DOCUMENTATION

Unified Print Pipeline

Contents

1. PROBLEM DEFINITION AND REQUIREMENTS

1.1 Problem Statement

To design and implement a unified print pipeline that enables a CAD software plugin to directly transfer 3D print geometries to a custom laser scancard. This bypasses traditional 'slice and save first, then print' workflows and establishes a seamless, single-click print experience from CAD to hardware.

1.2 Functional Requirements

ID	Requirement	Acceptance Criteria
1	The system must allow users to directly initiate printing from Cura via a menu action	"Send to Scancard" button triggers print pipeline
2	The plugin must extract G-Code per layer and recover XY toolpath coordinates	Layer-wise point list generated for every extruding segment
3	The plugin must generate continuous vector paths (no missing segments)	Interpolation must be applied, and processing pipeline must be verified to produce correct continuous geometry
4	The system must transform toolpaths into galvo-space coordinates within valid scan field limits	All coordinates must be mapped to [0, 65535] range before sending
5	The plugin must encode galvo scan vectors into a format compliant with scancard firmware	Log and verify packet formation before sending to scancard
6	The plugin must update Cura's layer preview UI to reflect the current print progress	Cura preview slider jumps to the last printed layer
7	Errors must be reported to user	Errors should be reported as pop-up messages
8	The user must be able to safely abort the print at any layer	Cancel returns system to safe idle state without sending further packets

1.3 Non-Functional Requirements

Category	Requirement	Rationale
Reliability	No blocked UI thread during network communication	Cura must remain responsive at all times
Scalability	Plugin must support 100+ layers per print without hanging	CAD prints typically contain 100+ layers
Maintainability	Processing pipeline must be modular	Faster debugging and component updates
Portability	Plugin runs on default Cura Python environment	Easy installation and testing
Usability	Zero-configuration workflow - one click from UI to print	Beginner-friendly for lab operators

1.4 Success Criteria

- Single-click print workflow validated on hardware
- Cura preview correctly synced with physical print progress
- 100% of extruding G-Code movements are correctly extracted and reconstructed into continuous vector paths
- Print runs to completion without system crashes or disconnects
- Any failure triggers a warning and safe abort without hardware fault

1.5 Constraints

- Scancard firmware expects a specific packet structure unique to itself
- Scancard expects packets only via UDP
- Galvo field limits dictate the valid digital coordinate range to [0,65535]
- Scancard firmware expects individual coordinates, not vector paths
- Plugin must run in Cura’s Python environment without rebuild

2. HIGH-LEVEL ARCHITECTURE

2.1 System Overview

The Unified Print Pipeline integrates directly into Cura and replaces the traditional multi-step export workflow. The system extracts sliced geometry from Cura, transforms it into galvo-space coordinates, encodes these coordinates into a scancard-compatible format, and streams commands live to the printer hardware — all in a non-blocking single-click user flow. The Unified Print Pipeline is built as a modular, event-driven Cura plugin that converts sliced geometry into scancard-ready galvo scan coordinates. Each subsystem is responsible for a specific stage in the print process — from geometry extraction to packet streaming.

2.2 Architecture Diagram

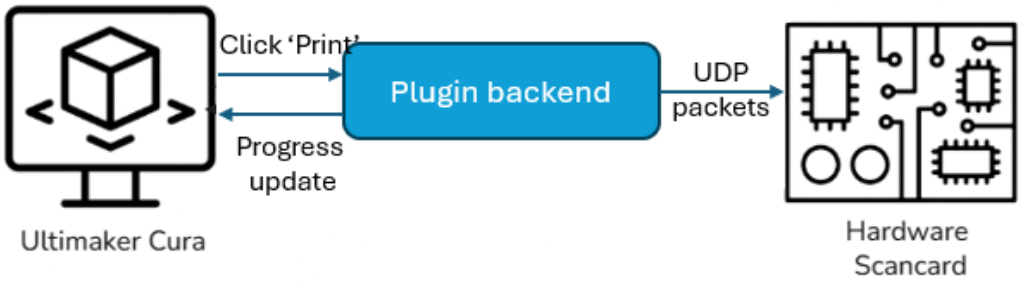


Fig 1: High-level overview of system architecture

Figure 1 represents a high-level overview of the system. The solution is directly integrated into Cura such that a single click 'Print' triggers the print pipeline. The click signals the plugin backend to start the print pipeline, which ultimately ends with the backend sending geometry data via UDP to the scancard. After successfully sending the layer data to the scancard, the backend updates the layer preview in Cura frontend to reflect the current layer being printed.

2.3 Major Components

Figure 2 represents the major components in the print pipeline.

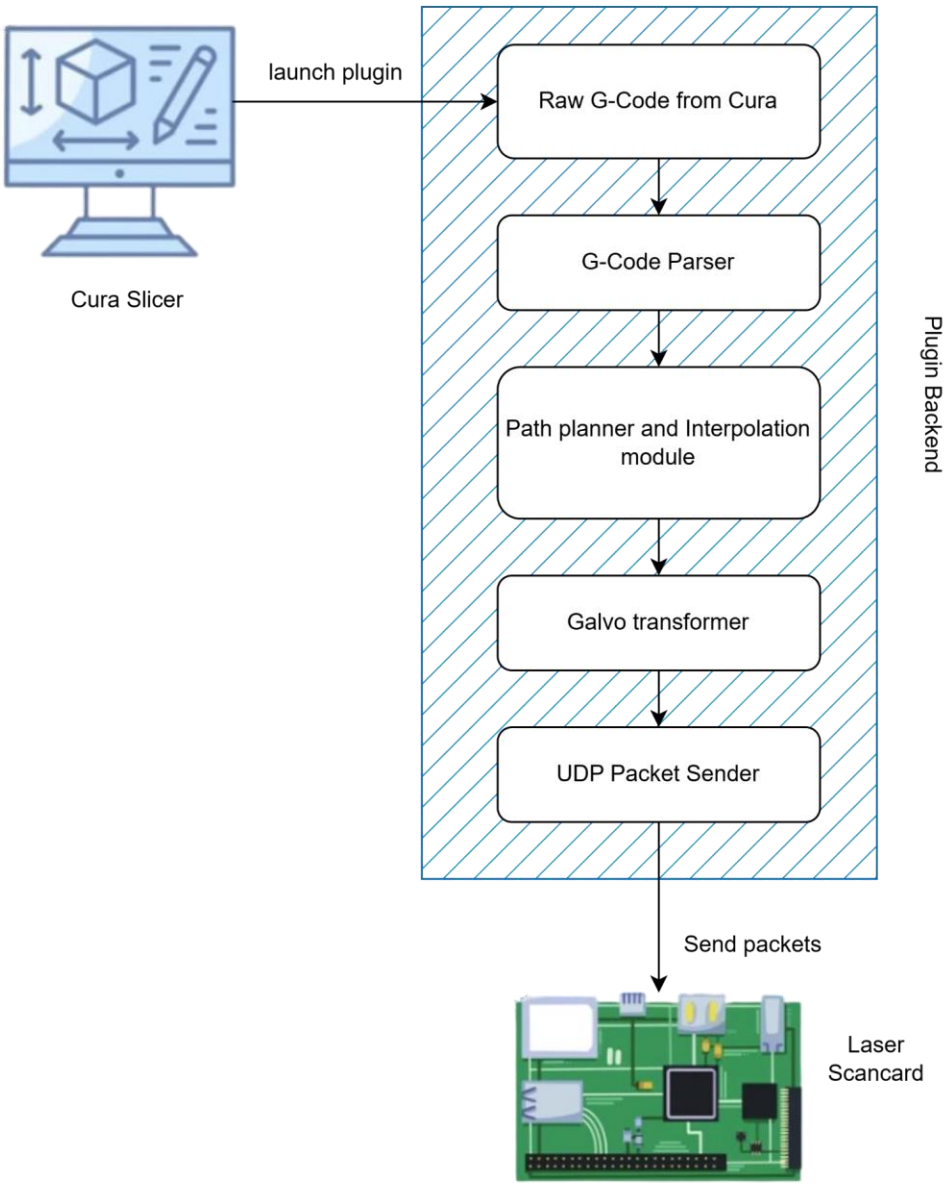


Fig 2: Detailed overview of system architecture

2.3.1 Cura (Slicer & UI)

- Cura software takes care of slicing with its built-in internal slicer
- This provides G-Code of the desired object
- Cura acts as a UI for both printing and monitoring

2.3.2 G-Code Parser

- This module reads the G-Code generated by Cura
- Detects extrusion segments via G1 commands and extrusion E increase
- Produces raw segment pairs for each layer as:

[(x1,y1) , (x2,y2) , (x3, y3) , ……]

where

x1, y1 → coordinates of starting point of segment 1
x2, y2 → coordinates of ending point of segment 1
x3, y3 → coordinates of starting point of segment 2

2.3.3 Path planner and Interpolation module

- Each segment is divided into smaller steps
- Ensures smooth and continuous galvo movement

2.3.4 Galvo transformer

- Takes the coordinates and converts them into galvo scale
- Maps values to galvo digital range [0, 65535]

2.3.5 UDP Packet Sender

- Converts galvo coordinates into scancard accepted packet structure
- Streams the packets to scancard

3. COMPONENT DESIGN

3.1 Structural Overview

Cura is built on top of Uranium, a modular application framework developed by Ultimaker. Uranium provides all the internal systems behind Cura's UI, slicing pipeline, settings engine, event loop, MVC architecture, and plugin ecosystem.

The Unified Print Pipeline is implemented entirely within this framework, as a Uranium Extension Plugin, which gives it complete access to Cura's slicing results, UI controls, scene data and application lifecycle.

3.1.1 Uranium and its components

Uranium consists of the following structural components -

1. **Application Core**

The Application is the top-level object in Uranium
It :

- Initializes on startup
- Loads plugins
- Manages global settings and preferences
- owns and instantiates the Controller
- Runs the Qt event loop

All plugins - including the Unified Print Pipeline - are registered and created through the Application.

2. **Controller**

The Controller manages everything the user sees and interacts with in Cura
It provides access to:

- the Scene (loaded meshes, G-Code, slicing metadata)
- the active view (Prepare, Preview, Monitor tabs)
- the SimulationView (layer preview slider)
- Rendering and view transitions

The plugin obtains a reference to the Controller through the Application and uses it to read sliced G-Code and update the preview slicer.

3. **PluginManager**

The PluginManager is part of the Application Core.
It is responsible for:

- Discovering and loading plugins

- Registering them into Cura's Extension system
- Adding menu entries
- Connecting plugin callbacks to the UI

The Unified Print Pipeline is loaded via this mechanism and appears under the 'Extensions' menu.

4. **View / UI layer**

This is the full UI rendering and interaction layer of Cura, powered by Uranium:

- 3D model view
- Preview mode
- SimulationView (layer slider)
- Dialogs, pop-ups, menu items

All UI updates, including the real-time layer slicer during printing, occur through this layer.

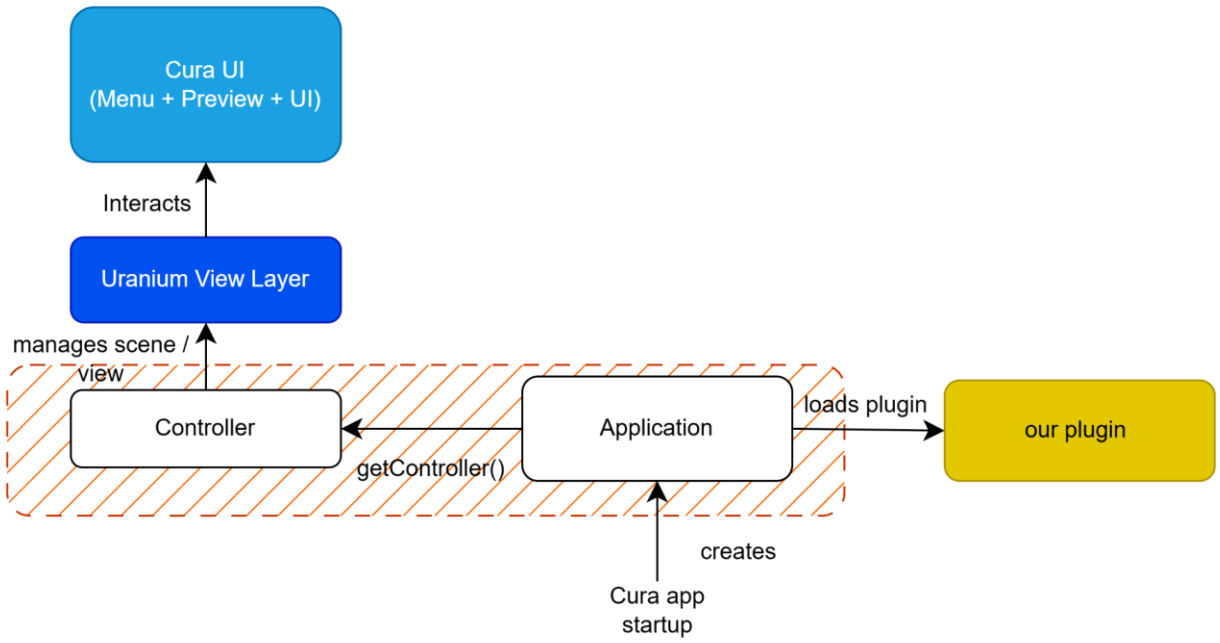


Fig 3: How our plugin fits in Cura's Uranium framework

3.1.2 How the Plugin fits into Uranium

The Unified Print Pipeline plugin sits inside the Application via the Extension API.
From there, it:

1. Gets the Controller to access sliced G-Code and scene data
2. Reads layer and path information from the Scene
3. Sends all geometry to backend modules for processing
4. Updates the SimulationView slider to reflect real hardware progress
5. Uses the View/UI layer for all dialogs and user interaction

Uranium uses a Model-View-Controller-like architecture, where the Scene is the Model, Cura's UI is the View, and the Controller sits between them. The plugin interacts with the Controller to read slicing metadata from the Scene and update the Preview View without breaking MVC separation.

3.2 Plugin Components

3.2.1 Components inside the Plugin

(a) G-Code Parser

(i) STEP 1: Extract G-Code from Geometry

This function retrieves the G-code generated by Cura after slicing directly from Cura's internal scene object. Once a print job is sliced, Cura stores the resulting G-code in the Scene Object under *scene.gcode_dict*, where each key corresponds to an extruder index (typically 0 for single-extruder setups). The plugin first checks whether slicing has been performed and whether valid G-code is available. The extracted G-code may be stored as a list of strings, so it is concatenated into a single text block for further processing. For debugging and verification, the G-code is written to a local text file and the initial 100 characters are logged in Cura's logs. The entire G-code text is then passed to a custom parser to extract layer-wise toolpath segments for downstream processing.

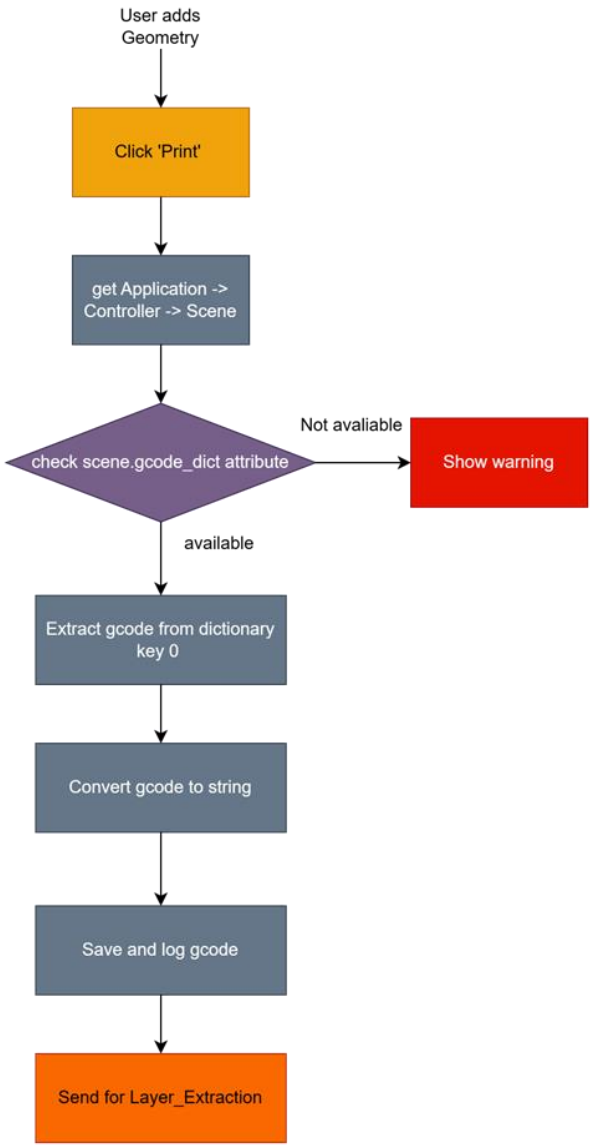


Fig - Code flow diagram showing G-Code extraction

(ii) STEP 2: Convert G-Code to coordinates

The G-code parser processes the sliced toolpath line by line to reconstruct extrusion segments on a per-layer basis. Layers are identified either explicitly through Cura's ***;LAYER:*** comments or implicitly through changes in the Z-height. The parser tracks the current tool position, extrusion value, and extrusion state to distinguish between travel moves (G0) and material-depositing moves (G1). Extrusion is detected when the extruder value (E) increases, while G1 commands without an E parameter inherit the previously active extrusion mode, enabling continuous extrusion tracking. For each extruding move,

the parser stores consecutive XY coordinate pairs, where each pair represents a single line segment of deposited material. The output is a dictionary mapping each layer to an ordered list of XY point pairs, allowing efficient reconstruction and visualization of layer-wise toolpaths.

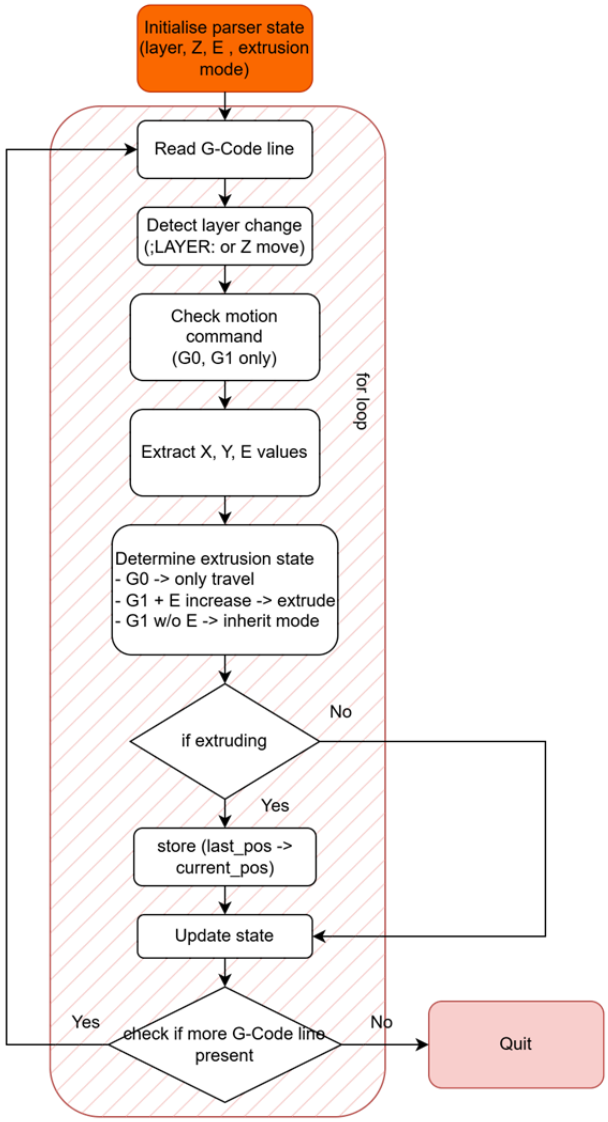


Fig - Flow diagram depicting the rest of the G-Code parsing

(b) Path planner and Interpolation module

After extracting layer-wise extrusion segments from the G-code, the global spatial bounds of the entire build are computed from the extracted coordinates. The user is

prompted to select an output directory, after which the layer-wise geometry is stored as serialized text files. Prior to export, each linear extrusion segment is resampled at a fixed spatial resolution using linear interpolation. This converts long toolpath segments into uniformly spaced points, resulting in a dense, resolution-controlled representation of the toolpath suitable for the scancard.

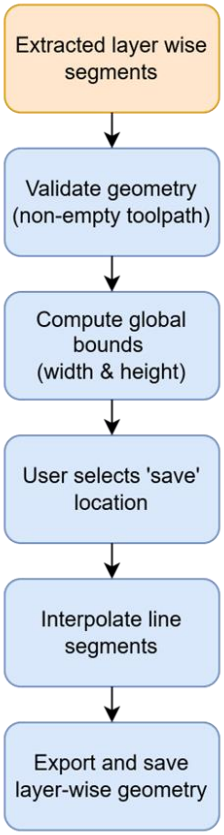


Fig - Flow diagram of Path planning and Interpolation module logic

(c) Galvo transformer

The coordinate transformation maps reminder-space toolpath coordinates into the digital input range of the galvo-based scan card. First, the minimum X and Y values across all layers are subtracted to shift the origin to the lower-left corner of the build. The geometry is then recentered by subtracting half of the global canvas width and height, ensuring symmetric placement around the origin. These centered coordinates are normalized to the range [−1,1][-1, 1][−1,1] based on the canvas dimensions and subsequently mapped to the full 16-bit galvo range [0,65536][0, 65536][0,65536]. An optional scaling factor is applied to reduce the effective output range, preventing galvo saturation and allowing finer control over the scan amplitude. The final

integer-valued coordinates are formatted into a representation compatible with the scan card firmware.

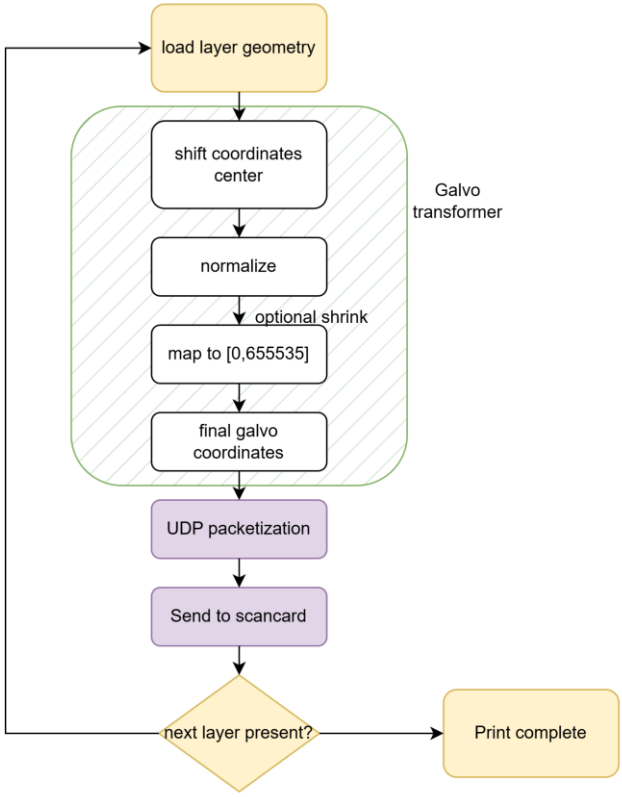
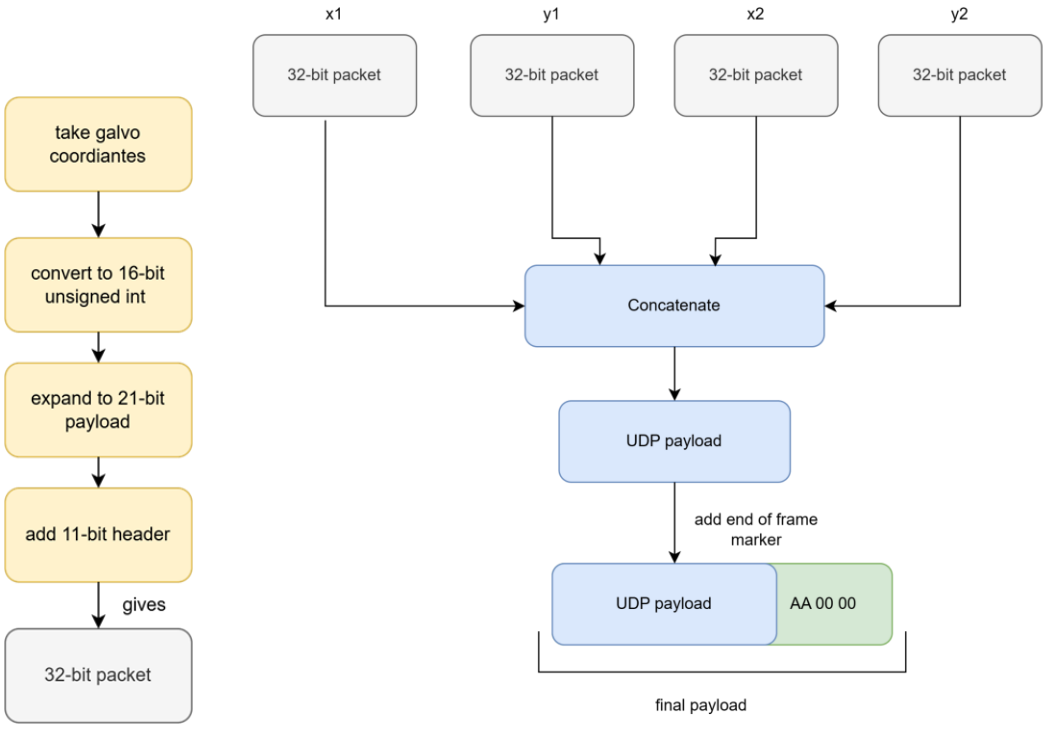


Fig - Flow diagram of Galvo transformer module

(d) UDP Packet Sender

The scan card interface expects toolpath data in a tightly packed binary format rather than raw coordinate values. To satisfy this requirement, each coordinate value is converted into a 16-bit unsigned integer, expanded to a 21-bit payload field to match the scan card protocol, and combined with an 11-bit command header to form a 32-bit packet. For each scan point, four such packets—corresponding to the X and Y coordinates of both left and right laser channels—are concatenated into a single UDP payload with an end-of-frame marker. The resulting payload stream is then transmitted sequentially to the scan card, enabling synchronized dual-laser scanning with deterministic timing and ordering.



(a) (b)

Fig (a) and (b) represents the UDP packetization workflow specific for the custom laser scancard

(e) API contract

- Cura-generated G-code is parsed into layer-wise extrusion segments
- Toolpaths are normalized globally and mapped to 16-bit galvo coordinates
- These 16-bit ints are expanded to 21-bit packets following a fixed rule
- Each field is encoded as a 32-bit packet (11-bit header + 21-bit payload)
- Four such 32 bit packets corresponding to [x1, y1, x2, y2] are concatenated and converted to raw bitstream
- x1,y1 correspond to left laser, and x2, y2 to the right laser
- Payloads are transmitted sequentially

4. DATA FLOW

- The user selects a 3D geometry in the CAD environment and initiates the print workflow.
- The geometry is exported and passed it to Cura's slicing pipeline.
- Cura generates G-code, which is accessed by the plugin through the scene object.
- The extracted G-code is parsed to identify layer-wise extrusion paths.
- Layer-wise geometry is normalized using global spatial bounds and mapped to the galvo coordinate space.
- Linear extrusion segments are interpolated at a fixed spatial resolution to generate dense vector paths.
- The vector paths are converted into hardware-compatible command packets in the format [x1, y1, x2, y2]
- Command packets are assembled into binary UDP payloads following the scan card protocol.
- The payloads are transmitted sequentially to the printer's scan card for layer-wise execution.
- After each layer is sent to the scancard, SimulationView is updated to reflect the completion of that layer in the print process

5. DESIGN RATIONALE

This design emphasizes a fast, operator-friendly, and hardware-safe workflow that eliminates intermediate failure points and simplifies the print process

5.1 Why is direct pipeline required?

- Traditional workflow involves:

$$\text{CAD} \rightarrow \text{Slice} \rightarrow \text{Export} \rightarrow \text{Transfer} \rightarrow \text{Print}$$

- Problems with this type of workflow are as follows
    - Manual steps introduce human error
    - Multiple software tools increase failure points
    - Time lost switching between systems
    - No real-time print monitoring feedback

The proposed unified print pipeline involves just one step:

$$\text{CAD} \rightarrow (\text{One click}) \rightarrow \text{Print}$$

This gives us benefits such as -

- Seamless user experience for lab operators
- Ensures coordinate integrity from slice to hardware

- Enables real-time progress sync inside Cura
- Reduces operational complexity and onboarding time

5.2 Why Cura?

- Cura is a widely used open-source slicer actively managed by a large community
- It exposes a Python-based plugin system, making integration straightforward
- The slicing engine is already optimised; no need to build a custom slicer
- Cura's frontend and backend can be manipulated via custom plugin code

Because of the reasons listed above, Cura provides the ideal foundation for a professional workflow without reinventing core slicing logic.

5.3 Why Python?

- Cura internally runs Python - plugins can reuse its runtime environment
- Excellent libraries for geometry processing, regex parsing, and math
- Enables rapid prototyping and debugging during hardware bring-up

5.4 Why modular design?

Each processing step (parsing, toolpath formation, transformation, encoding, UDP send) is a separate module. This gives us the following benefits:

- Improves testability: individual components can be validated independently
- Simplifies debugging and logging
- Allows easy upgrades and scalability

5.5 Why was socket-based communication chosen?

Socket-based UDP communication was chosen for transferring data to the hardware because the scancard firmware natively supports UDP streaming and expects the data to come through UDP protocol only.

6. ERROR HANDLING

The system implements defensive error handling across every stage of the print pipeline to ensure operator safety, GUI responsiveness, and hardware protection.

6.1 Error during Slicing

Trigger conditions

- User clicks “Send to Scancard” before slicing
- Cura fails to generate G-Code
- No object has been placed in the build space

Response

- Shows GUI warning - “Please slice the model first.”
- Prevents the start of the print pipeline, does not send any commands to hardware

6.2 Error during geometry parsing

Trigger conditions

- G-Code missing XY coordinates
- Invalid or unsupported G-code format

Response

- Abort safely with a popup and log the warning
- No malformed data flows downstream

6.3 Error during path planning

Trigger conditions

- Zero-length segments
- Degenerate geometry (only travel moves)

Response

- Triggers a error popup and aborts the print cycle
- Logs the malformed path segments for further inspection

6.4 Error during printing

Trigger conditions

- User wants to abort process after a layer is done
- Error in any of the above steps during layer printing

Response

- A popup appears in case of error
- A popup asking the user whether to continue appears for each layer

6.5 Abort on malformed geometry or User Cancel

Trigger conditions

- Empty layer set after parsing
- User presses ‘No’ when asked whether to continue to next layer

Response

- Stop transmission loop entirely
- Log: “Print job cancelled by user”
- Show safe termination dialogs

7. HELPER SIMULATION AND VISUALISATION TOOLS

To ensure safe testing without physical hardware and to validate toolpaths before sending them to the scancard, two auxiliary applications were developed.

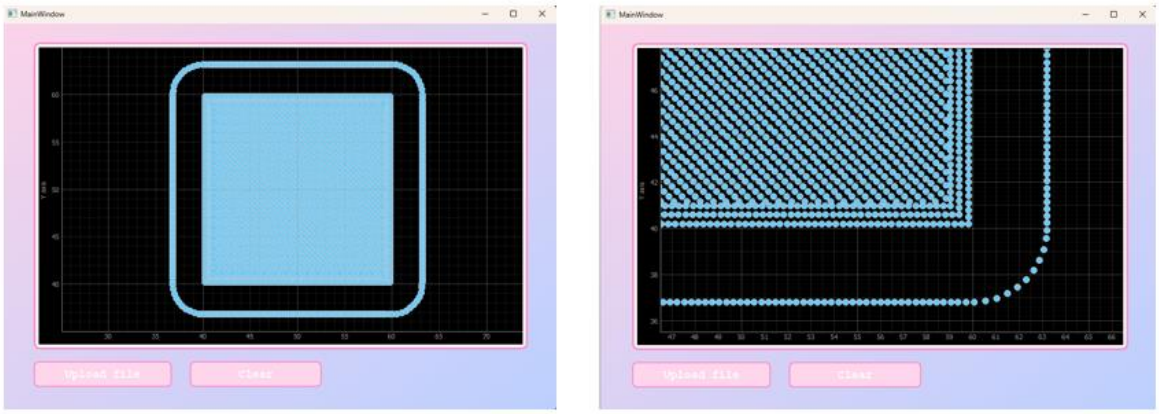
7.1 Layer Plotter Application (before galvo transformation)

A PyQt5-based desktop application designed to preview layer coordinates before galvo transformation.

This software takes layer files as input with the following format -

layer_x.txt

[(x1,y1), (x2,y2), (x3,y3), ……..]



(a) Simulation of coordinates before galvo transform

(b) Zoomed in view of Fig. a

It then maps the coordinates, which helps to verify whether the extraction and processing of the G-code from Cura were accurate. This helps us confirm the correctness of the data processing pipeline without having to turn on the actual laser hardware.

Key features

- Built using PyQt5 + pyqtgraph
- Uses ast.literal_eval for safe parsing of coordinate tuples, instead of eval() to avoid executing malicious text. This prevents execution of arbitrary Python code if a user uploads a malicious or malformed file.
- The parser validates that all loaded elements are numeric tuples before processing

- Shows an animation of the order in which coordinates are plotted, mimicking the actual print process
- Used for debugging layer splitting and hatch generation steps in the data processing pipeline

7.2 Galvo-Space Coordinate Visualizer

For a final sanity check of the processing pipeline, a standalone Python script with matplotlib was used to get a mapping of the coordinates after galvo transformation, just before they were converted into UDP packets to send to the scancard.

Purpose

- Ensure that the transformed coordinates remain within the galvo scan field limits of [0, 65535] digital units
- To make sure that the geometry is centred at the middle of the galvo field

Features:

- Parses transformed coordinate pairs.
- Draws line segments connecting point pairs.
- Shows galvo center crosshair for reference.
- Prints coordinate ranges and segment counts.
- Ensures vectors do not exceed scancard bounds.

Used for:

- Debugging galvo calibration
- Validating coordinate scaling math
- Previewing final scan path

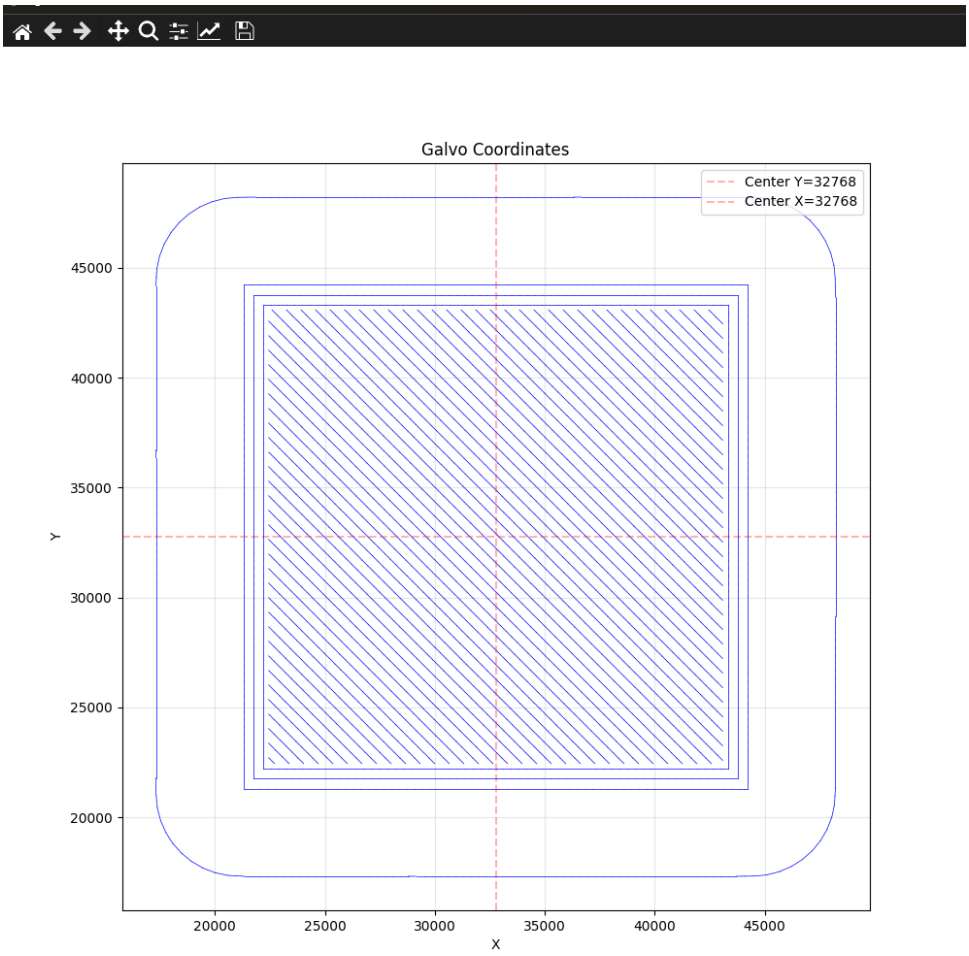


Fig. Galvo-space visualisation using Python script

8. LIMITATIONS

The current implementation of the Unified Print Pipeline prioritizes fast integration with Cura and reliable real-time communication with the scancard. As a result, the following limitations exist in the present design:

1. Single-threaded execution
    - The plugin executes within Cura's main Python/Qt event loop
    - Heavy operations such as geometry parsing, interpolation, and coordinate transformation are performed synchronously
    - UI responsiveness during printing is not preserved - the UI freezes when geometry parsing/printing of one layer takes place

2. Cura dependency
    - The pipeline relies on Cura's slicing engine and Uranium APIs
    - It cannot currently operate as a standalone application

3. UDP reliability tradeoffs
    - Communication with scancard uses UDP for low latency
    - UDP does not guarantee packet delivery or ordering at the protocol level
    - Hence the system relies on firmware-level ordering and timeout assumptions rather than transport-level guarantees

4. No closed-loop hardware feedback
    - The plugin assumes that the packets are received by the scancard once sent. It does not get an acknowledgement from scancard on receipt of packet. Hence there are no retry or resend options available if in case the packet is not received due to some error
    - The plugin assumes successful execution of scan commands once packets are sent
    - No real-time confirmation of galvo-position, printing or job completion is received from the hardware scancard side

5. Only layer-wise animation/preview available
    - SimulationView only shows layer-wise animation
    - It only shows which layer is getting printed and does not show which segment inside the layer is getting printed
    - Segment-wise animation per layer is not available

9. FUTURE IMPROVEMENTS

The current Unified Print Pipeline establishes a stable end-to-end workflow from Cura to the scancard. The following improvements are planned to enhance robustness, scalability, and usability in future iterations:

1. Pipeline state machine
    - Introduce an explicit state machine to track pipeline execution stages such as:
        - Geometry export
        - Slicing
        - Path generation
        - Packet transmission
        - Print completion/ failure
    - Each state transition will be validated and logged
    - Enables clearer progress reporting, deterministic error handling, and safe recovery from intermediate failures
    - This will simplify debugging and monitoring

2. Hardware-safe coordinate clamping
    - The current implementation performs mathematical normalization of coordinates into the galvo input range
    - Future versions will add explicit hard clamping of coordinates to the scancard's valid range
    - The goal is to clamp transformed galvo coordinates to [0,65535] before packet generation
    - This provides guaranteed hardware safety and protects galvo mirrors from out-of-range inputs caused by malformed geometry or numerical instability

3. Multi-threaded backend execution
    - Move geometry parsing, interpolation, and packet generation to background worker threads
    - Keep Cura's UI thread responsive during printing and layer processing
    - This will eliminate UI freezes during heavy computation and improves user experience for complex geometries