h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

TEAM PROJECT

UI Development & Hardware Integration for Keyence LJ-X8400

(Electrical Engineering and Information Technology M.Sc.)

Team Members

- Muhammad Zeeshan Yousaf
- Muhammad Hasan Khan
- Utkarsh Mishra
- Kasmin Talukdar
- Amritha Anitha Vasanthan
- Ashwini Joshi

Submitted to: Prof. Dr.-Ing. Sven Rogalski

Table of Contents

Executive summary

The project delivers a browser-based application for the Keyence LJ-X8400 laser profiler that captures live 2D profiles, enables precise 2D measurements, compares profiles side-by-side, stitches multiple scans into 3D surfaces with slice inspection, and exports data to CSV. The architecture uses a FastAPI backend that wraps the sensor interface and a React frontend for visualization and interactive measurement. Compared to vendor software, the app is zero-install, cross-platform, tailored to required measurements, offers a clean data/API pipeline, and is extensible to other industrial sensors with the same UI pattern.

Introduction

In modern industrial automation, quality inspection, and academic research, 2D laser profilers play a critical role in capturing precise surface and dimensional measurements of physical objects. Although manufacturers such as Keyence provide proprietary software for operating these sensors, such tools often impose limitations in terms of integration flexibility, customization, and user interface adaptability.

This project introduces a fully customized, modular graphical user interface (UI) developed specifically for the Keyence LJ-X8400 laser profiler. The system enables real-time data acquisition, visualization, and analysis, with comprehensive support for both 2D and 3D profile rendering, advanced measurement tools, robotic integration, and remote sensor control over Ethernet. Designed with scalability and user experience in mind, the solution was implemented following professional GitLab-based workflows, ensuring robust version control, structured task management, and maintainable development practices.

Objectives

The primary objectives of this project are as follows:

- Design and develop a custom user interface (UI) for the Keyence LJ-X8400 laser profiler to enable efficient visualization, processing, and management of profile data.
- Implement real-time scan capture, on-screen display, and seamless data export functionality.
- Incorporate measurement tools for point-to-point, vertical, and horizontal dimensions to support detailed inspection tasks.
- Achieve 3D reconstruction capabilities by stitching together multiple 2D scans.
- Integrate the sensor with a robot-mounted setup using custom-designed 3D-printed hardware to allow flexible scanning configurations.

- Provide interactive 2D and 3D visualization environments, including slicing tools for detailed analysis of selected sections.
- Utilize GitLab issue tracking for structured team collaboration, progress monitoring, and milestone management.
- Maintain comprehensive documentation of all progress, code, and design work, following professional engineering and software development standards.

Stakeholders and Team Roles

Academic supervisors reviewed progress and safety. The six-person team covered frontend, backend, measurement features (2D, compare), 3D visualization, documentation, and testing. Roles overlapped to ensure continuity, and ownership for wiki pages and components was assigned.

System Overview

The developed system comprises both hardware and software components, designed to work in an integrated and modular manner for efficient sensor operation, data visualization, and analysis.

Hardware Components

Sensor

Keyence LJ-X8400 Laser Profiler

Controller

LJ-X8000

Measuring Range

Z-axis range: ±60 mm around 380 mm reference distance (Full scale 315 mm)

X-axis field: 180 mm (near) / 240 mm (far), extendable up to 320 mm

Resolution

3,200 points per profile; minimum interval 75 µm (optional down to 50 µm)

Maximum Scan Rate

2D mode: up to 1 kHz

3D mode: up to 16 kHz (with compatible controller)

Light Source

Blue semiconductor laser, wavelength 405 nm, output 10 mW

Laser Class

Class 2M (IEC 60825-1, FDA (CDRH) Part 1040.10)

Construction Material

Aluminium housing

Weight

Approx. 1,300 g (2.87 lb)

Interface

Ethernet (TCP/IP)

Output Format

2D line profile (Z-values)

Safety and Compliance

The LJ-X8400 is classified as Class 2M per IEC 60825-1. Operational safety requires labels and signage, keeping the beam path below eye level, and controlling access. Electrical compliance follows IEC 60204-1, with fusing and grounding. EMC compliance follows EN 61000-6-2/-6-4. The sensor head is IP67; the controller should be installed in a protected cabinet. Quality management is ISO 9001; materials comply with RoHS. Maintain calibration certificates and CE documentation.

Custom Mounting Bracket Assembly

To enable integration with a KUKA iiwa robotic arm, a dedicated sensor mounting bracket system was designed and fabricated. This mount ensures the sensor is firmly fixed while allowing correct orientation for scanning.

Sensor Mount

The sensor mount is a precisely machined part designed to fit the Keyence LJ-X8400 body. It includes mounting holes aligned with the sensor's fixture points and a top bracket to attach to an intermediate adapter.

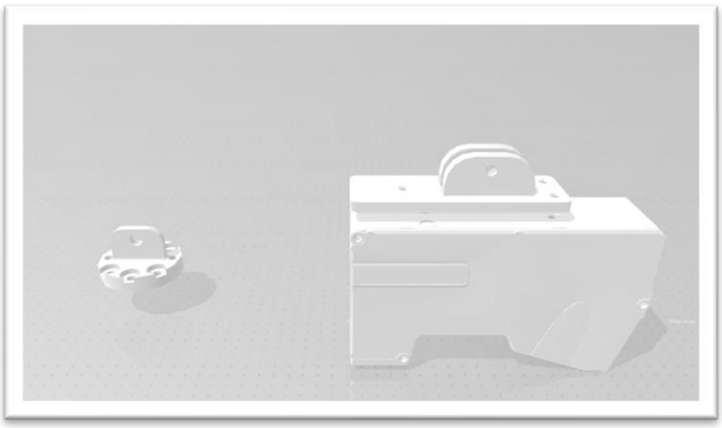


Fig 1: Sensor Mount

Robot Interface Plate

This plate serves as the interface between the sensor mount and the robotic arm's tool flange. It is designed to match the KUKA iiwa bolt pattern and features elongated slots for fine positional adjustment.

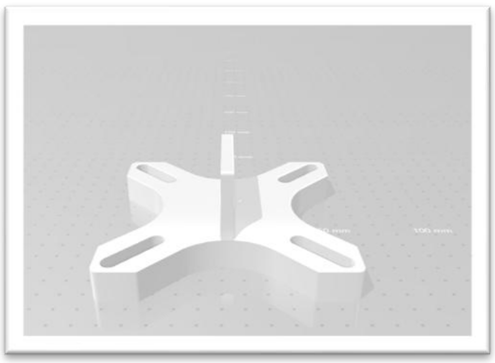


Fig 2: Robot Interface Plate

Both components were designed in Autodesk Fusion 360 and exported as STL files for 3D printing. The parts were manufactured using a Bambu Lab X1 Carbon printer.

Software Components

Frontend (React)

Provides an interactive graphical interface for real-time 2D and 3D visualization, measurement, and control. The frontend is developed using the React framework, providing a modular and responsive interface for real-time data visualization and user interaction. It integrates interactive 2D and 3D chart components, supports zooming, panning, slicing, and measurement modes, and communicates with the backend through REST API and WebSocket channels. The design prioritizes clarity, performance, and adaptability, enabling smooth handling of high frequency laser profile data while maintaining intuitive controls for operators.

Key frontend features include:

- Live profile rendering with millisecond-level update capability.
- Interactive 3D surface visualization with slicing and measurement tools.
- Side-by-side comparison of 2D datasets.
- Configurable settings via modal windows for communication and capture parameters.
- CSV import/export for offline analysis.

Application Architecture and Entry Point

The application starts with App.js, which serves as the central orchestrator that establishes the overall structure and routing system. This component wraps the entire application in a ´´ScanDataProvider´´ context, ensuring that all scan data and UI states are globally accessible throughout the component tree. The App component implements a sophisticated layout system using Flexbox CSS classes, creating a three-section structure consisting of a fixed header at the top, a flexible main content area in the middle, and a collapsible sidebar on the left. The routing system is handled by React Router DOM, which manages client side navigation between four distinct pages without requiring full page reloads. The App component also manages two critical pieces of local state: the sidebar open/closed state and the help modal visibility state, both of which are passed down to child components through props.

The ´´Header´´ component is deliberately minimal but serves an important branding purpose. It displays the application title ´´KEYENCE LJ-X8400 Sensor Control and Data Viewer´´ in a sticky header that remains visible at the top of the screen regardless of scroll position. The header uses CSS classes to ensure it stays above other content with a high z-index value and provides visual separation through borders and shadows.

The ´´Sidebar´´ component implements a sophisticated navigation system that can dynamically collapse and expand. When expanded, it shows full navigation labels alongside icons imported from Heroicons React library. When collapsed, it shows only icons in a compact vertical arrangement. The sidebar uses React Router's NavLink component to provide active state management, automatically highlighting the current page with different background colours. The navigation includes four main sections: Live Data (home icon), 3D Data (cube icon), 2D Captured Data (squares icon), and Compare 2D Data (arrows icon). Additionally, the sidebar includes a help button that triggers the global help modal and a collapse/expand toggle button positioned on the right edge that allows users to maximize their viewing area when needed.

Live Data Page

The ´´LiveDataPage´´ component serves as a minimal wrapper that renders the ´´LiveProfileCapture´´ component, which contains all the sophisticated real-time functionality. This design pattern allows for future expansion where additional controls or information displays could be added to the page without modifying the core capture functionality.

The ´´LiveProfileCapture´´ component is the most complex component in the application, handling real-time sensor communication, data visualization, and capture functionality. This component manages multiple interconnected state variables including connection status, sensor configuration parameters (IP address, port, polling rate, sample interval), real-time profile data, capture state (capturing, paused), zoom and pan settings, and accumulated scan data for export. The component implements a sophisticated polling mechanism that continuously requests profile data from the backend API when the sensor is connected, with configurable intervals that default to 200 milliseconds.

The data transformation pipeline within LiveProfileCapture is particularly noteworthy. When raw sensor data arrives from the backend, it undergoes several processing steps: X coordinates are converted from sensor units to millimetres and centred around a 120mm midpoint, Z coordinates are normalized by subtracting the minimum value to ensure all heights start from zero, and the processed data is stored in React state to trigger re-rendering of the visualization. The SVG-based chart system uses mathematical coordinate transformations to convert data coordinates into screen pixels, implementing zoom and pan functionality through dynamic viewport calculations.

The capture functionality operates independently of the real-time display, using a separate interval timer that saves profile snapshots at configurable intervals (default 10ms) when capture mode is active. The captured data is stored as an array of objects containing timestamps and Z-value arrays, which can be exported as CSV files when the user stops capturing. The CSV export functionality creates properly formatted files with headers and handles data serialization automatically.

The ´´SettingsModal´´ component, nested within LiveProfileCapture, provides a clean interface for configuring sensor connection parameters. This modal implements pending state management, where changes are held in temporary state variables until the user explicitly applies them by clicking the "Set" button. The modal includes connection and disconnection controls with appropriate visual feedback and disabled states based on current connection status.

3D Data Page

The ´´ThreeDDataPage´´ component creates a sophisticated dual-pane interface combining 3D surface visualization with 2D profile analysis. This page implements a grid-based layout system that responsively adapts to different screen sizes, showing the 3D viewer on the left and a 2D profile viewer on the right. The page manages local state for slice data and measurement parameters, creating an isolated measurement environment that doesn't interfere with other pages' measurement states.

The component establishes communication between the ´´Viewer3D´´ and ´´ProfileViewer´´ components through callback functions. When users interact with slice controls in the 3D viewer, the onSliceChange callback updates the slice data and axis information, which is then formatted into a compatible data structure for the 2D viewer. This integration allows users to examine specific cross-sections of 3D surface data in detail using the full suite of 2D measurement tools.

The Viewer3D component leverages React Three Fiber and Three.js to create an interactive 3D environment. This component manages complex 3D surface reconstruction from CSV data, implementing efficient geometry generation using Three.js BufferGeometry for optimal performance. The surface data is processed into vertex positions, colours based on height values, and triangle indices for mesh rendering. The component includes sophisticated colour mapping that applies HSL colour gradients based on height values, creating intuitive visual representations where different colours represent different elevation levels.

The 3D viewer implements slice plane visualization using semi-transparent box geometries that can be positioned along X or Y axes. Users can control slice position through slider controls, and the slice plane serves as a visual guide for understanding which data is being analysed in the 2D viewer. The OrbitControls integration provides smooth camera manipulation with mouse interactions for rotating, zooming, and panning the 3D view. The component also includes a camera reset functionality that returns the view to a default isometric perspective.

Captured Data Page

The ´´CapturedDataPage´´ component provides a comprehensive interface for analysing previously captured scan data. This page heavily leverages the ´´ProfileViewer´´ component while integrating with the global ´´ScanDataContext´´ to maintain captured data and measurement states across navigation events. The page implements a clean, centred layout that maximizes the available space for data visualization and analysis.

The component retrieves multiple state variables from the ScanDataContext, including the captured scans array, current scan index, zoom and pan settings, measurement states, and measurement handle positions. This integration ensures that users can navigate away from the page and return without losing their analysis progress. The page includes an optional scan slider that appears when multiple scans are loaded, allowing users to navigate through captured data sets with visual feedback showing the current scan number and total count.

The ProfileViewer integration includes the ´´withFileLoader´´ and ´´withMeasurementDisplay´´ flags, enabling full functionality for loading CSV files and accessing advanced measurement features. The onLoad callback ensures that newly loaded data is properly integrated into the global state management system, making it available for other components that might need access to captured data.

Compare Data Page

The ´´CompareDataPage´´ component implements the most sophisticated multi-viewer system in the application, allowing users to perform detailed comparative analysis between two different scan datasets. This page manages an extensive set of state variables through the ScanDataContext, maintaining separate but potentially synchronized states for left and right viewers including scan data, current scan indices, zoom levels, pan positions, measurement states, and handle positions.

The component's most innovative feature is the lock/unlock functionality that allows users to synchronize interactions between the two viewers. When locked, actions performed on one viewer (zooming, panning, moving measurement handles, toggling measurement modes) are automatically mirrored on the other viewer. This synchronization is implemented through higher-order functions that wrap the individual setter functions, creating synchronized updates when lock mode is active.

The layout system uses a responsive design that accommodates various screen sizes while ensuring both viewers remain fully functional. The component includes individual scan sliders for each viewer when multiple scans are loaded, allowing independent navigation through different datasets. Each ProfileViewer instance operates with its own measurement system, enabling direct comparison of measurements between different scans or different positions within the same scan dataset.

ProfileViewer Component

The ProfileViewer component serves as the foundational visualization and measurement system used across multiple pages. This component implements a sophisticated SVG-based charting system with advanced interaction capabilities including mouse-based measurement handle manipulation, wheel-based zooming with cursor-following behaviour, drag-based panning with momentum, and point-and-click measurement functionality.

The component's measurement system includes two distinct modes: handle-based measuring with draggable red vertical lines and orange horizontal lines that can be positioned anywhere on the chart, and point-based measuring where users click two points on the curve to analyse the path between them. The measurement calculations are comprehensive, including horizontal distances in millimetres, vertical distances in millimetres, curved path distances following the actual profile shape, straight-line distances between points, angular measurements between selected points, and minimum/maximum height values within selected ranges.

The coordinate transformation system implements mathematical conversions between data coordinates (millimetres) and screen coordinates (pixels), handling dynamic zoom and pan operations while maintaining measurement accuracy. The component includes sophisticated viewport management that ensures measurement handles remain visible during zoom operations by automatically adjusting pan positions when necessary.

The CSV file loading functionality includes robust parsing that handles various CSV formats, extracting timestamp information and Z-value arrays from properly formatted files. The component's export capabilities can generate new CSV files from current data states, enabling data sharing and external analysis workflows.

ScanDataContext

The ´´ScanDataContext´´ component implements a comprehensive global state management system using React's Context API, avoiding the complexity of external state management libraries while providing centralized data access. This context manages multiple categories of state including surface data for 3D visualization, scan collections for different pages (captured, compare left/right, 3D), measurement states for each page/viewer combination, and UI states like zoom levels, pan positions, and current scan indices.

The context ensures that measurement states are preserved across navigation events, allowing users to maintain their analysis progress when switching between pages. The separation of state categories prevents cross-contamination between different analysis workflows while enabling data sharing where appropriate. The context also implements selective persistence, maintaining certain settings in local Storage while keeping measurement data in memory for performance reasons.

Help Centre System

The ´´HelpCenter´´ component provides context-sensitive documentation through a modal interface that can be accessed from any page. This component implements a tabbed interface that matches the main navigation structure, providing relevant help content for each page. The help system includes detailed explanations of measurement tools, navigation controls, data loading procedures, and advanced features like 3D slice analysis and compare mode synchronization.

This comprehensive frontend architecture creates a professional-grade laser profiling application with real-time data acquisition, sophisticated measurement capabilities, 3D visualization, and comparative analysis tools, all built using modern React development practices and responsive design principles.

Backend (FastAPI)

The backend serves as the critical bridge between the high-precision sensor hardware and the interactive, user-facing frontend. Built with a modern, high-performance technology stack, its primary role is to manage sensor communications, process high-frequency data streams in real-time, and expose a robust, well-defined API for client interaction. The architecture is designed around a three-tier model, ensuring a clear separation of concerns between presentation, business logic, and data access. The core of the backend is implemented using the ´´FastAPI´´ framework in Python, chosen specifically for its asynchronous capabilities which are essential for handling the non-blocking, I/O-bound operations inherent in real-time sensor data acquisition. This report details the system's architecture, the rationale behind key technology choices, the layered design, the specific functionalities provided to the frontend, and considerations for deployment, error handling, and future scalability. The result is a backend system that is not only powerful and efficient but also maintainable and extensible for future industrial and research applications.

Backend Architecture Overview

The application follows a classic Client-Server architecture. The frontend (Client) is a browser based React application, and the backend (Server) is a standalone Python application. The two communicate over the network via standard web protocols. This decoupled design offers significant advantages in terms of development, deployment, and scalability.

Architectural Model

The backend is logically separated from the Keyence LJ-X8400 controller and the React frontend. This separation ensures that changes in one component have minimal impact on the others.

- Keyence Controller: The hardware responsible for operating the laser sensor and providing the raw profile data over an Ethernet connection.
- Backend Server (FastAPI): The central processing unit of the system. It connects directly to the controller, requests and receives data, processes it, and serves it to the frontend.
- Frontend Client (React): The user interface running in the user's web browser. It is responsible for all data visualization and user interaction, making requests to the backend for data and control commands.

The flow of data and control is as follows: The Backend establishes a TCP/IP connection to the Keyence Controller to receive raw scan data. The Frontend communicates with the Backend using two primary methods: a REST API for stateful commands (like connecting/disconnecting) and a WebSocket for a persistent, low-latency connection to stream live profile data for real-time visualization.

Technology Stack and Rationale

The choice of technology is critical for a system that demands real-time performance and reliability. The backend stack was selected to meet these specific requirements.

Core Technologies

Programming Language: Python 3.10+

Python's extensive ecosystem of libraries, strong community support, and readability make it ideal for rapid development. Libraries like ctypes are crucial for interfacing with the C-based Keyence SDK.

Web Framework: FastAPI

This was a key strategic choice. FastAPI is a modern, high-performance web framework for Python based on standard type hints. Its benefits are detailed in the next section.

ASGI Server: Uvicorn

Uvicorn is a lightning-fast Asynchronous Server Gateway Interface (ASGI) server, built on uvloop and httptools. It is the recommended server for running FastAPI applications and is essential for realizing the performance benefits of the asynchronous framework.

Hardware Interfacing: ctypes Library

The official Keyence SDK is provided as a set of DLLs (Dynamic Link Libraries). The ctypes library is a standard Python foreign function interface that allows Python code to call functions in these DLLs directly, enabling low-level control of the sensor without needing to write C/C++ code.

Why FastAPI?

For an application centred around real-time data streaming, the choice of web framework is paramount. Traditional synchronous frameworks like Flask or Django can struggle with long-lived connections and I/O-bound tasks, as they typically handle one request per worker process at a time. FastAPI was chosen over these alternatives for several compelling reasons:

Asynchronous by Design: FastAPI is built on top of Starlette and Pydantic and fully supports asynchronous code using Python's async/await syntax. This is the single most important feature for our use case. When the backend is waiting for new data from the sensor over the network (an I/O-bound operation), an async framework can yield control and handle other tasks, such as responding to another API request from the frontend. This prevents the entire server from being blocked while waiting for data, leading to significantly higher throughput and responsiveness.

High Performance: Thanks to its asynchronous nature and the underlying performance of Starlette and Uvicorn, FastAPI is one of the fastest Python frameworks available, rivalling the performance of NodeJS and Go. This ensures that the backend can handle the high data rates from the Keyence sensor (up to 16 kHz) without becoming a bottleneck.

Native WebSocket Support: WebSockets are a first-class citizen in FastAPI. Implementing the real-time data streaming endpoint (/ws/live_profile) is straightforward and efficient, allowing for a persistent, bidirectional communication channel between the server and the client with minimal overhead.

Automatic API Documentation: FastAPI automatically generates interactive API documentation (using Swagger UI and ReDoc) from the code's type hints. This is invaluable for development, debugging, and for any future developers who need to understand and interact with the API.

Data Validation and Serialization: By using Pydantic, FastAPI provides robust data validation. You can define the expected data shapes for incoming requests, and FastAPI will automatically parse, validate, and document them. This drastically reduces the amount of boilerplate code needed for data validation and helps prevent bugs.

Backend Architectural Layers

To ensure maintainability and a clear separation of concerns, the backend is structured into three distinct logical layers: the API Layer, the Business Logic Layer, and the Data Access Layer.

API Layer

This is the outermost layer and the sole entry point for the frontend client. It is responsible for handling all incoming HTTP requests and WebSocket connections. Its duties include defining API routes, validating incoming data, serializing response data into JSON, and managing WebSocket lifecycle events before delegating work to the Business Logic Layer.

Business Logic / Service Layer

This layer contains the core application logic. It is completely decoupled from the web framework and the specific details of sensor communication. Its responsibilities include maintaining the application's state (e.g., connection status), implementing core features like data capture and 3D stitching, and coordinating calls to the Data Access Layer.

Data Access / Communication Layer

This is the lowest-level layer, responsible for all direct interaction with the Keyence hardware. It wraps the Keyence SDK functions, handles the TCP/IP socket connection, sends low-level commands to the sensor, and parses the raw incoming data into a usable Python data structure.

Detailed Backend Functionality

This section elaborates on the backend services required to power the features described in the UI workflow.

Sensor Connection Management

The user initiates a connection from the UI, sending a POST request to the /connect endpoint. The API layer validates this, the Business Logic layer orchestrates the command, and the Data Access Layer uses the SDK wrapper to establish the connection. The result is propagated back to the frontend, which polls a /status endpoint to update the UI.

Real-Time Data Streaming via WebSocket

Upon a successful connection, the frontend connects to the /ws/live profile WebSocket endpoint. The backend then starts an asynchronous loop, continuously fetching the latest profile data from the sensor via the Data Access Layer and streaming it through the WebSocket to the client for live rendering.

Data Capture and CSV Export

When the user starts a capture, a flag is set in the Business Logic Layer. During the live stream, profiles are saved to an in-memory buffer. When the user stops the capture and requests an export, the backend formats the buffered data into a CSV file and sends it to the client as a file download attachment.

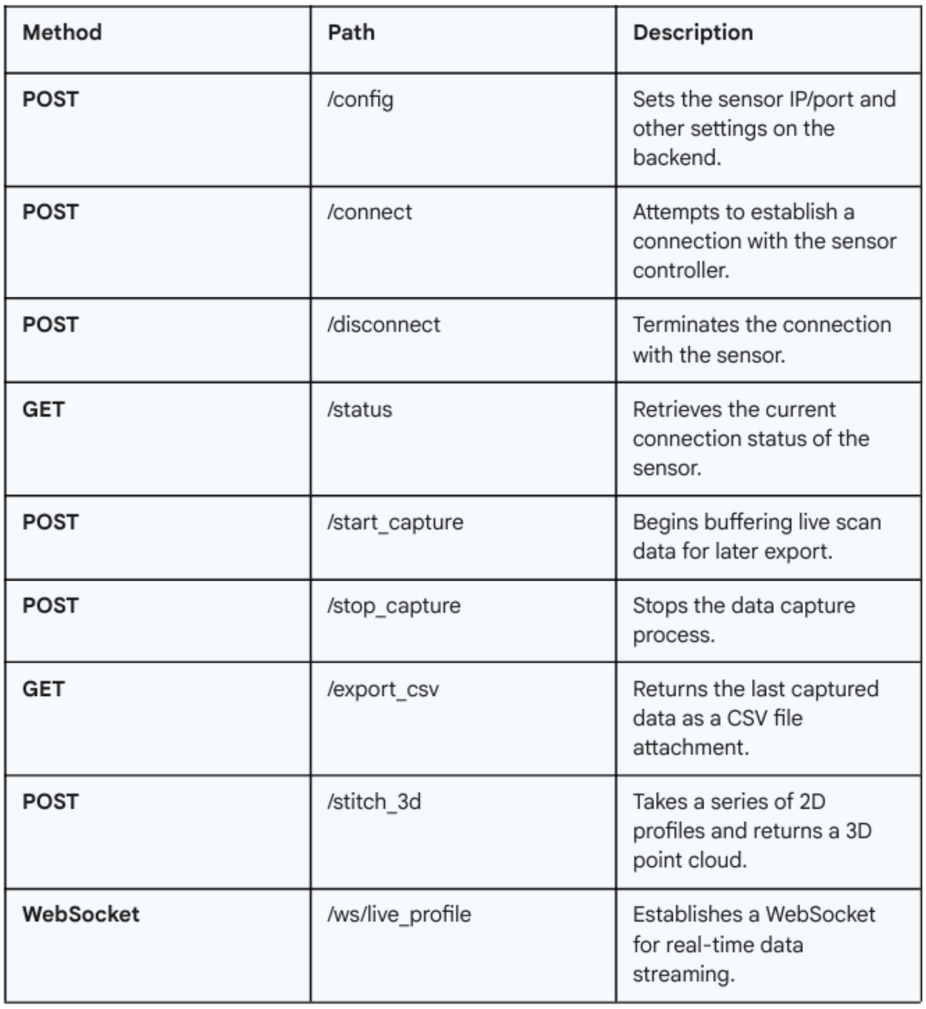
3D Surface Reconstruction (Stitching)

The backend receives a set of captured 2D profiles at a /stitch_3d endpoint. The Business Logic Layer runs a stitching algorithm, which assigns a Y-coordinate to each 2D (X,Z) profile, effectively creating a 3D point cloud. This point cloud data is then returned to the frontend for rendering.

Data Transformation: Shifting to Positive Axis

To simplify frontend rendering, the backend provides a transformation service. After a profile is fetched, a function in the Business Logic Layer finds the minimum X and Z coordinates and calculates an offset. This offset is then added to every point in the profile, shifting the entire dataset into the positive quadrant before it is sent to the client.

API Endpoint Definitions



Method	Path	Description
POST	/config	Sets the sensor IP/port and other settings on the backend.
POST	/connect	Attempts to establish a connection with the sensor controller.
POST	/disconnect	Terminates the connection with the sensor.
GET	/status	Retrieves the current connection status of the sensor.
POST	/start_capture	Begins buffering live scan data for later export.
POST	/stop_capture	Stops the data capture process.
GET	/export_csv	Returns the last captured data as a CSV file attachment.
POST	/stitch_3d	Takes a series of 2D profiles and returns a 3D point cloud.
WebSocket	/ws/live_profile	Establishes a WebSocket for real-time data streaming.

Error Handling and Connection Recovery

A robust backend must gracefully handle expected and unexpected failures.

Connection Errors: If the backend fails to connect to the sensor (e.g., wrong IP, network issue), the /connect endpoint will return an error status with a descriptive message, which the frontend can display to the user.

Data Stream Interruption: During live streaming, the connection to the sensor might drop. The Data Access Layer is responsible for detecting this. It will then notify the Business Logic Layer, which will update the application state to is connected = False. This status change will be reflected in the next /status poll, and the WebSocket may be closed with an appropriate error code.

Automatic Reconnection: A future enhancement could involve the backend attempting to automatically reconnect to the sensor in the background for a certain number of retries before declaring the connection lost.

Invalid API Requests: FastAPI's Pydantic integration automatically handles validation. If the frontend sends a request with missing or malformed data, the API will automatically return a 422 Unprocessable Entity response with details about the validation error.

Deployment and Scalability

Deployment: The backend is a standard Python application. It can be deployed using Docker for containerization, which simplifies dependency management and ensures a consistent environment. A production deployment would typically involve running the Uvicorn server behind a reverse proxy like NGINX for handling HTTPS and load balancing.

Scalability:

Vertical Scaling: The current single-instance architecture can be scaled vertically by running it on a more powerful server with more CPU cores and memory.

Horizontal Scaling: While the current application is designed for a single sensor, the architecture could be extended to support multiple sensors. This would require more complex state management, potentially introducing a message broker or a shared cache to manage communication between multiple backend instances. For the scope of this project, a single backend instance is sufficient.

Data Handling and Processing

Sensor Measurement Range and Normalization

The Keyence LJ-X8400 laser profiler has a Z-axis measurement range of ±60 mm around its reference distance of 380 mm, giving a total vertical span of 120 mm. This means the sensor can detect surfaces located up to 60 mm above and 60 mm below the reference distance.

When installed in our test setup, the sensor was mounted such that the minimum detected height corresponded to –43 mm within this operating range. Consequently, raw measurement data contained negative Z-values for positions below the reference plane.

To standardize the measurement display and simplify interpretation, a normalization offset was applied in the software. Specifically, the –43 mm mounting offset was added to all incoming measurement values. This adjustment ensures that:

The minimum height at the mounted position corresponds to 0 mm.

All displayed values fall within a strictly positive range.

The full sensor dynamic range (up to +60 mm above reference) is preserved.

This normalization step aligns the data representation with the sensor's actual installation geometry and provides a consistent basis for subsequent measurement, visualization, and 3D reconstruction tasks.

Dead Zone Handling and Interpolation

During scanning, the laser profiler occasionally encounters regions where the emitted beam cannot be reflected to the sensor, typically because the surface is blocked by a sharp edge, hidden feature, or steep geometry. These regions are known as dead zones. In a dead zone, the sensor cannot provide a valid measurement and instead may output irregular or noisy values. If left unprocessed, these artifacts distort the reconstructed 3D surface by introducing unrealistic spikes or voids.

To ensure reliable reconstruction, a clipping strategy was implemented. All measurement values falling below the calibrated baseline of 0 mm (after normalization) were clipped and excluded from the dataset. In our mounted configuration, this effectively removed data below –43 mm in raw terms. By discarding these invalid points, the 3D reconstruction is based solely on valid surface measurements, eliminating dead-zone artifacts and producing a cleaner, more accurate surface representation.

This approach provides a practical balance: dead zones are not interpolated artificially, which could introduce false geometry, but instead are filtered out, ensuring the reconstructed object faithfully represents only the visible portions of the surface.

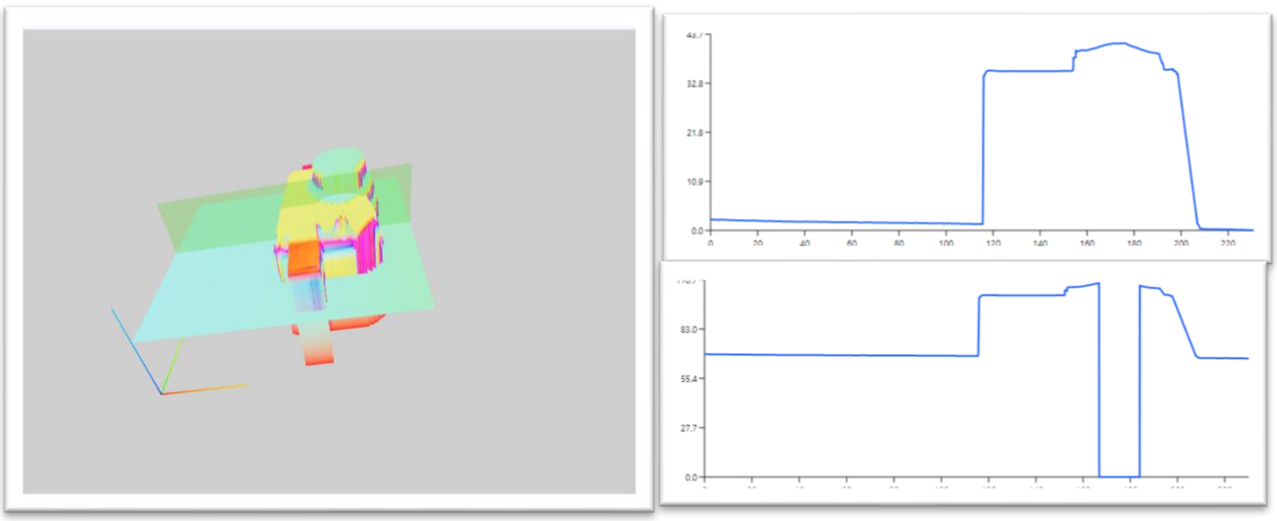


Fig 3: Dead Zone Handling

Filtering and Interpolation Mechanisms

Accurate profiling depends not only on the sensor's raw optical capability but also on the data processing functions that refine and stabilize measurements.

The laser profiler applies a combination of filtering and interpolation mechanisms to achieve consistent results across different measurement conditions.

Horizontal Interpolation

The sensor captures thousands of discrete points along each profile line. At edges or partially hidden surfaces, gaps may appear in the dataset. Horizontal interpolation fills these gaps by estimating missing values from neighboring data points. This produces smoother cross-sections and ensures continuity of the profile, particularly when reconstructing full 3D surfaces.

Vertical Filtering (Spike Suppression)

Sudden noise spikes can occur due to reflections from glossy surfaces, edges, or ambient interference. Vertical filtering suppresses such isolated spikes without altering the true surface geometry. This results in stable height measurements and prevents false peaks in the dataset.

Measurement Range and Limits

The sensor operates within a Z-axis range of ±60 mm around its reference distance, with repeatability on the order of a few microns. The X-axis field spans up to 240 mm, providing dense point coverage across each scan. While interpolation and filtering enhance data reliability, they are bounded by these physical measurement limits. Surfaces outside the valid range cannot be corrected by software and must be managed by proper sensor placement.

Together, these mechanisms ensure that the displayed data is smooth, stable, and physically valid. Interpolation restores continuity where data gaps occur, filtering removes outliers that could distort the profile, and clipping ensures that only measurements within the valid dynamic range are used. These steps form the basis for high-quality 2D visualization and accurate 3D reconstruction in the application.

Measurement Algorithms

Rulers

Two vertical rulers define ΔX, two horizontal rulers define ΔZ. Z@A and Z@B are interpolated where vertical rulers intersect the polyline.

Two-point measurements

Click points A and B on the curve. Straight distance is the Euclidean distance between A and B. Path distance is the sum of segment lengths along the polyline from A to B. Angle is atan2 of ΔZ over ΔX in degrees. Visible min and max Z are computed over the clipped X window.

Sensor Communication Layer (Python SDK Wrapper)

Interfaces with the Keyence SDK for data acquisition and sensor control over Ethernet.

GitLab Workflow

Used for collaborative development, version control, and issue tracking.

Application Data Flow

The overall data flow of the application begins at the sensor and ends with processed measurements being stored and visualized across multiple pages. Each stage in this flow contributes to converting raw laser measurements into usable height profiles and 3D reconstructions.

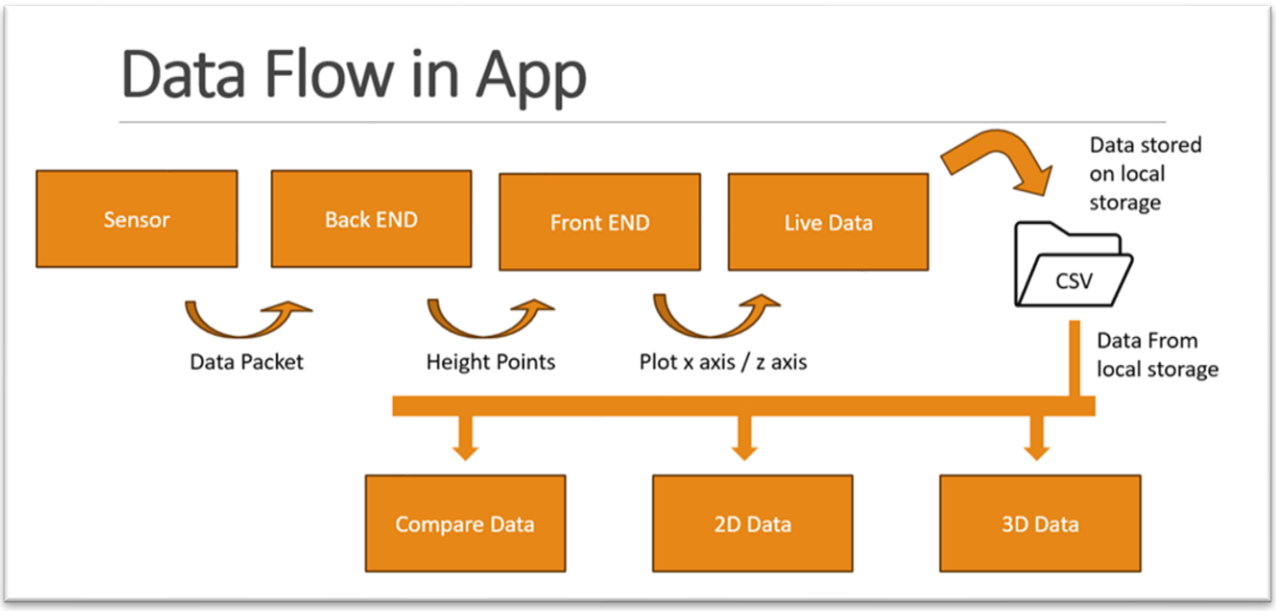


Fig 4: Data Flow

Sensor

The Keyence LJ-X8400 captures a line profile by emitting a laser beam and detecting reflected light. This raw information is packaged as a data packet containing intensity and height information for each sampled point across the X-axis.

Backend

The backend receives the data packets over Ethernet. Using the Keyence SDK wrapper, the raw sensor data is decoded into height points (X–Z coordinates). The backend then streams this information to the frontend via WebSocket, ensuring real-time performance.

Frontend

The frontend processes the incoming height points and plots them along the X-axis and Z-axis in the user interface. This is the first stage where the user can directly observe the live profile of the scanned surface.

Live Data Page

The live data view continuously updates with new incoming profiles. At this stage, users can start, pause, or stop the stream, zoom into the plot, or export the data. Data captured in this mode can be saved locally in CSV format for further processing.

Local Storage (CSV Export and Import)

Exported CSV files serve as a bridge between live acquisition and offline analysis. Once stored, these files can be reloaded into the application at any time, ensuring that scans can be preserved, compared, or shared without requiring continuous connection to the sensor.

Data Utilization in Application Pages

Compare Data Page: Loads two separate datasets (either live or from CSV) side by side to compare profiles, allowing synchronized or independent measurements.

2D Data Page: Displays a single saved scan with measurement tools for distances, angles, and profile ranges.

3D Data Page: Uses multiple 2D profiles to reconstruct a 3D surface, enabling slicing and cross-sectional inspection.

This flow ensures that raw sensor outputs are progressively transformed into structured, interpretable visualizations. By separating live acquisition from CSV-based storage, the application supports both real-time monitoring and offline analysis in a unified workflow.

User Interface Structure

The user interface is divided into four main pages

Live Data Page: Real-time profile capture and visualization

The Live Data page enables real-time acquisition and visualization of profile data from the Keyence LJ-X8400 laser profiler. Incoming scan data is streamed from the backend via WebSocket, ensuring minimal latency between measurement and display. The page provides a dynamic 2D profile chart that updates continuously, allowing operators to monitor surface geometry in real time. Users can initiate or stop live capture, adjust display parameters, and prepare data for further analysis or export. This functionality is essential for on-the-fly inspection, process monitoring, and immediate detection of surface anomalies.

The Live Data Page serves as the primary interface for real-time laser profile visualization. It allows users to:

- Establish a connection with the Keyence LJ-X8400 sensor.
- View live scan data rendered as a continuously updating 2D line chart.
- Capture individual scans or perform continuous scan acquisition.
- Export acquired data for subsequent analysis or archiving.
- Configure device communication parameters and capture settings for optimal performance.

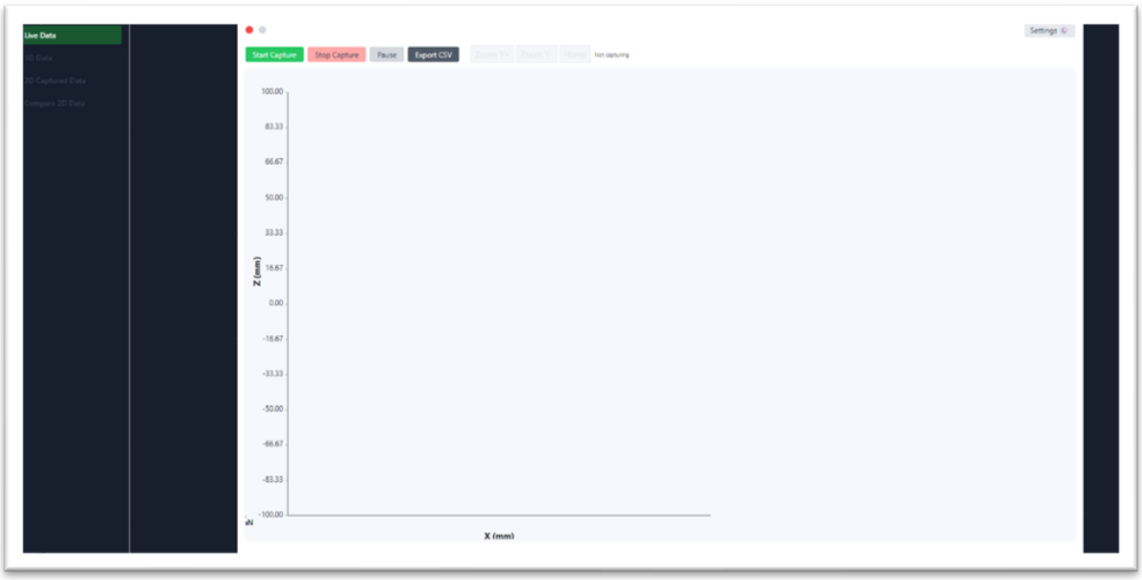


Fig 5: Live Data Page

Button and UI Behaviour

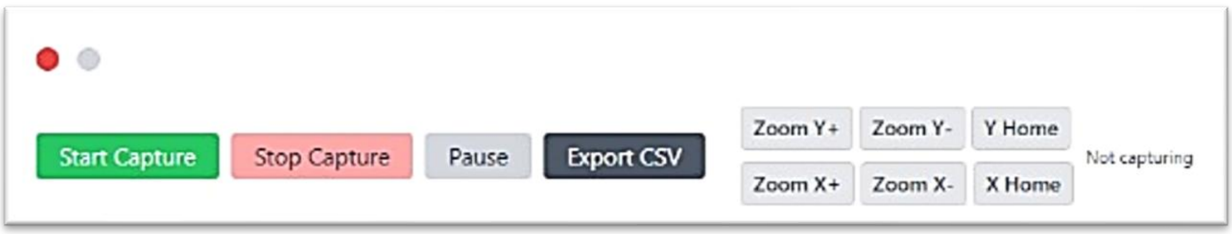


Fig 6: Sensor Connection Status and Control Buttons

The top bar of the Live Data Page provides key controls, status indicators, and configuration access for managing the real-time capture process

Sensor Connection Indicator (LED)

Displays the sensor’s connectivity status: green when connected, red when disconnected.

Data Arrival Indicator (LED)

Flashes yellow whenever new scan data is received from the sensor.

Control Buttons

Start capture

Initiates real-time scan acquisition.

Stop capture

Ends the ongoing scan session.

Pause

Temporarily halts live data updates without disconnecting from the sensor.

Export CSV

Saves captured scan data to a CSV file for later analysis.

Zoom Y+ / Zoom Y-

Adjusts the vertical scale of the 2D profile display for detailed inspection.

Home

Resets the display to its default view and scale.

Settings Button

Opens a configuration modal where users can adjust communication settings (e.g., IP, port), capture parameters (e.g., scan rate, resolution), and other operational preferences.

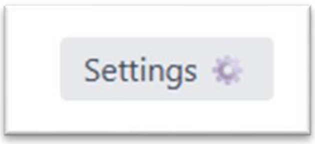
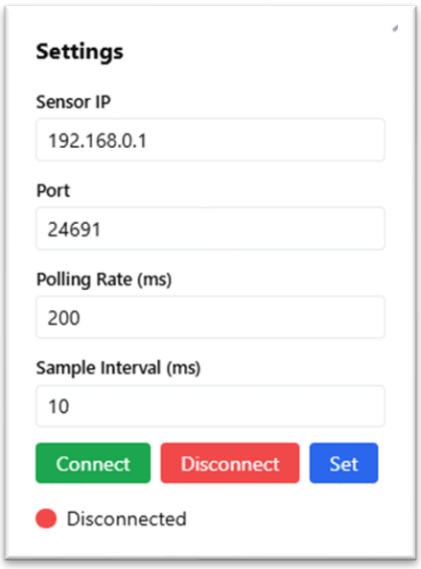


Fig 7: Settings Button

Setting Model

- IP Field: Sets sensor IP address
- Port Field: Sets sensor port number
- Polling Rate: Data fetch interval from sensor
- Sample Interval: Save interval during capture
- Set: Updates IP/port on backend
- Connect: Connects to sensor
- Disconnect: Disconnects sensor
- Status LED: Shows "Connected" or "Disconnected"


Settings

Sensor IP: 192.168.0.1

Port: 24691

Polling Rate (ms): 200

Sample Interval (ms): 10

● Disconnected

Fig 8: Configuration Settings

3D Data Page: 3D surface reconstruction and slicing

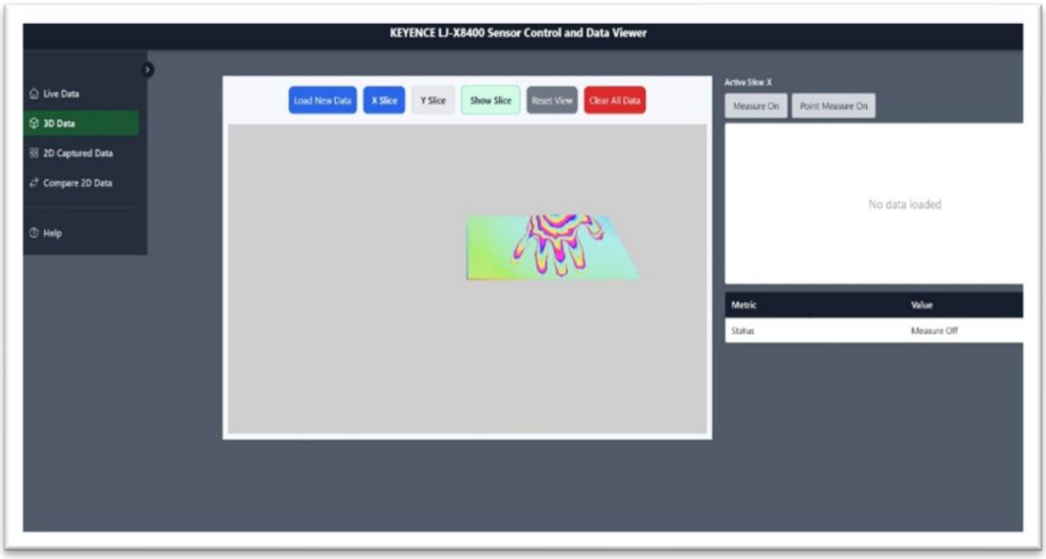
The 3D Data Page delivers an interactive 3D surface visualization of captured scan data, enabling comprehensive inspection of reconstructed objects. Users can freely rotate, zoom, and pan the view, as well as apply slicing tools along the X or Z axis to examine cross-sections in detail. The interface supports loading CSV-based 3D surface profiles and provides integrated measurement tools for accurately determining profile dimensions directly within the visualization.

3D Data Page Workflow

Page layout and initial state

The 3D Data Page features a central 3D viewport accompanied by a top control bar containing the following options:

Load New Data, X Slice, Y Slice, Hide/Show Slice, Reset View*,* and Clear All Data. The right-hand panel includes a slice chart display area and a metrics table. In the initial state, before any dataset is loaded, the chart area displays the message “No data loaded.”

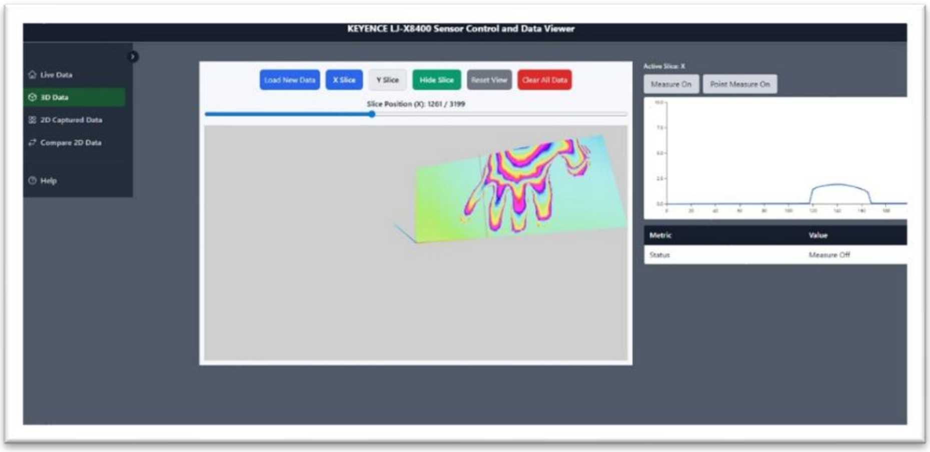


Metric	Value
Status	Measure Off

Fig 9: 3D Data Page after loading a CSV surface; controls, viewport, and side panel layout.

Load a 3D Surface

Click Load New Data to import a CSV based 3D surface profile. Once loaded, the reconstructed surface is displayed in the central viewport. Basic navigation controls include rotate or orbit using mouse drag, zoom using the scroll wheel, and pan using right-click drag or two-finger drag on a touchpad.

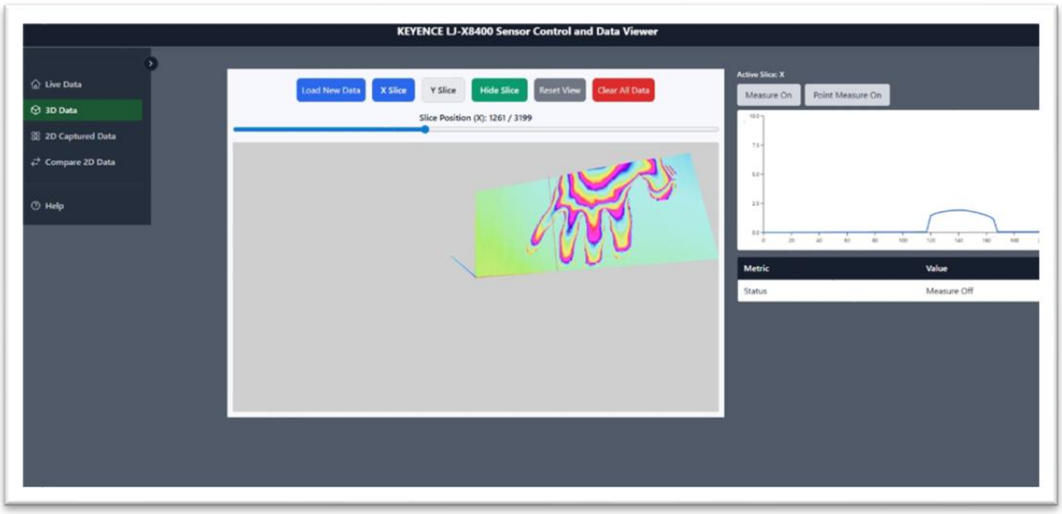


Metric	Value
Status	Measure Off

Fig 10: 3D X Slice active with position slider and X-cross-section shown in the chart.

Inspect an X-Slice

Click X Slice to insert a vertical slicing plane along the X-axis. A position slider labelled Slice Position (X) appears, allowing the user to move the slicing plane across the surface. The right-hand 2D chart displays the cross-section corresponding to the current X position, with the header indicating Active Slice: X. Use the Show/Hide Slice option to toggle the visibility of the slicing plane without altering its position.

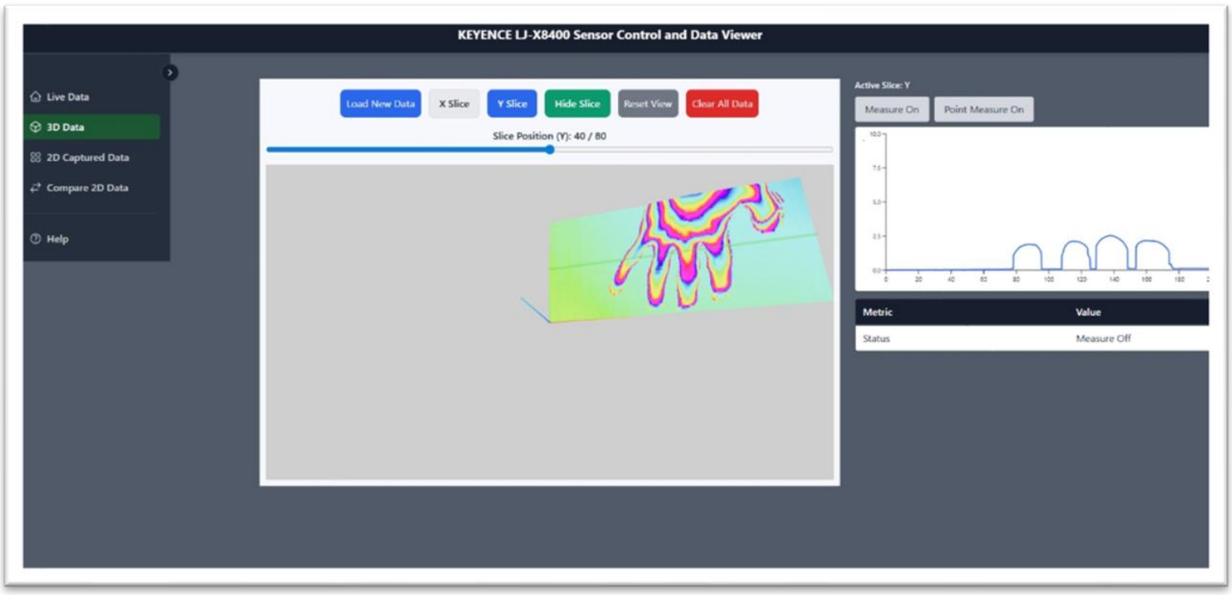


Metric	Value
Status	Measure Off

Fig 11: 3D Y Slice active with position slider and Y-cross-section in the chart.

Inspect a Y-slice

Click Y Slice to switch the slicing plane to the Y axis. Use the Slice Position (Y) slider to move the plane along Y. The 2D chart updates live with the current Y cross-section and the header shows “Active Slice: Y”.



Metric	Value
Status	Measure Off

Fig 12: 3D Y Slice view while scanning through the object to inspect multiple cross-sections.

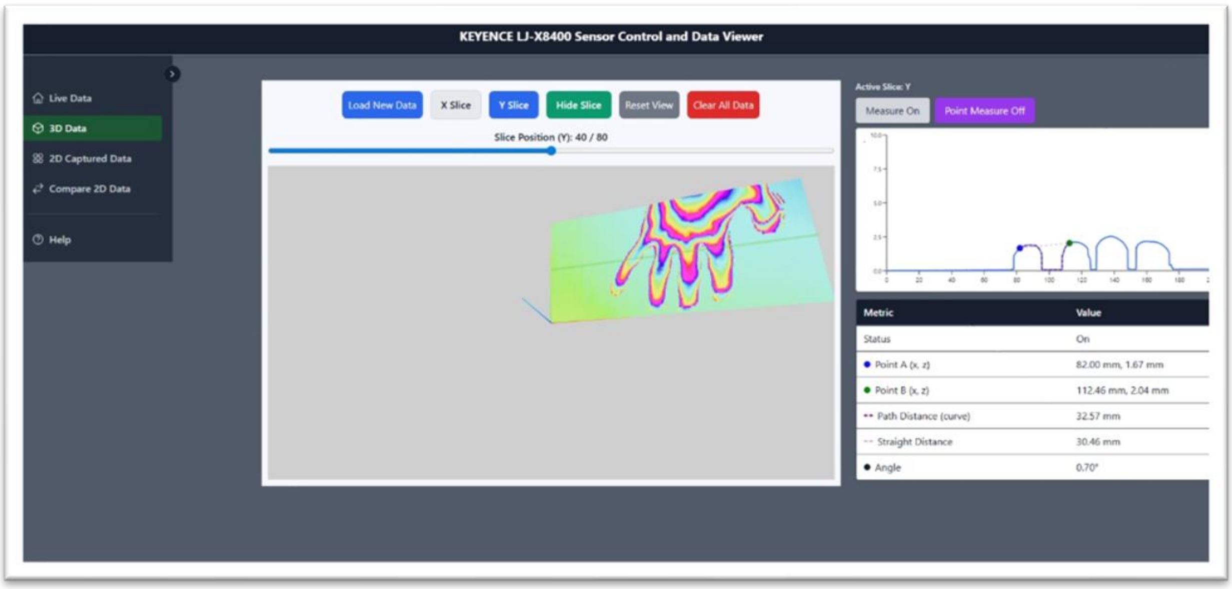
Measurement mode on a slice

Toggle Measure On to enable ruler-based measurements on the 2D slice chart.

The metrics table populates with:

- Visible X (mm) – current horizontal span.
- ΔX (red lines) – distance between the two vertical rulers.
- Z@A / Z@B – profile heights at each vertical ruler (with linear interpolation if between.
- ΔZ (orange lines) – vertical difference between the two horizontal rulers.
- Path Distance (curve) – cumulative distance along the profile between A and B.
- Z Min / Max (range) – min and max height within the visible region.

Toggle Measure Off to return to view-only mode.



Metric	Value
Status	On
● Point A (x, z)	82.00 mm, 1.67 mm
● Point B (x, z)	112.46 mm, 2.04 mm
Path Distance (curve)	32.57 mm
Straight Distance	30.46 mm
● Angle	0.70°

Fig 13: 3D Measurement mode enabled on the slice chart; metrics table populated with ΔX, ΔZ, Z@A/Z@B, path distance, and range.

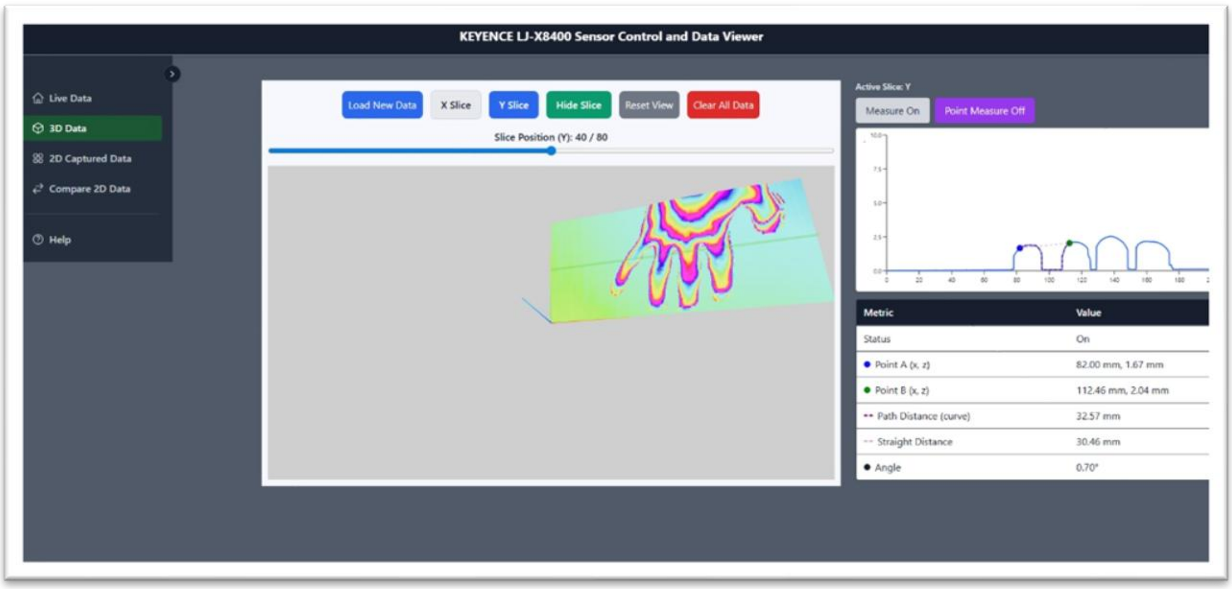
Point to point measurements

Toggle Point Measure On to select two arbitrary points on the slice curve.

The metrics table reports:

- Point A (x, z) and Point B (x, z) coordinates.
- Path Distance (curve) between A and B along the profile.
- Straight Distance (Euclidean) between A and B.
- Angle: atan2(Δz, Δx) expressed in degrees.

A dotted segment on the chart highlights the measured path.



Metric	Value
Status	On
● Point A (x, z)	82.00 mm, 1.67 mm
● Point B (x, z)	112.46 mm, 2.04 mm
•• Path Distance (curve)	32.57 mm
-- Straight Distance	30.46 mm
● Angle	0.70°

Fig 14: 3D Point-to-point measurement on the slice; points A and B with straight distance, path distance, and angle.

Reset and clear

Reset View restores the original camera position and target in the 3D viewport and resets the 2D chart zoom.

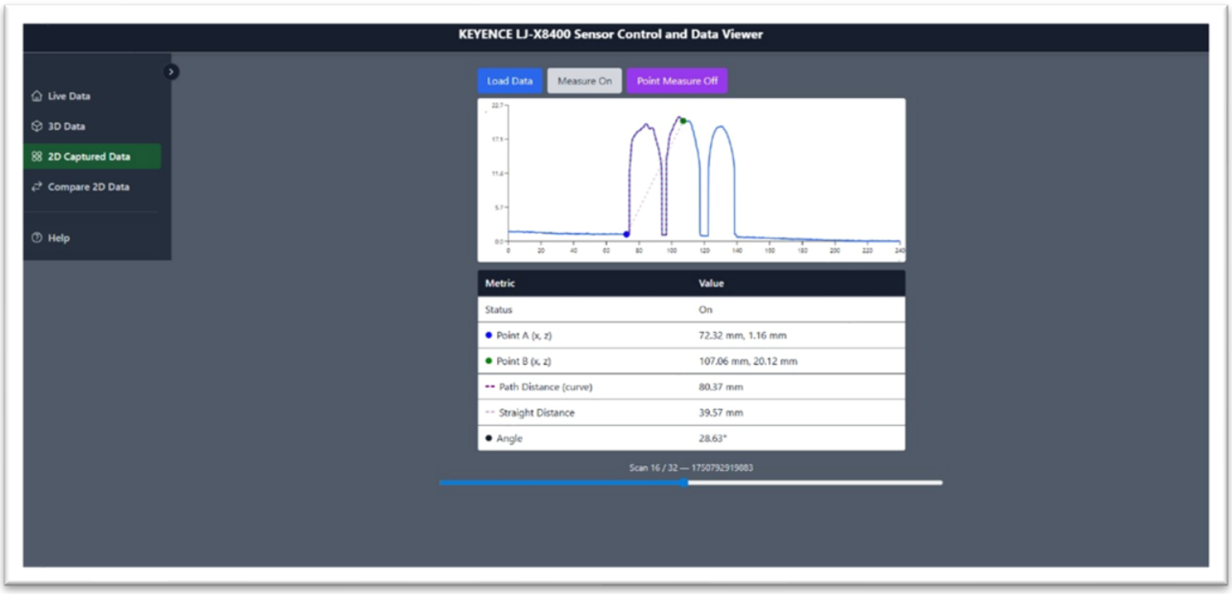
Clear All Data removes the loaded surface and clears the chart and metrics, returning the page to its initial state.

2D Captured Data Page

2D Captured Data Page Workflow

Page Layout and Initial State

The page contains a large 2D profile chart at the top, a metric table below it, and a horizontal scan slider at the bottom. The top control bar includes Load Data, Measure On/Off, and Point Measure On/Off buttons. In the initial state, no measurements are active, and the metric table displays only the status row.

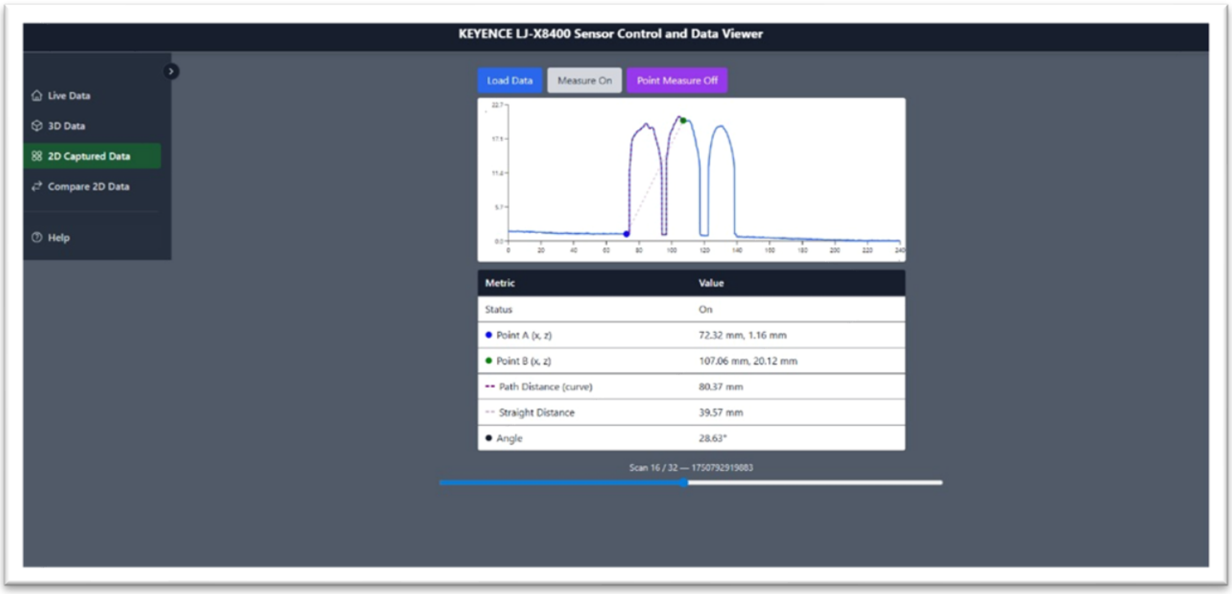


Metric	Value
Status	On
● Point A (x, z)	72.32 mm, 1.16 mm
● Point B (x, z)	107.06 mm, 20.12 mm
-- Path Distance (curve)	80.37 mm
-- Straight Distance	39.57 mm
● Angle	28.63°

Fig 15: 2D Captured Data Page, Profile loaded with measurement tools disabled

Load and View a Captured Profile

Click Load Data to import a saved scan profile from CSV format. The loaded profile appears on the 2D chart, with the horizontal axis representing X position (mm) and the vertical axis representing height Z (mm). Use the scan slider to switch between different captured scans in the dataset.



Metric	Value
Status	On
● Point A (x, z)	72.32 mm, 1.16 mm
● Point B (x, z)	107.06 mm, 20.12 mm
-- Path Distance (curve)	80.37 mm
-- Straight Distance	39.57 mm
● Angle	28.63°

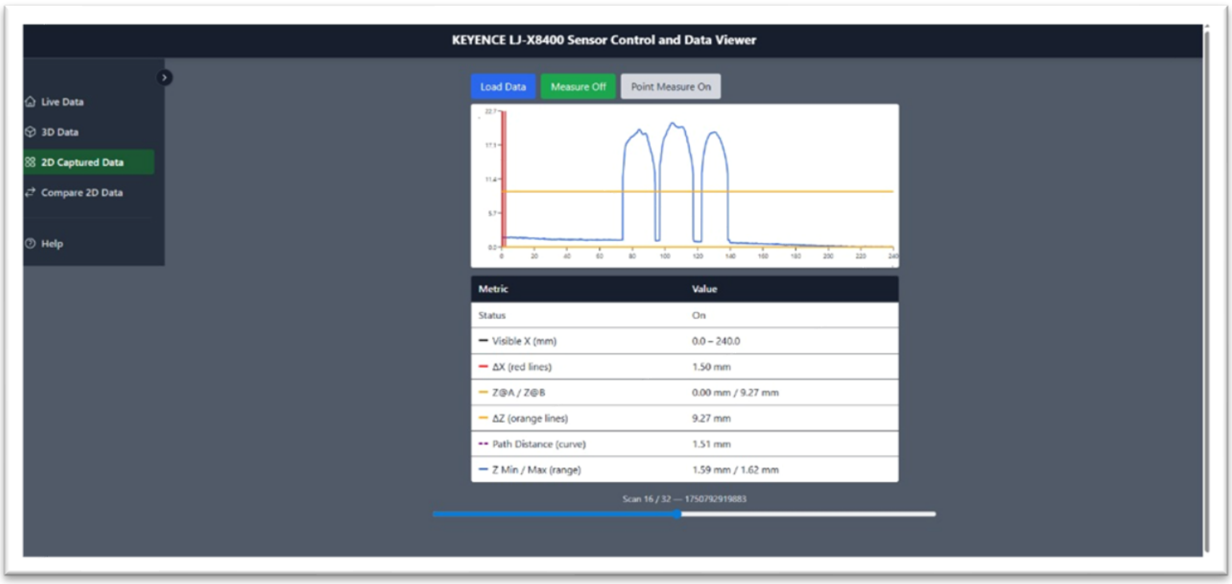
Scan 16 / 32 — 1750792919883

Fig 16: 2D Captured Data Page, Profile loaded with measurement tools disabled

Measurement Mode

Activate Measure On to enable ruler-based measurements on the chart. The metric table displays:

- Visible X (mm) – The horizontal range currently visible in the chart.
- ΔX (red lines) – Horizontal distance between two vertical measurement rulers.
- Z@A / Z@B – Height values at each vertical ruler.
- ΔZ (orange lines) – Vertical difference between two horizontal rulers.
- Path Distance (curve) – Cumulative distance along the profile between the two vertical rulers.
- Z Min / Max (range) – Minimum and maximum Z values within the visible range.



Metric	Value
Status	On
— Visible X (mm)	0.0 – 240.0
— ΔX (red lines)	1.50 mm
— Z@A / Z@B	0.00 mm / 9.27 mm
— ΔZ (orange lines)	9.27 mm
-- Path Distance (curve)	1.51 mm
— Z Min / Max (range)	1.59 mm / 1.62 mm

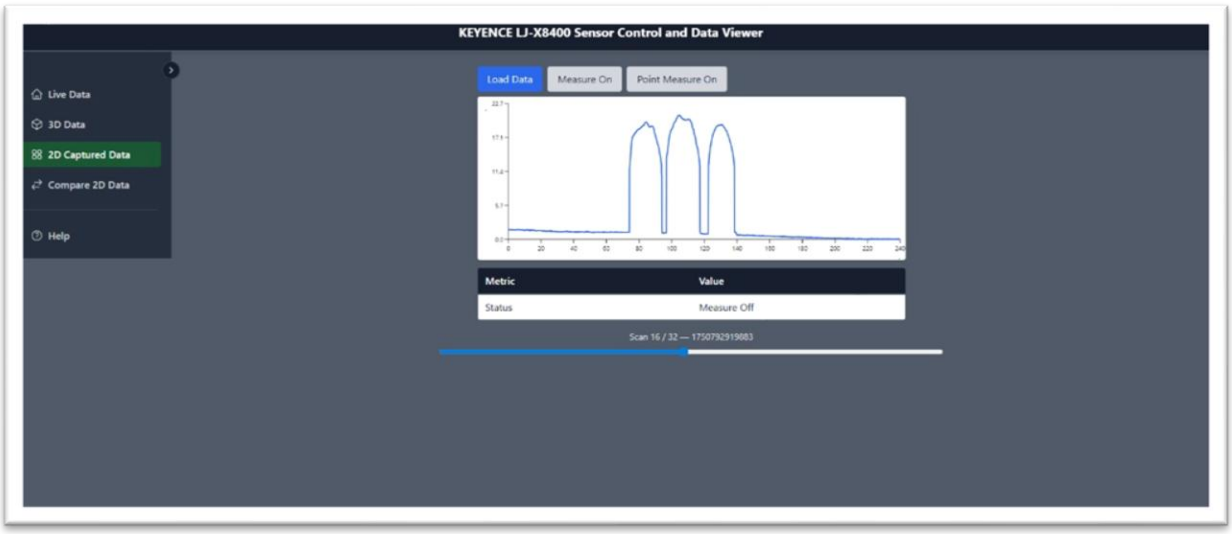
Scan 16 / 32 — 1750792919883

Fig 17: 2D Captured Data Page, Measurement mode enabled

Point-to-Point Measurement Mode

Activate Point Measure On to select two specific points (A and B) on the profile. The metric table then shows:

- Point A (x, z) – X and Z coordinates of the first point.
- Point B (x, z) – X and Z coordinates of the second point.
- Path Distance (curve) – Distance along the profile between A and B.
- Straight Distance – Euclidean distance between A and B.
- Angle – Inclination from the X-axis, in degrees.



Metric	Value
Status	Measure Off

Scan 16 / 32 — 1750792919883

Fig 18: 2D Captured Data Page, Point-to-point measurement mode enabled

Navigation and Control

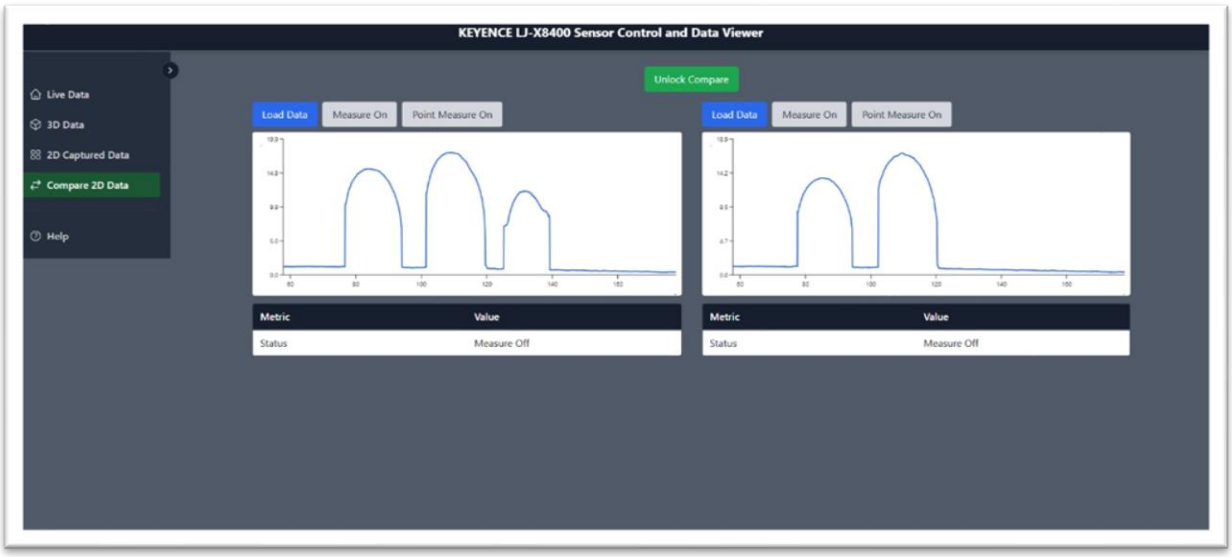
The scan slider at the bottom allows quick switching between multiple captured scans for review and comparison. Measurements remain active while switching scans unless explicitly turned off.

Compare 2D Data Page

Compare 2D Data Page Workflow

Page Layout and Initial State

The Compare 2D Data Page displays two independent 2D profile charts side-by-side for simultaneous analysis. Each chart has its own control bar (Load Data, Measure On/Off, Point Measure On/Off) and corresponding metric table. At the top centre, an Unlock Compare / Lock Compare button allows synchronized navigation between both charts. In the initial state, no measurements are active, and the metric tables show only scan status.



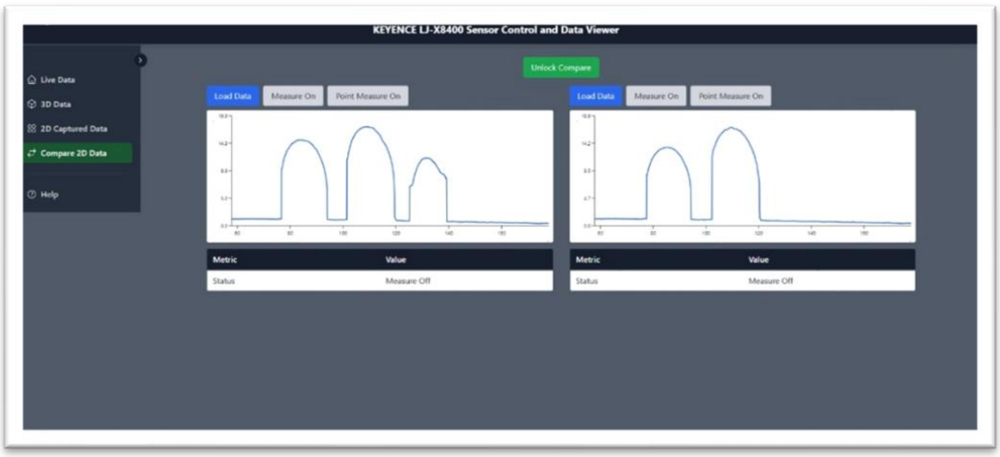
Metric	Value
Status	Measure Off

Metric	Value
Status	Measure Off

Fig 19: Compare 2D Data Page, Profiles loaded, measurement tools disabled

Loading and Viewing Profiles

Click Load Data on each chart to import separate 2D scan profiles from CSV files. The loaded profiles are displayed with the horizontal axis representing X position (mm) and the vertical axis representing Z height (mm). With Unlock Compare active, each chart’s scan slider operates independently; locking comparison synchronizes slider movement between both charts.



Metric	Value
Status	Measure Off

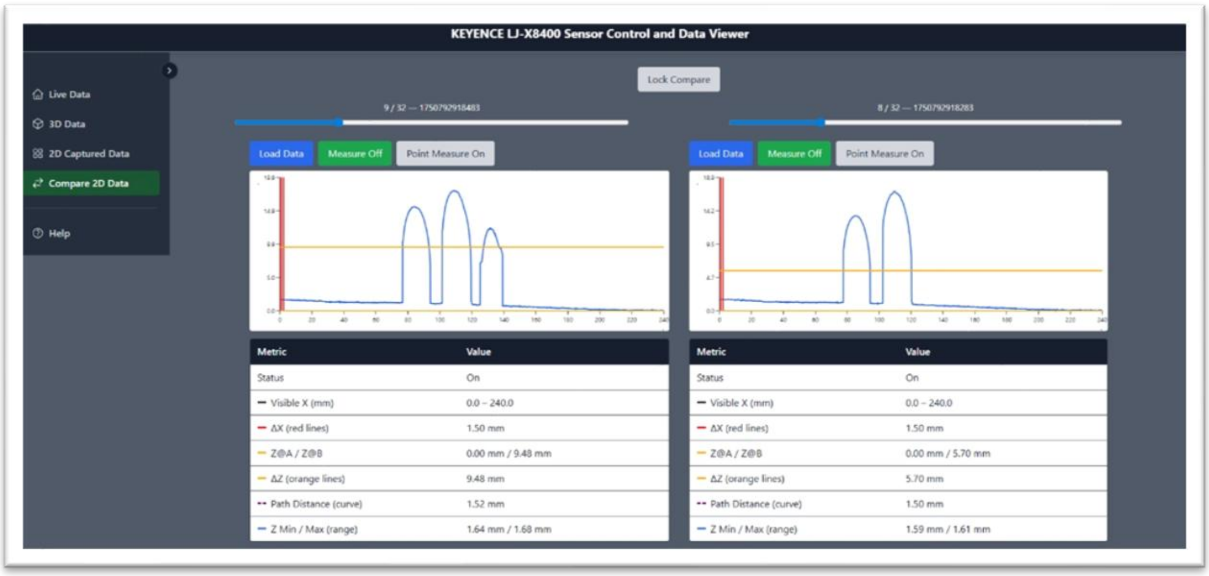
Metric	Value
Status	Measure Off

Fig 20: Compare 2D Data Page, Profiles loaded, measurement tools disabled

Measurement Mode

Activating Measure On in either chart enables ruler-based measurements on that profile. The metric table displays:

- Visible X (mm) – Current horizontal range visible.
- ΔX (red lines) – Horizontal distance between vertical measurement rulers.
- Z@A / Z@B – Height values at each vertical ruler.
- ΔZ (orange lines) – Vertical difference between horizontal rulers.
- Path Distance (curve) – Cumulative profile distance between the rulers.
- Z Min / Max (range) – Minimum and maximum Z values within the visible range.



Metric	Value
Status	On
— Visible X (mm)	0.0 – 240.0
— ΔX (red lines)	1.50 mm
— Z@A / Z@B	0.00 mm / 9.48 mm
— ΔZ (orange lines)	9.48 mm
-- Path Distance (curve)	1.52 mm
— Z Min / Max (range)	1.64 mm / 1.68 mm

Metric	Value
Status	On
— Visible X (mm)	0.0 – 240.0
— ΔX (red lines)	1.50 mm
— Z@A / Z@B	0.00 mm / 5.70 mm
— ΔZ (orange lines)	5.70 mm
-- Path Distance (curve)	1.50 mm
— Z Min / Max (range)	1.59 mm / 1.61 mm

Fig 21: Compare 2D Data Page, Measurement mode enabled on both charts

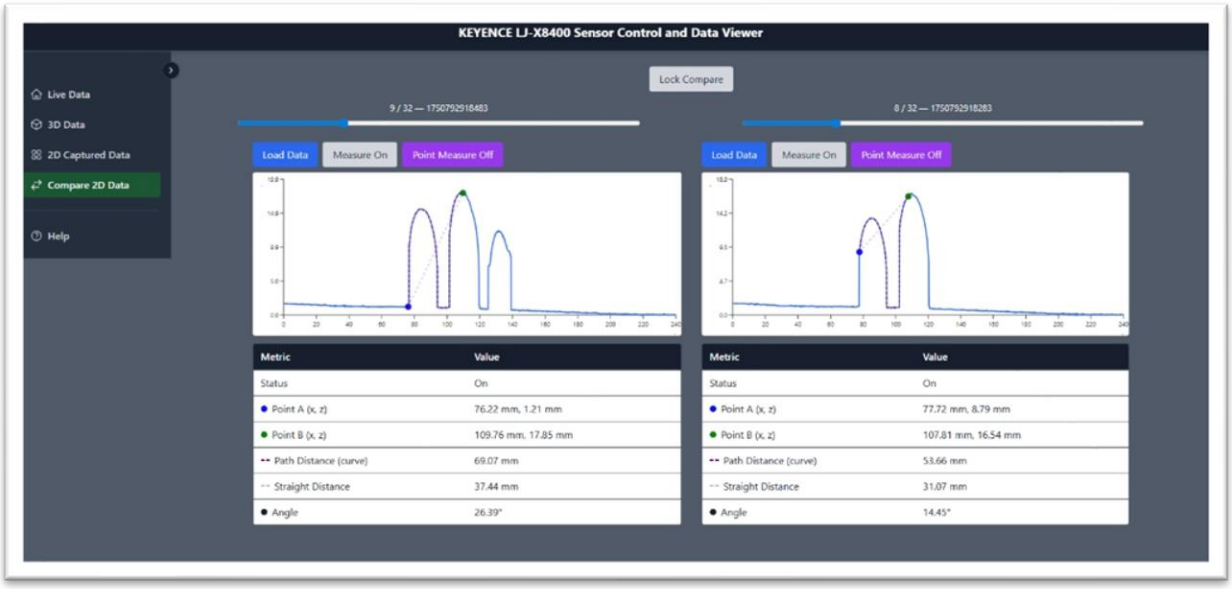
Point to Point Measurement Mode

Activating Point Measure On allows selection of two arbitrary points (A and B) on a profile.

The metric table updates with:

- Point A (x, z) – Coordinates of the first point.
- Point B (x, z) – Coordinates of the second point.
- Path Distance (curve) – Distance along the profile between A and B.
- Straight Distance – Euclidean distance between the two points.
- Angle – Inclination from the X-axis in degrees.

Both charts can operate measurement modes independently, allowing side by side comparison of different metrics for two datasets.



Metric	Value
Status	On
● Point A (x, z)	76.22 mm, 1.21 mm
● Point B (x, z)	109.76 mm, 17.85 mm
-- Path Distance (curve)	69.07 mm
-- Straight Distance	37.44 mm
● Angle	26.39°

Metric	Value
Status	On
● Point A (x, z)	77.72 mm, 8.79 mm
● Point B (x, z)	107.81 mm, 16.54 mm
-- Path Distance (curve)	53.66 mm
-- Straight Distance	31.07 mm
● Angle	14.45°

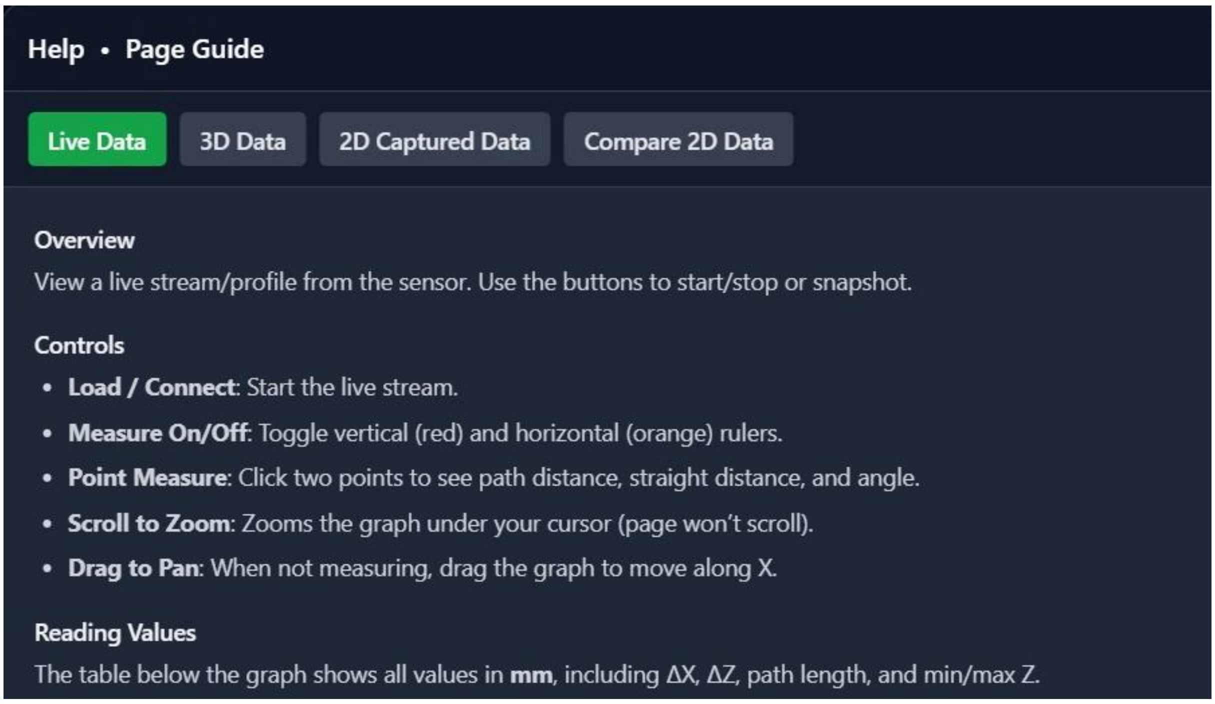
Fig 22: Compare 2D Data Page, Point-to-point measurement mode enabled on both charts

Synchronized Review

When Lock Compare is enabled, both scan sliders move in unison, allowing synchronized comparison of the same scan index across two different datasets. This mode is useful for comparing identical object scans captured under different conditions.

Help Page

Provide short, in-app instructions with screenshots to reduce onboarding time.



Overview

View a live stream/profile from the sensor. Use the buttons to start/stop or snapshot.

Controls

- **Load / Connect**: Start the live stream.
- **Measure On/Off**: Toggle vertical (red) and horizontal (orange) rulers.
- **Point Measure**: Click two points to see path distance, straight distance, and angle.
- **Scroll to Zoom**: Zooms the graph under your cursor (page won't scroll).
- **Drag to Pan**: When not measuring, drag the graph to move along X.

Reading Values

The table below the graph shows all values in **mm**, including ΔX, ΔZ, path length, and min/max Z.

Fig 23: Help Page

Deployment and Validation

Prerequisites: Node.js 18+, Python 3.10+ or newer, vendor library for real sensor access, network reachability to the controller.

Backend commands: create a virtual environment, install fastapi and uvicorn (or requirements.txt), start uvicorn app:app on port 8000.

Frontend commands: install dependencies, run npm start, open http://localhost:3000.

Smoke test: call /status, set IP/port, connect, and request /profile. The UI will then display profiles and allow capture and export.

Functionality Implemented

The system delivers a comprehensive set of features for real-time laser profile acquisition, visualization, and analysis:

- Real-time data streaming from the sensor via WebSockets.
- Live profile plotting with interactive measurement tools.
- Manual and automated scan capture with user-defined interval control.
- Export of scanned profiles to CSV format for offline analysis.
- Measurement capabilities including point-to-point (distance and angle), vertical, and horizontal rulers.
- Side-by-side profile comparison with synchronized zoom and interaction.
- 3D surface viewer with mesh slicing, inspection, and measurement functionality.

- Configurable IP and port settings through the UI, with LED indicators for connection status and data arrival.

Sensor Communication

Sensor connectivity is managed through a Python-based backend that interfaces with the Keyence DLL (LJX8_IF.dll) using the ctypes library. Communication is established over Ethernet, with IP address and port parameters configurable from the frontend interface. The backend exposes REST API and WebSocket endpoints to control the connection, retrieve live profile data, and manage capture processes.

Data Visualization

2D Profiles: Rendered as live-updating SVG graphs within React, supporting drag-and-click measurement interactions.

3D Surfaces: Visualized using Three.js (via react-three-fiber) with orbit controls, colour-mapped mesh representation, and slice inspection tools for cross-sectional analysis.

Measurements and 3D Stitching

The system offers precise measurement tools for both 2D and 3D modes:

Vertical/Horizontal Rulers: For height and width measurements on profiles.

Point-to-Point: Calculates both straight-line distance and inclination angle.

Slice-Based Measurements: Available in 3D mode for cross-sectional analysis.

Captured 2D profiles can be stacked into a surface grid, enabling the creation of stitched 3D meshes for full object reconstruction and inspection.

Robotic Integration

A robotic mounting solution was designed for integrating the sensor with a KUKA iiwa robot. The mount was created in Autodesk Fusion 360, optimized for stability and alignment, and 3D printed using a Bambulab X1 Carbon printer.

- The robotic scanning workflow will be implemented in the next development phase.
- STL files for the mount are available for replication or modification.

GitLab Issues Completed

All project requirements defined in GitLab were successfully completed:

- Development of a real-time UI for sensor control and data visualization.
- Implementation of capture and export functionality.
- Creation of measurement and 3D stitching tools.
- Design and fabrication of a robotic sensor mount.

- Full documentation of progress and adherence to Git best practices.

Conclusion

This project has successfully developed and deployed a fully functional, custom user interface for the Keyence LJ-X8400 laser profiler. The system supports high-resolution data acquisition, visualization, and measurement in both 2D and 3D modes, offering precise and intuitive tools for profile analysis.

Built with a modular architecture and implemented using clean coding practices, the platform ensures scalability, maintainability, and ease of integration with future enhancements. Comprehensive documentation accompanies the solution, enabling seamless adoption and further development for production use or advanced research applications.

Future Work

Planned enhancements include:

- Integration with ROS for automated robotic scanning workflows.
- Development of surface flattening and alignment algorithms.
- Implementation of profile-based defect detection using machine learning techniques.
- Real-time optimization of robotic scanning paths based on profile data.

Extensibility To Other Sensors

Pattern: a thin driver for the vendor SDK or protocol, a normalized data model (profiles, frames, point clouds, or time series), a stable REST API, and the same reusable React UI.

Driver options: SDK libraries, TCP/UDP, Modbus/TCP, EtherNet/IP, OPC UA, MQTT, RS-232/485, and vision interfaces such as GigE Vision/GenICam. Once wrapped, new sensors drop into the same pages with minimal UI changes.

Measurement reuse: rulers, two-point distances, angles, min/max, and slice-based analysis apply across many devices. The app's compare and CSV/API outputs remain the same.

References

- Keyence LJ-X8000 Series Documentation

https://www.keyence.com/products/sensor/laser-2d/lj-x8000/

- FastAPI – Modern, Fast Web Framework for Python

https://fastapi.tiangolo.com/

- React – JavaScript Library for Building User Interfaces

https://reactjs.org/

- ctypes – A Foreign Function Interface for Python https://docs.python.org/3/library/ctypes.html