Design Document: Automated Intelligent Document Stamping System

Team 6 Senior Design Project

Yanzhen Chen [UID: 3220111908]

Zhiqiang Qiu [UID: 3220111914]

Xuliang Huang [UID: 3220111926]

Jiaheng Zeng [UID: 3220111929]

Advisor: Prof. Fangwei Shao

ZJU-UIUC Institute

Date: May 26, 2026

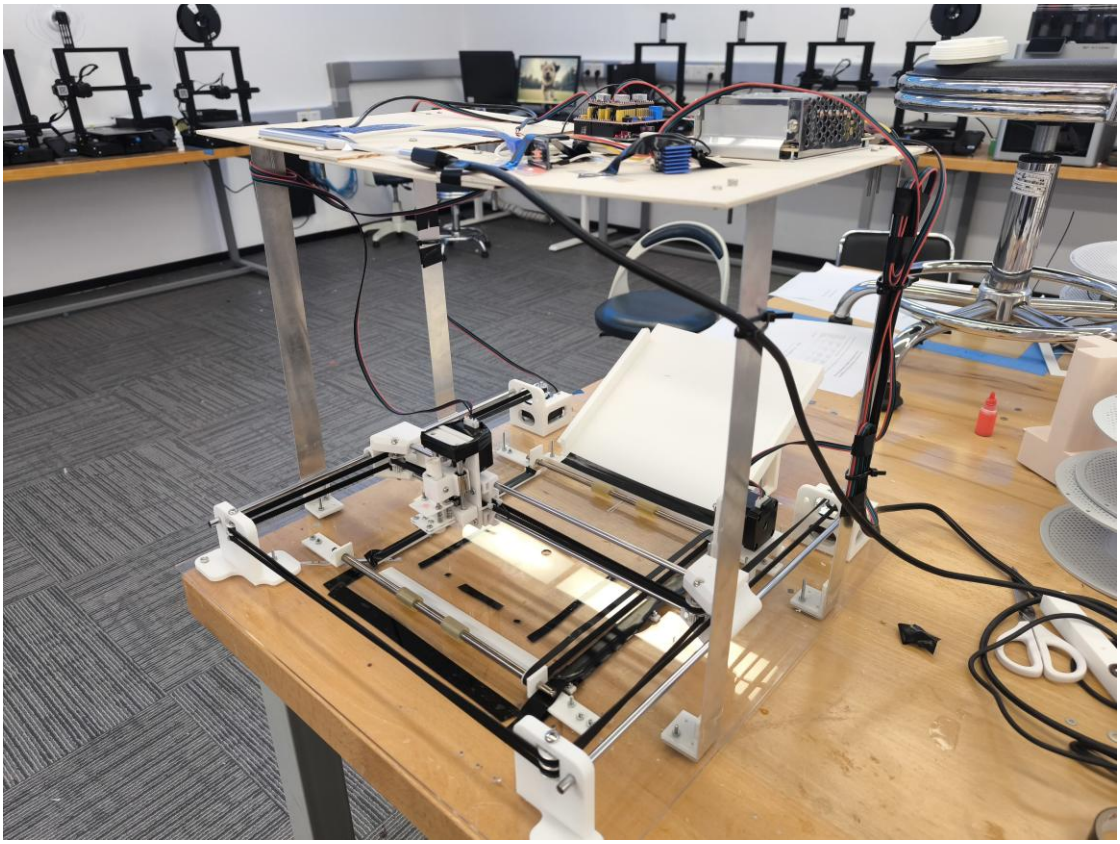

Figure 1. Prototype automatic stamping system used in the final demonstration.

Abstract

This project implements a desktop automatic stamping system for administrative document handling. The final prototype combines a modified belt-driven writing-robot frame, a spring-loaded Z-axis stamp head, an independent roller feed, a fixed overhead USB camera, and a Python-based control application. The software is implemented as a FastAPI backend with a web/pywebview operator interface. OpenCV is used for paper detection, feature matching, and homography-based coordinate mapping, while serial links send GRBL-compatible G-code to the X/Y/Z controller and roller commands to an Arduino Nano. The system supports document-recognition, camera-targeting, and manual-positioning workflows, allowing the operator to choose the stamping point from an uploaded document, a live camera image, or the current machine position.

Table of Contents

1. Introduction

1.1 Problem Statement

Administrative offices often process many forms that require a physical stamp in a consistent location. Manual stamping is repetitive, slow, and sensitive to human variation in placement, angle, and pressing force. A low-cost desktop machine that can accept common document inputs and place a stamp at a user-selected target would reduce operator workload while preserving the flexibility needed for different forms.

1.2 Solution Overview

The final system is a camera-guided automatic stamping machine. The operator uses the frontend to upload a PDF or image, select a target from the live camera view, or manually position the head. The backend converts the selected target into a paper-relative coordinate, maps it through the calibrated camera/stamp region, generates G-code, and sends motion commands over serial. A separate roller controller feeds the sheet before and after stamping. The software stack uses OpenCV for vision processing, FastAPI for the backend API, PySerial for serial communication, and GRBL-compatible motion control for the X/Y/Z stage [1]-[4].

1.3 High-Level Requirements

9. The vision subsystem shall localize the selected stamp target within the calibrated stamp region with a final stamping error of less than 5 mm after calibration.
10. The motion subsystem shall execute X/Y/Z stamping commands through GRBL-compatible serial control and complete the preview-to-stamp sequence without leaving the configured workspace bounds.
11. The user interface shall support three operating modes: document recognition, camera targeting, and manual positioning, with a usable dry-run/debug path before live motion.

2. Design

2.1 System Architecture

Layer	Component	Role
User interface	web/index.html, web/app.js, web/styles.css, pywebview shell	Target selection, calibration, serial setup, jogging, dry-run preview, and live stamping.
Backend API	src/stamping_system/api.py	FastAPI routes for configuration, document upload, camera frames, paper detection, target preview, motion, stamping, and roller control.
Vision and mapping	document_matching.py, vision.py, calibration.py, targeting.py	OpenCV feature matching, paper detection, homography, and conversion from document/camera pixels to machine coordinates.
Motion planning	pipeline.py, gcode.py, serial_link.py	G-code generation, workspace validation, serial connection management, jogging, moving, and stamping.
Paper feed	roller_link.py, config/machine.toml	Independent roller control through Arduino Nano commands and feed calibration values.
Hardware	GRBL controller, Arduino Uno/Nano, NEMA 17 motors, TMC2209/A4988 drivers, USB camera	Physical motion, stamping, sheet transport, and camera input.

Table 1. Implemented system architecture.

2.2 User Operating Workflow

Mode	Operator Input	Implemented Workflow
Mode A: document recognition	Upload an A4 PDF/image and click the stamp point on the preview.	Render preview, match document features against the current camera frame with OpenCV, map the relative point to the detected paper, validate the stamp region, and run the stamp sequence.
Mode B: camera targeting	Click the target directly on the live or simulated camera frame.	Convert the camera pixel to a paper-relative coordinate, resolve it through calibration, preview G-code, then execute move/stamp.
Mode C: manual positioning	Jog the machine until the stamp is physically above the target.	Use the confirmed current machine position as the target and execute the Z stamping cycle with optional feed commands.

Table 2. User operating modes.

2.3 Vision and Software Module

The implemented vision path uses OpenCV rather than a production VLM. Uploaded documents are rendered to an image preview. For document-recognition mode, AKAZE features are extracted from both the preview and camera frame; a Hamming-distance matcher, ratio test, and RANSAC homography map the selected relative document point to a live camera pixel. Paper and region checks use saved ROI polygons, color/edge features, and configured thresholds. This approach matches the actual source file document_matching.py and avoids claiming unsupported VLM behavior.

- Camera input supports a USB camera, uploaded simulation frame, or placeholder image for software testing without hardware.
- Calibration stores pixel-to-real correspondences and stamp-region polygons in config/machine.toml.
- The application returns debug images with document outlines, selected target dots, and match statistics for operator verification.

2.4 Motion Control Module

The X/Y/Z stage is controlled by a GRBL-based motion controller through a persistent serial connection. The backend generates metric motion commands for travel, jogging, and stamping, while applying workspace bounds and axis scale factors before live motion.

The Motion controls in the UI follow a CNCjs-style GRBL workflow, including serial connection, controller unlock, status query, manual jogging, continuous jogging, dry-run preview, and live execution [5]. This allows the system to reuse familiar CNC control concepts while customizing the workflow for document stamping instead of general CNC machining.

2.5 Paper Feed Module

The paper feed module uses a roller mechanism driven by a NEMA 17 stepper motor through a GT2 belt. In the implemented prototype, an Arduino Nano with a TMC2209 driver controls the roller independently from the GRBL X/Y/Z motion controller. Its controls are placed in the Motion section of the UI and follow the same CNCjs-style layout used for serial connection, jogging, and motion feedback [5]. However, the paper feed is treated as a separate subsystem rather than a GRBL fourth axis, so feed length, direction, and recovery behavior can be calibrated independently. The software supports basic feed calibration, direction control, and timeout handling to prevent continuous motion if feeding fails.

2.6 User/Developer Interface and Configuration

The front end is separated into a User Entry and a password-protected Developer Entry. The User entry only provides the three daily stamping modes: document recognition, camera targeting, and manual positioning, so operators can select a target and run the stamping workflow without changing machine settings. The Developer entry is used for calibration and configuration, including camera setup, stamp-region alignment, workspace bounds, serial connection, motion tuning, paper-feed calibration, and firmware notes. This separation keeps normal operation simple and reduces safety risks from accidental parameter changes.

3. Subsystem Requirements and Verification

Subsystem	Requirement	Verification
Vision	Document-recognition mode shall compute a target from an uploaded document and camera frame when at least 10 reliable feature matches and 8 RANSAC inliers are available.	Run 20 sample document/camera pairs; log match count, inliers, target pixel, and success/failure message from /api/document/match.
Vision	Camera/stamp-region mapping shall keep final stamp placement error below 5 mm after calibration.	Place 10 marked targets across the stamp region, run the dry preview and live stamp sequence, and measure center-to-target error with a ruler or caliper.
Motion	The backend shall reject live moves outside configured X/Y workspace bounds.	Set known workspace limits, attempt in-bound and out-of-bound jog/move commands, and verify only valid commands are sent over serial.
Motion	The Z cycle shall press and release without requiring a force sensor in the implemented prototype.	Run 20 stamping cycles on scrap paper and inspect whether the mark is present and whether the head returns to safe Z after each cycle.
Paper feed	The roller feed shall advance a sheet by the calibrated feed length and stop on timeout rather than running indefinitely.	Run 30 feed cycles with the current belt tension, record feed success, timeout count, and whether the paper reaches the configured ROI.

User interface	The interface shall provide dry-run/debug visibility before live stamping.	For each mode, preview the target and verify that the UI shows target coordinates, debug image/region status, and generated G-code before live execution.

Table 3. Subsystem requirements and verification.

4. Tolerance Analysis

The critical subsystem function is mapping a selected document or camera target to the physical stamp point. The final error can be modeled as the sum of camera matching error, calibration error, motion error, and stamp error:

$$E_{total} = E_{vision} + E_{calibration} + E_{motion} + E_{stamp}$$

Based on the final tests, the measured stamping error was less than 5 mm, while the vision error was less than 30 pixels, corresponding to about 0.20 configured machine units. Therefore, most remaining errors come from mechanical factors such as belt tension, camera-holder vibration, and stamp-center misalignment.

Error Source	Current Design Control	Residual Risk
Vision feature matching	AKAZE features, ratio test, RANSAC homography, debug overlay.	Low-texture or highly repetitive forms may produce too few inliers.
Calibration	Stored stamp-region polygon and machine corner mapping.	Camera mount vibration can invalidate saved calibration.
X/Y motion	GRBL motion, workspace bounds, axis scale factors.	Loose belts and frame flex can cause repeatability loss.
Z pressing	Spring-loaded stamp head and timed Z motion.	Uneven support can incline the stamp and create inconsistent force distribution.

Table 4. Tolerance contributors and residual risks.

5. Cost

Item	Quantity	Estimated Cost (RMB)
USB camera	1	180
Pillars, base, and frame parts	1 set	100
Arduino Uno, Arduino Nano, CNC expansion board, drivers, wiring, and USB hub	1 set	400
Stamp, belts, pulleys, printed gears, and miscellaneous hardware	1 set	50
Total		730

Table 5. Prototype cost estimate.

6. Schedule

Phase	Zhiqiang Qiu	Yanzhen Chen	Xuliang Huang	Jiaheng Zeng
Week 5	Paper path CAD	Vision/software setup	Gantry frame assembly	Component sourcing
Week 6	Roller prototype	OCR/OpenCV testing	Motor tuning	frame planning
Week 7	Paper feeding test	Calibration, mapping test	Z-axis stamper test	3D-printed part preparation
Week 8	Mechanism refinement	System logic integration, UI design and test	Placement Optimization	Baseboard mounting and integration
Week 9	Assembly	Debugging	Assembly	Assembly

Table 6. Project schedule and task ownership.

7. Ethics and Safety

7.1 Ethics

The system may process administrative documents containing private information. Following the IEEE Code of Ethics, document images should be used only for target calculation and debugging, and unnecessary storage should be minimized [6]. The current design keeps processing local to the operator machine and does not rely on cloud vision services during normal operation. When sensitive documents are used, runtime preview and snapshot files should be cleared after testing, and access to developer configuration should be limited to authorized users.

7.2 Safety

- **Mechanical safety:** Belts, pulleys, rollers, and the Z-axis stamp head are pinch hazards. Operators should keep hands outside the motion region during live motion and use dry-run preview before enabling hardware.
- **Electrical safety:** The prototype uses low-voltage controllers, motors, and USB devices. Wiring should be insulated, strain-relieved, and checked before operation.
- **Motion safety:** Workspace bounds, serial connection status, timeout handling, and dry-run mode reduce unexpected motion. A physical emergency stop is recommended for future revisions.
- **Access safety:** The User entry limits normal operators to the three stamping modes, while the password-protected Developer entry restricts calibration and motion parameters to trained users.
- **Reliability safety:** Known risks include camera-holder vibration, roller belt slipping, and uneven stamp pressing. Future revisions should stiffen the camera support, improve belt tensioning, and constrain the Z head with dual guide rods.

8. References

[1] OpenCV, "OpenCV documentation." [Online]. Available: https://docs.opencv.org/. Accessed: May 27, 2026.

[2] S. Ramírez, "FastAPI documentation." [Online]. Available: https://fastapi.tiangolo.com/. Accessed: May 27, 2026.

[3] C. Liechti, "pySerial documentation." [Online]. Available: https://pyserial.readthedocs.io/. Accessed: May 27, 2026.

[4] gnea, "Grbl: An open source, embedded, high performance g-code-parser and CNC milling controller," GitHub. [Online]. Available: https://github.com/gnea/grbl. Accessed: May 27, 2026.

[5] cncjs, "cncjs: A web-based interface for CNC milling controller running Grbl, Marlin, Smoothieware, or TinyG," GitHub. [Online]. Available: https://github.com/cncjs/cncjs. Accessed: May 27, 2026.

[6] IEEE, "IEEE Code of Ethics." [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. Accessed: May 27, 2026.