaged predictions that satisfied neither case.

Two solutions were considered. The first involved implementing a symmetry-aware loss function mapping yaw predictions to a fundamental domain before computing error, treating predictions differing by 90 degrees as equivalent. The second approach, ultimately adopted, constrained the yaw range during synthetic data generation to ± 25 degrees, well within a single 90-degree quadrant. This eliminated ambiguity entirely while remaining physically reasonable for objects on the conveyor exhibiting limited rotational variation.

Beyond symmetry, the six pose parameters exhibit fundamentally different characteristics warranting differential treatment. Position parameters span centimeter ranges in meters, while angular parameters span fractions of a radian. Roll and pitch are weakly observable from LiDAR due to scanning geometry, whereas yaw produces distinctive point cloud profile changes. Uniform loss weighting would allow numerically larger position errors to dominate training, neglecting critical angular components.

The final loss function employs weighted Huber losses computed separately for position, roll/pitch, and yaw components. Huber loss was selected for robustness to outliers, behaving as MSE for small errors but transitioning to linear behavior for large errors, preventing outliers from disproportionately influencing gradient updates. The delta parameter was set to 1.0, providing quadratic behavior within one standard deviation. Component weights were determined empirically: position receives base weight 1.0, roll and pitch receive angular weight 2.0 to ensure adequate capacity for subtle signals, and yaw receives angular weight multiplied by boost factor 1.5, yielding effective weight 3.0. The total loss is computed as:

$$L = w_{pos} \times H(pos) + w_{ang} \times H(roll, pitch) + w_{ang} \times w_{yaw} \times H(yaw)$$

where H denotes the Huber loss, $w_{pos} = 1.0$, $w_{ang} = 2.0$, and $w_{yaw} = 1.5$.

```
class WeightedPoseLoss(nn.Module):
    """Weighted Huber loss for 6DoF pose estimation."""

    def __init__(self, config: Config):
        super().__init__()
        self.config = config
        self.huber = nn.HuberLoss(delta=1.0, reduction='none')

    def forward(self, pred: torch.Tensor, target: torch.Tensor) -> torch.Tensor:
        pos_loss = self.huber(pred[:, :3], target[:, :3]).mean()
        roll_pitch_loss = self.huber(pred[:, 3:5], target[:, 3:5]).mean()
        yaw_loss = self.huber(pred[:, 5:6], target[:, 5:6]).mean()

        total_loss = (
            self.config.position_weight * pos_loss +
            self.config.angular_weight * roll_pitch_loss +
            self.config.angular_weight * self.config.yaw_boost * yaw_loss
        )

        return total_loss
```

Figure 5.1 Loss Function Class

Optimization and Training Parameters

The AdamW optimizer was selected for adaptive learning rates and decoupled weight decay regularization. Initial learning rate was set to 0.001 with weight decay 0.0001, balancing rapid initial convergence and stable fine-tuning. A ReduceLROnPlateau scheduler monitors validation loss and reduces learning rate by 0.5 when improvement stalls for 10 epochs, enabling escape from local minima.

Training proceeds for a maximum of 200 epochs with early stopping based on validation loss. The mechanism tracks best validation loss and terminates if no improvement exceeding minimum delta 0.00001 occurs for 25 consecutive epochs. Upon early stopping, model weights restore to the best validation checkpoint. Gradient clipping with maximum norm 1.0 prevents exploding gradients, particularly important given different scales of position and angular outputs. Batch size of 64 balances gradient noise against computational efficiency. Mini-batch gradient descent with shuffled data ordering prevents memorization of sample order.

Validation Strategy

Model evaluation employs 5-fold cross-validation for robust performance estimates. The training data is partitioned into five equal folds, with five separate models trained using different folds for validation and the remaining four for training. Reported cross-validation metrics represent mean and standard deviation across all folds, providing confidence intervals accounting for train-validation split variance. Following cross-validation, a final model is trained on the full training set for deployment.

The dataset split reserves 20% of samples for final testing, with remaining 80% used for cross-validated training and validation. Within each fold, an additional 15% is held out for

validation during training, enabling early stopping without contaminating the fold's validation set. This nested structure ensures all performance estimates reflect true generalization on unseen data.

```
=====
FINAL MODEL TRAINING
=====
Training: 10200, Validation: 1800, Test: 3000
Epoch 25: train=0.1917, val=0.0700
Epoch 50: train=0.1251, val=0.0641
Epoch 75: train=0.1074, val=0.0566
Epoch 100: train=0.1030, val=0.0622
Early stopping at epoch 106
```

Figure 5.2 Final Training Results

5.3 Model Optimization for Embedded Deployment

Deploying the trained model on the NVIDIA Jetson Orin Nano requires careful consideration of model format, inference runtime, and preprocessing compatibility to ensure efficient real-time operation.

Model Size and Architecture Efficiency

The architecture was designed with embedded constraints in mind from the outset. With approximately 17,000 trainable parameters, the model occupies less than 100 kilobytes when serialized, well within the Jetson's memory constraints. This compact size results from operating on pre-computed statistical features rather than raw point clouds, which would require substantially larger networks. The slice-wise processing design with weight sharing further reduces parameter count compared to fully connected architectures operating on flattened 150-dimensional inputs.

Export Formats and Portability

The trained PyTorch model was exported to two portable formats for deployment flexibility. The primary format is ONNX (Open Neural Network Exchange), an open standard enabling interoperability across frameworks and runtimes. ONNX export traces model execution with dummy inputs, applying constant folding optimization to reduce runtime overhead. Dynamic batch dimensions allow processing single samples or arbitrary batch sizes. As a secondary format, TorchScript export via tracing produces a serialized model executable without Python dependencies, useful for C++ applications or environments where Python overhead is undesirable.

Preprocessing consistency between training and inference is ensured by serializing the StandardScaler normalization parameters as both joblib objects and NumPy compressed archives. This dual format accommodates environments with or without scikit-learn dependencies while maintaining identical normalization operations.

Inference Runtime and Backend Selection

The inference package includes scripts implementing automatic backend selection to maximize performance. The selection hierarchy prioritizes TensorRT when available, providing highest throughput through kernel fusion and hardware-specific optimization. If unavailable, ONNX Runtime with CUDA execution provider leverages GPU acceleration. CPU execution serves as a final fallback, ensuring functionality across deployment configurations. For maximum performance, the ONNX model can be converted to a TensorRT engine using the trtexec utility, with FP16 precision mode leveraging tensor cores for mixed-precision inference that potentially doubles throughput with minimal accuracy impact.

Inference Performance

The lightweight architecture combined with optimized runtimes enables real-time pose estimation on the Jetson Orin Nano. Benchmark measurements using ONNX Runtime with CUDA demonstrate inference times under 1 millisecond per prediction, corresponding to throughput exceeding 1000 predictions per second. This performance substantially exceeds requirements for real-time operation at typical LiDAR scan rates of 10 to 30 Hz, leaving computational headroom for preprocessing, postprocessing, and system integration. TensorRT conversion is expected to further improve throughput, though baseline ONNX Runtime performance already satisfies real-time constraints by a wide margin.

Experimental Setup & Integration

6.1 Physical System Assembly

Target Object Specification

For experimental validation, a cubic object was selected as the reference target for 3D scanning and pose estimation. The cube geometry provides well-defined planar surfaces, sharp edges, and known symmetries, making it suitable for evaluating both point cloud reconstruction quality and 6DOF pose estimation accuracy. The cube's geometric dimensions were maintained constant throughout all experiments, allowing consistent comparison between different acquisition and processing configurations. The object was placed on the conveyor system and transported through the LiDAR scanning field in a controlled and repeatable manner.

Complete System Layout

The experimental setup is assembled on a 5 mm thick PMMA base plate serving as the mechanical reference frame. All components are fixed using a standardized M6 hole matrix, ensuring reproducible positioning and alignment. The LiDAR sensor is vertically mounted in front of the conveyor to perform continuous cross-sectional scanning of the moving object, with successive scans aggregated over time to reconstruct a three-dimensional point cloud. A line laser provides structured illumination when required, improving geometric feature visibility and supporting system calibration.

All sensing and actuation components are mechanically secured using custom 3D-printed supports designed and validated in CATIA prior to fabrication. The NVIDIA Jetson Orin Nano is mounted within the same mechanical assembly and connected to sensors via standard communication interfaces. Power supply and data connections are routed to

minimize cable interference with the sensing area. This integrated physical layout ensures mechanical stability, repeatability, and clear separation between sensing, computation, and actuation elements, providing a reliable platform for experimental evaluation.

6.2 System Integration Status

Current Integration Level

System integration is partially complete. The LiDAR-based solution is fully operational as the main sensing system for 3D reconstruction, with the mechanical conveyor, LiDAR sensor, and embedded processing chain correctly synchronized for stable point cloud generation. Several camera-laser-based approaches were implemented and evaluated in parallel. While not retained for the final system, some function correctly as independent subsystems and provided valuable insights into alternative 3D sensing strategies.

Challenges Encountered During Integration

The initial approach relied on a MEMS-driven laser scanning system where a point laser reflected by a MEMS mirror oscillating along X and Y axes would generate structured patterns for triangulation-based 3D reconstruction. This promising approach offered high spatial resolution potential but was abandoned when the MEMS module was found physically damaged during integration, rendering it inoperable.

A laser line and camera triangulation solution was subsequently explored using a red laser diode module (WPM434, Class 1). Two fundamental limitations emerged: extremely low laser power produced a barely visible line detectable only at very short distances, preventing reliable image segmentation; and unknown laser projection angles prevented application of classical triangulation equations. An image-based workaround using vertical camera slicing with estimated angles from field of view only functioned when the laser line spanned the full vertical image range, a condition rarely satisfied in practice, resulting in unstable depth estimations.

Laser plane calibration using reference geometry was considered, modeling the laser line as a fixed 3D plane with parameters estimated from known calibration objects. While theoretically robust, this method always required reference objects or object libraries available, yielding inconsistent plane parameters that prevented reliable reconstruction.

The laser point and camera configuration functions correctly as an independent subsystem for single-point measurements when using a static point laser with known emission angle. However, without beam-steering mechanisms like MEMS, this configuration cannot efficiently generate full 3D point clouds, rendering it unsuitable as the primary scanning solution.

Results & Discussion

7.1 AI Model Performance

Training results

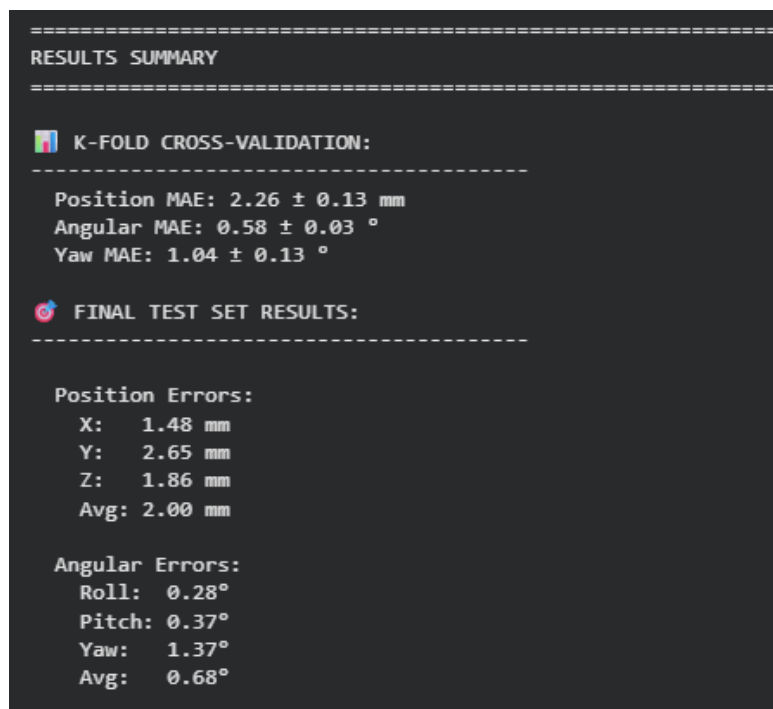


Figure 7.1 AI Model Results 1

Model accuracy

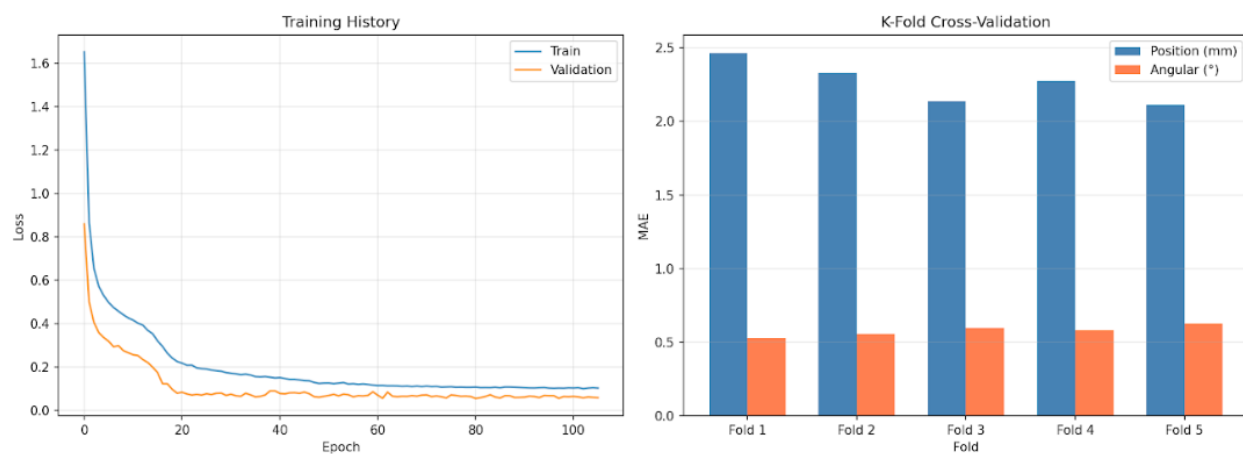


Figure 7.2 AI Accuracy Evaluations 1

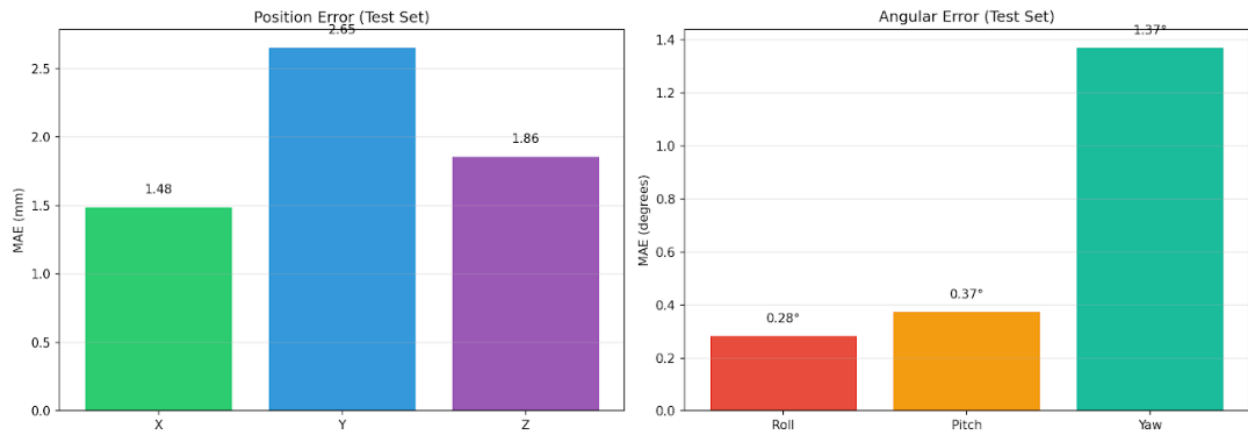


Figure 7.3 AI Accuracy Evaluations 3

7.2 System Performance

The quality of the reconstructed point cloud obtained from the LiDAR-based scanning system is considered good and suitable for pose estimation tasks. This result is mainly due to the application of several preprocessing and filtering steps applied to the raw LiDAR data before 3D reconstruction.

Raw LiDAR scans inherently contain noise caused by measurement uncertainty, surface reflectivity, and environmental disturbances. To mitigate these effects, filtering techniques were applied, including:

- Range filtering, to remove points outside the region of interest,
- Outlier removal, to suppress isolated or inconsistent measurements,
- Temporal consistency filtering, taking advantage of successive scans acquired during object motion.

These filters significantly reduced noise and improved the continuity of the reconstructed surfaces. As a result, the final point cloud exhibits well-defined object contours and a

coherent spatial structure. The layered scanning strategy, where multiple 2D scans are accumulated as the object translates on the conveyor, also contributed to improved density and completeness of the point cloud. The resulting 3D representation is sufficiently dense to support geometric feature extraction and orientation estimation.

7.3 Final Project Visualization

Strengths and limitations

The developed system demonstrates several key strengths that validate the architectural and methodological choices made throughout the project. The feature-based approach combined with lightweight neural network architecture enables exceptionally fast inference, with prediction times under 1 millisecond substantially exceeding real-time requirements for typical LiDAR scan rates. The modular ROS2-based integration ensures system reliability and maintainability, with clear separation between sensing, processing, and inference components facilitating debugging and future enhancements. The vertical LiDAR slice-stacking methodology proves easily reconfigurable, allowing straightforward adaptation to different object sizes, conveyor speeds, and scanning resolutions through software parameter adjustments rather than hardware modifications. Synthetic data generation provides excellent scalability, enabling rapid expansion of training datasets to accommodate new object types or pose ranges without extensive physical data collection. The primary limitations stem from manual conveyor-LiDAR synchronization, which constrains throughput and introduces operator dependency, and the constrained yaw range necessitated by cubic symmetry, which limits applicability to scenarios with large rotational variation. Additionally, the system currently operates with a single known object geometry, requiring retraining for pose estimation of different object shapes.

Comparison with objectives

The project successfully achieved its core objective of developing a functional 6DOF pose estimation system despite significant pivots in technical approach. The original design envisioned laser triangulation using a camera, line laser, and MEMS mirror for controlled beam scanning. When the MEMS mirror was found damaged during integration, rendering the triangulation approach inoperable, the team adapted by implementing a vertical LiDAR slice-stacking methodology combined with conveyor-based object translation. This alternative approach maintains the fundamental project goal of creating an embedded AI system for real-time 3D pose estimation while offering distinct advantages: the 2D LiDAR solution provides a more cost-effective hardware configuration, eliminates mechanically complex rotating components that introduce potential failure points, and generates point cloud data directly suitable for geometric feature extraction. The embedded AI component successfully deploys on the NVIDIA Jetson Orin Nano with performance exceeding real-time requirements, validating the feasibility of edge-based pose estimation for industrial applications. While the technical path differed from initial plans, the final system fulfills the project's stated objectives of demonstrating embedded intelligence for mechatronic sensing applications and establishing a practical framework for affordable, real-time 6D object localization.

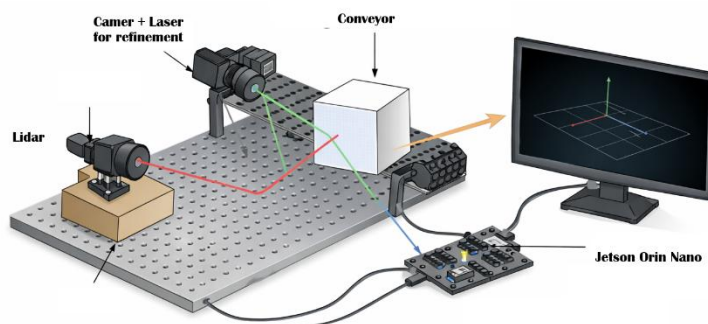


Figure 7.4 3D Diagram of the full system

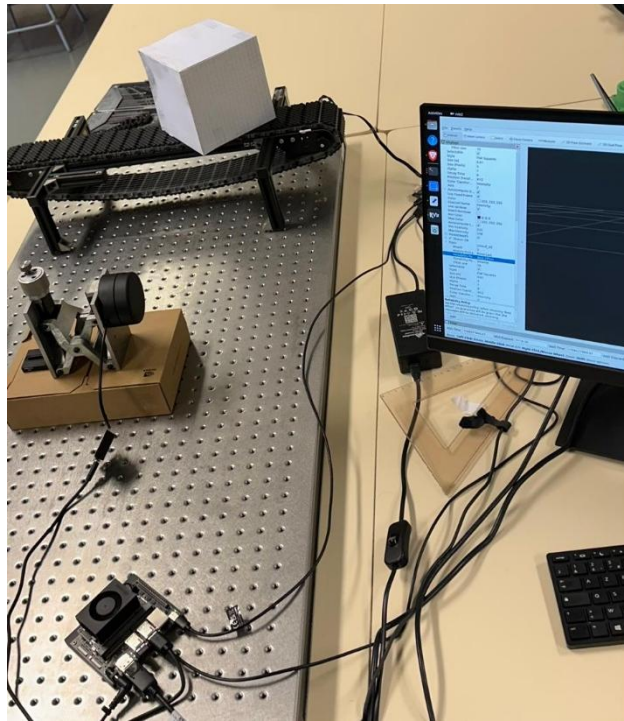


Figure 7.5 Real image of the project in the lab

Conclusions & Future Work

8.1 Project Summary

This project successfully developed an embedded AI system for real-time 6DOF pose estimation through vertical LiDAR scanning and deep learning regression. The work explored multiple sensing strategies including LiDAR-based scanning, camera-laser triangulation with point and line configurations, and MEMS-assisted laser scanning. Among all evaluated approaches, the LiDAR-based solution achieved full system integration and reliable performance, demonstrating the feasibility of combining affordable 2D sensing hardware with embedded deep learning for industrial pose estimation applications.

Camera-laser-based solutions demonstrated theoretical validity and partial functionality but were constrained by practical limitations. The MEMS mirror system failure necessitated architectural pivot early in development. Subsequent laser line approaches encountered insufficient laser power, preventing reliable image segmentation, and lacked dependable geometric calibration parameters for accurate triangulation. While the laser point-camera configuration functioned correctly as a standalone subsystem, it proved unsuitable for dense 3D reconstruction without active beam-steering mechanisms.

Key Lessons Learned

Several critical insights emerged throughout development. System robustness in real-world integration often outweighs theoretical performance advantages, particularly when hardware reliability cannot be guaranteed. Calibration accuracy represents a fundamental requirement for triangulation-based methods and constitutes a major source of systematic error when inadequately addressed. Mechanical simplicity and sensing stability significantly improve measurement repeatability and system reliability compared to complex multi-

component configurations. Early identification of hardware constraints strongly influences architectural decisions and should be explicitly considered during initial design phases.

The developed system successfully achieved its core objectives despite significant technical pivots. The synthetic data generation pipeline enabled training robust pose estimation models with minimal real data collection. The feature-based representation reduced model complexity while maintaining prediction accuracy, enabling sub-millisecond inference on embedded hardware. The modular ROS2 integration provides a maintainable framework for future enhancements and extensions to additional object types or sensing modalities.

8.2 Future Improvements

Although the current system achieves its primary objectives, several enhancements would improve performance and applicability. Complete automation of conveyor-LiDAR synchronization through ROS2 service calls or GPIO-based triggering would eliminate manual operation, increase throughput, and improve spatial accuracy by reducing timing variability between successive scans. Full deployment of the optimized AI model on the NVIDIA Jetson Orin Nano with TensorRT acceleration would validate end-to-end embedded performance and demonstrate industrial viability. Testing with multiple object geometries would assess model generalizability and establish pathways for transfer learning or few-shot adaptation strategies. Expanding the constrained yaw range through symmetry-aware loss functions or explicit symmetry handling in the network architecture would broaden applicability to scenarios requiring full rotational coverage. Integration of the line laser and camera subsystem for surface refinement or texture mapping would provide complementary information to enhance pose accuracy on objects with subtle geometric features.

8.3 Broader Applications

The developed methodology extends naturally to diverse industrial and robotic applications. In quality control scenarios, the system enables non-contact dimensional verification, surface defect detection, and automated inspection workflows without requiring expensive 3D scanners or manual measurement procedures. For robotic manipulation tasks, real-time pose estimation provides essential feedback for pick-and-place operations, bin picking, and automated assembly systems where object localization accuracy directly impacts task success rates. Autonomous systems including mobile robots and automated guided vehicles can leverage similar sensing and processing architectures for environment perception, obstacle detection, and navigation in structured industrial environments. The combination of affordable sensing hardware with embedded AI processing democratizes access to advanced 3D perception capabilities previously limited to high-cost specialized systems.

8.4 Closing Remarks

This work demonstrates that practical embedded AI systems for 6DOF pose estimation can be developed using cost-effective hardware and innovative scanning methodologies. While the technical path diverged significantly from initial plans due to hardware failures, the adaptive approach yielded a functional system that fulfills core project objectives. The vertical LiDAR slice-stacking methodology, synthetic data generation pipeline, and feature-based deep learning architecture collectively establish a framework applicable to industrial mechatronic systems requiring real-time object localization. The lessons learned regarding system robustness, calibration requirements, and integration challenges provide valuable insights for future embedded AI projects bridging sensing hardware and intelligent processing.

References

- [1] D. F. García et al., "Laser triangulation for shape acquisition," in Handbook of Computer Vision and Applications, vol. 1, pp. 177-226, 1999.
- [2] F. Blais, "Review of 20 years of range sensor development," Journal of Electronic Imaging, vol. 13, no. 1, pp. 231-243, 2004.
- [Figure 2.1 & 2.2]: T. Hermary, "Principles of Laser Triangulation," Hermary, 2024.
[Online]. Available: <https://hermary.com/learning/principles-of-laser-triangulation/>
- [3] S. Thrun, "Robotic mapping: A survey," in Exploring Artificial Intelligence in the New Millennium, Morgan Kaufmann, 2003, pp. 1-35.
- [4] Y. Chen and G. Medioni, "Object modeling by registration of multiple range images," Image and Vision Computing, vol. 10, no. 3, pp. 145-155, 1992.
- [5] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 14, no. 2, pp. 239-256, Feb. 1992.
- [6] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz, "Towards 3D Point cloud based object maps for household environments," Robotics and Autonomous Systems, vol. 56, no. 11, pp. 927-941, 2008.
- [7] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 652-660.
- [8] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2017, pp. 5099-5108.
- [9] Y. Wang et al., "Dynamic graph CNN for learning on point clouds," ACM Trans. on Graphics, vol. 38, no. 5, pp. 1-12, 2019.

[10] S. Peng et al., "PVNet: Pixel-wise voting network for 6DoF pose estimation," in Proc. IEEE/CVF CVPR, 2019, pp. 4561-4570.

[11] P. Warden and D. Situnayake, "TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers," O'Reilly Media, 2019.

[12] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in Proc. IEEE/CVF CVPR, 2018, pp. 2704-2713.

[13] TensorFlow, "TensorFlow Lite for Microcontrollers," 2024. [Online]. Available: <https://www.tensorflow.org/lite/microcontrollers>.

[14] Google, "Coral Edge TPU," 2024. [Online]. Available: <https://coral.ai/products/accelerator>

[15] NVIDIA Corporation, "NVIDIA Jetson Orin Nano Series Technical Brief," Available: <https://developer.nvidia.com/embedded/jetson-orin-nano> [Accessed: Jan. 11, 2025].

[Figure 2.1] Yan, C. (2019, December 20). *Understanding of PointNet network architecture*. TechNotes. <https://yansz.github.io/2019/12/20/PointNet/>

[Figure 2.2] MathWorks. (n.d.). *Get started with PointNet++*. MATLAB & Simulink. Retrieved January 11, 2026, from <https://se.mathworks.com/help/lidar/ug/get-started-pointnetplus.html>

[Figure 3.3] NVIDIA Corporation, "NVIDIA Jetson Orin Nano," 2023. [Online]. Available: <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/>. [Accessed: Jan. 12, 2025].