ME 424 – Ballistic Gel Fragmentation

Final Report

Report Due: Thursday, April 30

Alexandra Bergman, Pierre Gathy, Fernando Rodriguez, Giovanni Smith

I have adhered to the Duke Community Standard in completing this assignment. I understand that a violation of the Standard can result in failure of this assignment, failure of this course, and/or suspension from Duke University.

Executive Summary

The U.S. Army DEVCOM Armaments Center Test and Evaluation Division conducts live-fire ballistic testing using ballistic gelatin blocks to capture and preserve fragmentation. Following each test, these blocks must be dissected to determine the location, mass, and composition of each embedded fragment. The current workflow relies entirely on repeated manual slicing, measuring, and extraction which is slow, labor-intensive, and difficult to perform consistently across tests. The team was tasked with designing and building a practical automated system that reduces labor time, preserves fragmentation evidence, maintains repeatability, and remains simple to set up and operate.

The primary design criteria driving this project were: a reduction in human labor time per block of greater than 50%, the ability to make precise cuts at user-specified coordinates, and the capacity to determine the resting location of all fragments through an improved slicing and probing workflow. Secondary criteria included cost efficiency, with the goal of reducing long-term lab operating costs and improving overall throughput.

The team's solution is an automated slicing machine that accepts user-input slice coordinates and executes each cut autonomously. A ballistic gelatin block is loaded into a fixed tray and positioned by dual lead screws driven by NEMA 17 stepper motors, which translate the block to each specified cut location with high repeatability. Cuts are executed by a dual linear actuator system driving a guillotine-style blade, producing straight, forceful slices through the gel. The system is built on a modular 80/20 aluminum extrusion frame and controlled via an Arduino-based interface with dedicated stepper and actuator drivers. Engineering analysis, including experimental friction and blade force testing, was used to validate the motor and actuator selections and confirm that the system meets its force, torque, and power requirements.

The current prototype successfully demonstrates the core automated slicing workflow. Outstanding items include integration of safety features such as an emergency stop, blade cover, and instruction panel, as well as improvements to wiring and cable management through a dedicated electronics enclosure. Future development will also target automatic fragment localization, enabling the system to generate cut coordinates directly rather than relying on manual user input. Overall, the prototype represents a significant step toward transforming a slow, manual laboratory process into a repeatable, machine-guided operation.

Contents

List of Figures

List of Tables

1 Background

The U.S. Army DEVCOM Armaments Center – Test and Evaluation Division is a research and testing organization within the U.S. Army responsible for the development and evaluation of armaments and munitions. Among its core activities is live-fire ballistic testing, in which projectiles and explosive devices are fired or detonated under controlled conditions to characterize their fragmentation behavior. Understanding how ballistics fragment is critical to evaluating both the lethality of offensive systems and the protective performance of armor and shielding materials.

To capture and preserve fragmentation for post-test analysis, the Armaments Center uses blocks of ballistic gelatin, a transparent, dense material that captures fragments in place upon impact. These blocks serve as a three-dimensional record of the fragmentation event. Once a test is complete, the gelatin blocks must be dissected in a pre-processing workflow designed to locate, extract, and characterize every fragment present.

The current dissection process is performed entirely by hand. A technician manually slices through the gelatin block at incremental intervals, measures the position of any fragments exposed at each cut face, and extracts them for further analysis. This approach presents several significant limitations. First, it is highly labor-intensive, requiring sustained technician attention for the duration of each block's dissection. Second, the quality and consistency of cuts depends on the individual performing them, introducing variability in slice spacing, blade angle, and applied force that can affect the accuracy of fragment localization. Third, the manual nature of the process makes it difficult to scale because as testing volume increases, the dissection bottleneck grows proportionally.

No widely adopted automated solution currently exists for this specific application. While general-purpose food slicers and industrial cutting machines are commercially available, none are designed to handle the physical properties of ballistic gelatin or accommodate the irregular sizes of post-test blocks. The absence of a purpose-built tool means that laboratories conducting this type of testing must continue to rely on slow, inconsistent manual methods, creating an opportunity for a targeted engineering solution.

2 Design Implications and Moral Vision

2.1 Societal and Industrial Impacts

On a societal level, the quality and speed of ballistic studies directly inform design decisions pertaining to munitions and protective armor development, both of which are significant factors in determining soldier survivability and mission completion. By reducing dissection time and improving consistency, the automated slicer contributes significant improvements to this quality and speed. As a result, the added automation contributes to the production of critical data in large quantities that ultimately allows defense engineers to make more informed design decisions.

At an industrial level, this system rather directly addresses a major throughput bottleneck in the process of ballistics analysis. More explicitly, the rate at which high volumes of used gel blocks can be analyzed is contingent on skilled manual labor. By replacing this time-intensive and laborious task wtih an automated solution, this rate can be scaled without necessarily having to scale up with amount of skilled laborers. Furthermore, not only does this system help scale throughput with reduced dependence on manual labor, but the automation aspect also allows those skilled laborers to redirect their attention to other higher-value analytical work including fragment characterization and data interpretation.

2.2 Public Safety

The system ultimately brings a blade mounted onto dual actuators into a laboratory environment, effectively introducing a hazard that otherwise would not exist within the current manual methodology. The cutting system is designed to cut through dense ballistic gelatin and therefore, unintended actuation or operator errors could be cause for serious injury. As a result, this report acknowledges the need for several integrated safeguards including a blade cover, an emergency stop, and an instruction panel to mitigate user errors. It is, however, worth noting that the automated assembly eliminates the factor of human fatigue and moves the operator's hands away from the blade path, thus decreasing vulnerability to the most hazardous aspect of the workflow.

2.3 Accessibility and Economic Impact

One of the more pertinent design criterion for the automated assembly is a low training requirement that will allow new technicians to become operators with little instruction. This effectively widens the threshold for entry into the gelatin processing role irrespective to skill level. Consequently, the simplicity of the systems operation significantly increases accessibility and potential for contribution to the workflow for all technicians while also reducing the extent to which gelatin processing becomes hyper-dependent on very few skilled technicians.

On the economic front, the automated system is intended to reduce human labor time by up to 50%. Taking the rough estimate that a new munition study may require the analysis of 500 blocks, the manual workflow would demand thousands of technician-hours that could effectively be accomplished with half the time and less military personnel. This would provide significant increases in labor efficiency at the modest cost of the prototypes primarily consisting of 8020 aluminum extrusions, Nema 17 motors, linear actuators and an Arduino microcontroller.

2.4 Environmental Considerations and Life Cycle Analysis

The current conceptual design of this project intends to incorporate the use of electromechanical components, structural supports, and motion hardware to produce a precision slicing system capable of creating clean cuts into a ballistic gelatin block. The prototype uses 80/20 aluminium extrusion for structural components, T8 lead screws for the block translation, NEMA 17 stepper motors for motion, custom high-infill PLA actuator mounts, linear actuators and Arduino electronics for control.

- The 80/20 aluminum extrusion frame forms the primary structural backbone of the system. Aluminum is highly recyclable at end of life and offers excellent strength-to-weight ratio and corrosion resistance, but its initial production remains energy-intensive due to the electrolysis processes required for primary aluminum smelting.
- Custom actuator mounts were fabricated from high-infill PLA via 3D printing. These parts are low-cost and replaceable, which reduces the consequence of mechanical failure but increases cumulative plastic consumption over the system's service life.
- The Arduino controller, stepper driver boards, actuator relay board, and associated wiring constitute the most environmentally sensitive portion of the system Proper disposal requires routing through a certified e-waste recycler; disposal in general waste streams is discouraged and may be regulated depending on the operating facility.

Overall, no materials currently considered pose acute toxicity hazards during normal operation. The main environmental concerns for this project are primarily due to the energy intensity of metal production, the plastic consumption introduced by the linear actuator mounts, and the e-waste burden introduced by the Arduino=based electronics and motor controllers. The gelatin blocks being processed also pose a concern despite its disposal being primarily the responsibility of the operating facility and associated environmental protocols.

Component	**Primary Material**	**Environmental Considerations**	**End-of-Life Path**
80/20 Extrusions	Aluminum alloy	• High embodied energy • Corrosion resistant • Abundant material	Highly recyclable
Stepper motor	Copper, steel, magnets	• Mining impacts • Rare earth content • Mixed material assembly	Partial recycling; e-waste stream
Actuator mounts	PLA	• Renewable feedstock • Rarely composted in practice • Low-cost and replaceable	General waste; industrial compost if available
Fasteners	Steel	• Low toxicity • Standardized manufacturing • Widely recyclable	Recyclable

Table 1: Qualitative life cycle considerations for primary system materials

2.5 Moral Vision Statement

This system is designed to automate and improve a process that informs how the US military determines the lethality and effectiveness of its ballistics. The team actively acknowledges that this work exists within a military context and also acknowledges the extent to which its contributions support offensive capability. However, the team also acknowledges the extent to which the systems contributions also support the designs of protective-systems as the same analysis support both branches of research. The ultimate goal is to increase the quality and quality of the data used to inform this analysis.

3 Client Needs and Design Criteria

3.1 Client Needs

The client needed a faster, lower-labor method for recovering ballistic fragment data from gelatin blocks without relying on full manual dissection. Current dissections require approximately 15-60 minutes per block, multiple technicians, and experienced personnel, which limits throughput during test campaigns involving up to 500 blocks. The system must recover the same core information used in existing analysis workflows, including fragment location, mass or mass estimate, material type, and approximate shape.

These needs directly informed the design criteria. The need for usable fragment data led to criteria for location accuracy, mass determination, material classification, and shape categorization. The need to reduce labor and support large test campaigns led to criteria for processing time, throughput, and low operator training. The need for range deployment led to criteria for portability, manageable data volume, realistic computing requirements, and compatibility with existing ATF lethality-modeling workflows.

3.2 Design Criteria

The primary design criteria are the requirements necessary for the design to satisfy the client's main goals. The system must locate fragments with millimeter-level accuracy so that results remain comparable to existing coordinate-based analysis methods. It must also determine fragment mass, either directly or by estimating volume and combining it with material class, because mass is a required input for lethality modeling. Material classification must distinguish common fragment materials such as lead, copper, and steel, while shape categorization must assign each fragment to a useful geometry class such as splinter, plate, chunk, or petal.

The system must reduce human labor time per block by more than 50%, using the current manual dissection process as the baseline. Data volume must remain manageable, specifically avoiding file sizes above 10 GB and avoiding heavy post-processing requirements, since previous CT-based workflows produced large files and software instability. The system must also be portable, safe, and simple enough for live-fire range operation. A low training requirement is necessary so that new operators can use the workflow with minimal instruction rather than relying on highly experienced technicians.

The secondary design criteria are 3D visualization and cost efficiency. Lightweight 3D renderings would help analysts interpret fragment locations and wound structure, but they are not required for the system to recover the core data. Cost efficiency is also secondary because lower long-term testing cost is desirable, but technical accuracy, speed, and range practicality are higher priorities.

3.3 Summary Table

Criterion	Description	Target / Performance Goal	Goal(s) Addressed
3D Fragment Location	Determine resting coordinates of all fragments with millimeter-level accuracy	Locate fragments within ±5 mm of their true position	Accuracy, model compatibility
Mass Determination	Measure or infer mass using volume and material class	Estimate fragment mass within ±10% of measured mass	Accuracy, lethality modeling
Material Classification	Classify fragments as lead, copper, steel, etc.	Correctly classify at least 90% of major fragment material types	Accuracy, usability
Shape Categorization	Assign fragment geometry class such as splinter, plate, chunk, or petal	Correctly assign at least 85% of fragments to a repeatable shape category	Accuracy, modeling, traceability
Processing Time Reduction	Reduce human labor time per block	Reduce manual processing time by greater than 50% compared to the current process	Throughput, scalability
Manageable Data Volume	Avoid excessive file sizes and heavy post-processing requirements	Keep scan/output files below 10 GB per gelatin block	Practicality, computation
Range Operability	Equipment must be portable, safe, and simple at a live-fire range	Allow setup and takedown by two operators in under 30 minutes without specialized equipment	Logistical feasibility
Low Training Requirement	Workflow usable by new operators with minimal instruction	Allow a new operator to complete the workflow after 15 minutes or less of instruction	Usability, manpower reduction
3D Visualization (Secondary)	Provide optional lightweight 3D renderings	Generate a usable 3D visualization in under 5 minutes per block	Interpretability
Cost Efficiency (Secondary)	Reduce long-term testing costs	Reduce recurring labor or processing cost by at least 25% relative to the manual method	Budget, program efficiency

Table 2: A summary of all design criteria considered.

4 Function Analysis and Journey Map

4.1 Function Analysis

A successful design must perform a set of functions that together convert a gelatin block into usable fragment data. These functions are stated in broad terms so that the analysis remains independent of any particular design.

The primary functions are those directly involved in recovering fragment data and the supporting infrastructure on which that recovery depends. These functions appear on the primary FAST diagram (Figure 1). The system must:

- Secure the block in a stable, repeatable reference frame
- Locate fragments within the block volume
- Determine fragment mass either directly or by inferring it from volume and material class

- Classify fragment material
- Categorize fragment shape into a useful geometry class
- Provide alignment reference
- Accept operator input
- Supply energy
- Generate motion

The supporting functions do not produce fragment data but are necessary for the system to be safe, deployable at a live-fire range, and serviceable between tests. The system must:

- Enclose hazards
- Be transportable
- Allow for disassembly

Each of the above functions maps directly onto one or more of the client needs and design criteria described in the previous section.

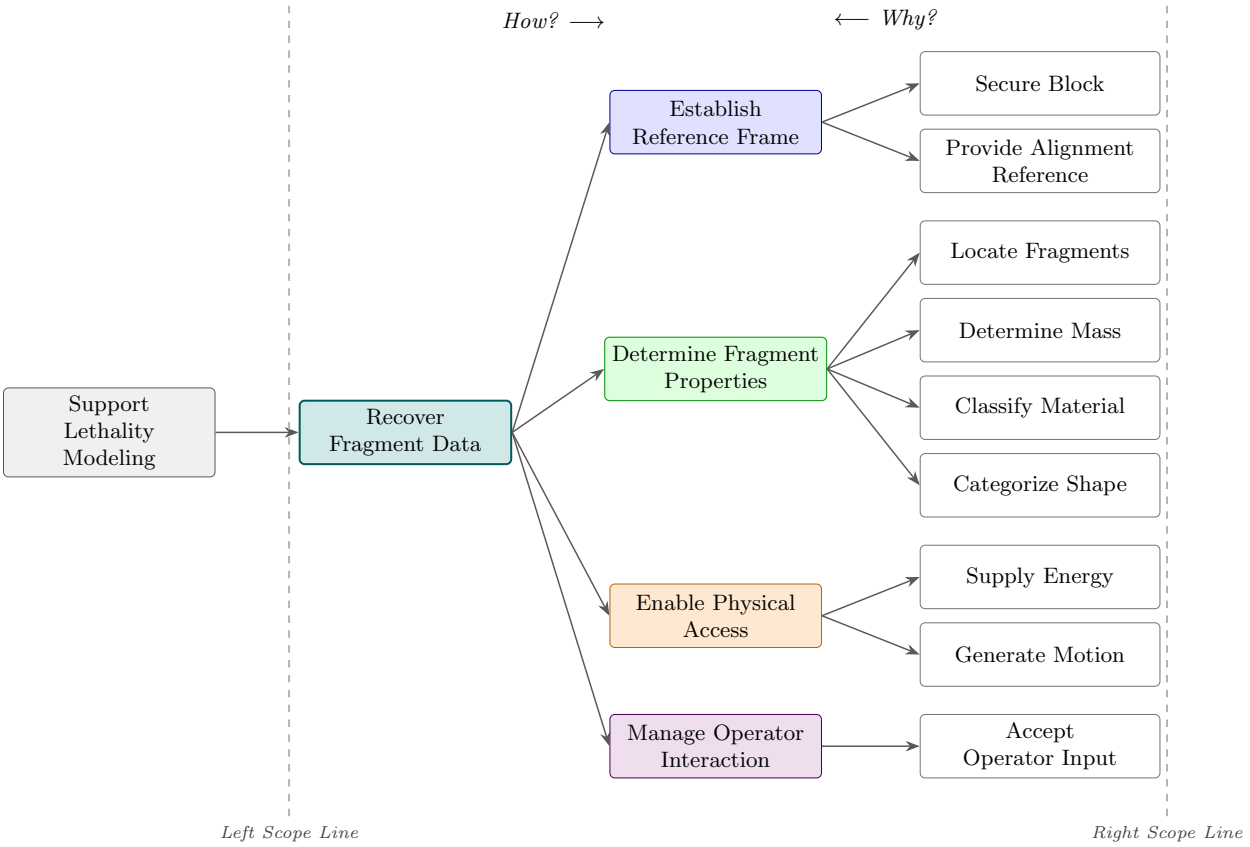


Figure 1: FAST diagram illustrating primary functions for the gelatin dissection system

4.2 Journey Map

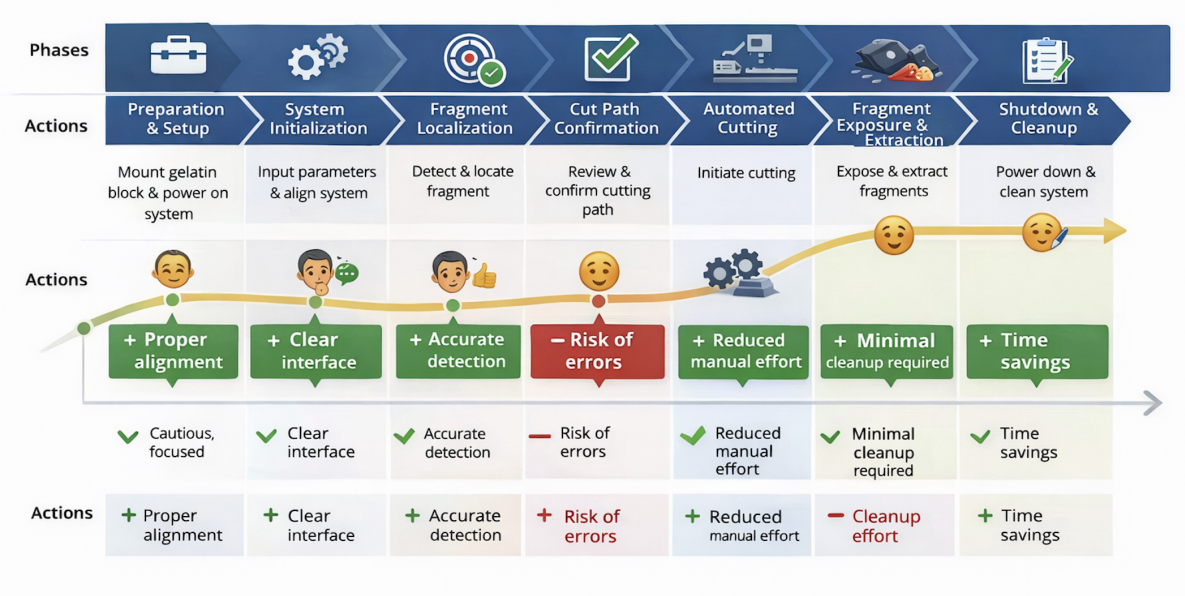


Figure 2: Journey map illustrating the ballistic testing technician's interaction with the gelatin dissection system across all primary phases of use

4.3 Influence on Design and Ideation

The function analysis and Journey Map influenced both how the team ideated and what the team converged on. Form-neutral functions in the FAST diagram produced five independent ideation topics, which were eventually narrowed down to the cutting-mechanism topic (Topic 2). These were further evaluated via a Pugh matrix and dot vote into the final coordinate-driven single-axis slicer.

The Journey Map influenced three specific design choices: the multi-fail-safe method of completing a cut (the controller has to explicitly sequence Zero $\rightarrow$ Load $x_i \rightarrow$ Run), the cleanup phase influenced the non-stick surfaces and separation of electronics, and the goal of minimizing operator skill influenced the physical x-axis ruler.

5 Ideation and Design Selection

5.1 Idea Generation

The team began with a broad brainstorming process to explore as many possible solution directions as possible before narrowing down. Ideas were generated across five functional categories: fragment localization and detection, gelatin cutting and access mechanisms, operator interaction and feedback, labor reduction/workflow efficiency, and cleanup/maintenance. This helped ensure that the team was not only thinking about the core cutting mechanism, but also the full user workflow surrounding setup, operation, safety, data collection, and reset. After the initial brainstorming phase, the ideas were screened to remove concepts that were outside the project scope, too technically complex, or unrealistic given the time and budget constraints. The remaining ideas were then refined and grouped into comparable solution types, such as electromagnetic/inductive detection, physical-contact detection, optical/thermal detection, guided blade systems, wire-based slicers, high-energy cutting systems, digital interfaces, material-handling systems, and cleanup-focused design features.

From these grouped features, the team created complete system-level concepts by combining compatible ideas across categories. Each proposed solution generally paired a fragment localization method with a gelatin cutting method, then added supporting interface, workflow, safety, and cleanup features. This produced ten initial solution concepts, including reference-guided slicing, real-time metal-detection slicing, thermal-imaging slicing, X-ray-guided slicing, basic blade or wire slicing, integrated waterjet cutting, CT scanning, and robotic extraction.

5.2 Downselection Method

To narrow the initial ideas into a smaller number of design solutions, the team used a Pugh matrix plus a dot vote. The Pugh matrix compared each concept against the current process using the team's main design criteria (see Table 8). Each criterion was weighted based on its importance to the client and project constraints, with labor-time reduction and safety given higher weights. Feasibility was also included to prevent the team from selecting a concept that performed well theoretically but could not be realistically created within the available time and budget. This made the downselection process more objective and directly tied each decision to the requirements the final design needed to satisfy.

After the matrix identified the strongest concepts, the team used dot voting to add team judgment and compare the top-ranked options. This combination allowed the team to reduce the number of concepts while balancing measurable design performance, client priorities, and practical implementation concerns.

6 Design and Design Components

6.1 Design Overview

The final design is a coordinate-driven automated slicing machine that reduces the manual labor required to pre-process ballistic gelatin blocks. The operator inputs one or more x-coordinates via a laptop interface, the machine then moves the gelatin block to each commanded position and executes a guillotine-style cut without further operator intervention.

The assembly is organized around four integrated subsystems:

- **Frame and Support:** An 80/20 aluminum frame provides the primary structure. Vertical uprights support the slicing mechanism, while horizontal rails and a flat stainless steel bed support the block-moving mechanism and contain the gelatin block during operation.
- **Block Moving Mechanism:** Two NEMA 17 stepper motors drive parallel lead screws attached to a pushing plate, translating the gelatin along the x-axis.
- **Slicing Mechanism:** A dual-actuator guillotine system actuates the cutting blade vertically through the full block cross-section. Custom high-infill PLA mounts secure the actuators to the 80/20 uprights, and a custom aluminum actuator-to-blade connector ensures force is uniform across the blade.
- **Electronics and Integration:** An Arduino-based controller coordinates stepper motor drivers and the actuator relay. The operator interface runs on a laptop connected via USB serial.

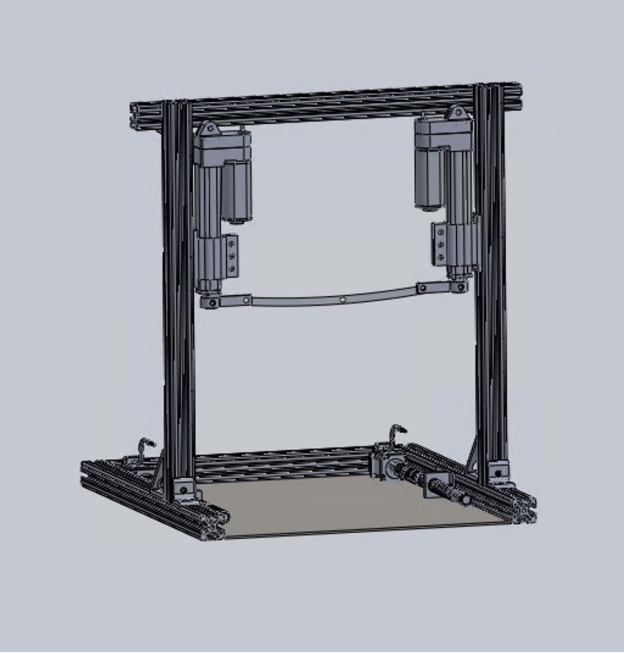


Figure 3: Solidworks Assembly Model

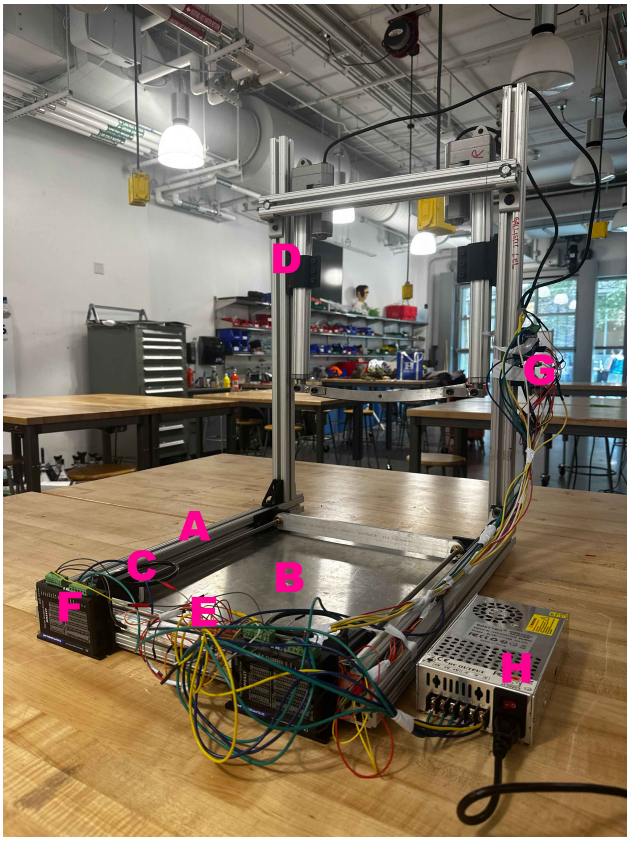


Figure 4: Photograph of the prototype assembly. Major components: (A) 80/20 aluminum extrusion frame, (B) stainless steel gelatin tray, (C) lead screw and NEMA 17 stepper motor (block moving mechanism), (D) vertical 80/20 uprights with linear actuators (slicing mechanism), (E) Arduino controller, (F) stepper motor driver boards, (G) actuator driver boards, and (H) Power supply (see Figure 22 for full wiring diagram)

6.2 Component Descriptions

6.2.1 Component 1: Frame & Support

The structural frame of the system is constructed using modular 80/20 aluminum T-slot extrusions, selected for their strength-to-weight ratio, ease of assembly, and adaptability. The frame consists of six primary members forming a rigid base and vertical support structure. The base is arranged in a U-shaped configuration using three horizontal extrusions, joined through a combination of drop-in T-nuts, concealed brackets, corner L-brackets, and gusset plates to ensure alignment and structural rigidity. A stainless steel base plate is mounted within this perimeter and serves as the support surface for the ballistic gelatin block. This material was selected for its durability, corrosion resistance, and low surface friction, allowing consistent translation of the gelatin during operation. The plate is secured using M6 fasteners through nine evenly distributed holes, ensuring uniform load transfer and minimizing deformation under combined loading conditions of approximately 600 N. Two vertical extrusion members are mounted orthogonally at the blade end of the frame and provide the primary support for the slicing mechanism. These uprights utilize drop-in T-nuts to mount custom 3D-printed actuator brackets, allowing adjustable positioning and alignment of the linear actuators. A horizontal cross-member connects the uprights, increasing lateral stiffness and preventing structural deflection under actuator loads. Overall, the frame serves as the primary load-bearing backbone of the system, maintaining alignment between subsystems while enabling modular assembly, ease of fabrication, and future design modifications.

6.2.2 Component 2: Block Moving Mechanism

The aluminium pushing plate is driven along the x-axis by two StepperOnline 17HS26-2304S NEMA 17 motors (1.8°, 79 N·cm holding torque, 2.3 A/phase), each coupled through an MH2025-8-8 8 mm→8 mm flexible jaw coupling to a 400 mm Tr8x8 lead screw with a T8 brass nut. Two parallel lead screws were chosen over a single centered screw to keep the resultant pushing force even. A single shared STEP/DIR pair (see *Electronics and Integration*) drives both motors in lockstep so that the screws cannot diverge and skew the plate. The lead screws are supported at the far end by bearings and at the motor end by the motor shaft itself. The brass nuts are bolted into the back face of the pushing plate. The pushing plate is aluminium, sized to the inside cross-section of the bed. The team experimented with various materials but ultimately selected aluminum 6061 after conducting analysis and testing on its strength capabilities. With an 8 mm lead and the 1/16 microstepping configured on the DM542T drivers, the linear resolution is ≈400 steps/mm.

6.2.3 Component 3: Slicing Mechanism

The guillotine is built around two Progressive Automations PA-09 Feedback Mini Industrial Actuators (12-inch stroke, 24 VDC, 330 lb rated load, integrated Hall-effect feedback at 660 pulses/inch). Two actuators were used in place of a single centered actuator because the dual configuration applies cutting force at both ends of the blade simultaneously, avoiding uneven cutting. Each actuator is mounted by a custom high-infill PLA mount and the mount is secured to the 80/20 uprights. The actuator rod ends have a custom actuator-to-blade connector made of 6061 aluminum with a bolt hole drilled out to match the geometry of the blade. To ease machining complexity, these mounts feature a tube-shaped actuator connection and welded flanges that support the blade. The blade itself is from a paper guillotine. Synchronization, end-of-travel detection, and stall-based homing of both actuators are handled in firmware via the IBT-2 drivers and Hall-feedback returns described in the Electronics and Integration subsection.

6.2.4 Component 4: Electronics and Integration

The control system consists of an Arduino Uno R3, two NEMA 17 stepper drives for the x-axis, two H-bridge-driven linear actuators for the guillotine, a single 24 V power supply, and a USB serial link to the operator laptop. A complete wiring diagram is given in the appendix.

Controller selection
An Arduino Uno R3 was selected due to its simplicity and availability. The control task wasn't too ambitious: two synchronized step-pulse axes, four digital outputs to drive two H-bridges, and four digital inputs reading Hall-effect feedback. The Uno provides the step-pulse generation using the *AccelStepper* library. Its 5 V logic is directly compatible with both the DM542T inputs and the IBT-2 PWM/enable inputs.

Stepper drive
The x-axis is driven by two StepperOnline DM542T digital stepper drivers, each configured for 16 microsteps per full step (giving $200 \times 16 = 3200$ pulses per revolution, matching the *TRAVEL_PULSES_PER_REV* constant in firmware) and 2.3 A peak phase current to match the StepperOnline 17HS26-2304S NEMA 17 motors (1.8°, 79 N·cm holding torque, 2.3 A/phase). The two drivers receive a single shared STEP/DIR pair from the Arduino (pins D2/D3 to the left driver, D5/D6 to the right driver). The DM542T was chosen because of the high continuous current capability needed for the 79 N·cm motors. Each motor shaft couples to a Tr8x8 lead screw (8 mm lead, 400 mm length) through an MH2025-8-8 8 mm→8 mm jaw coupler. The resulting linear resolution is 3200/8 mm ≈ 400 steps/mm (10,160 steps/inch).

Linear actuator drive
Each guillotine actuator is a Progressive Automations PA-09 Feedback Mini Industrial Actuator (12-inch stroke, 24 VDC, with built-in Hall-effect feedback nominally rated at 660 pulses per inch). Each actuator is driven by its own BTS7960 43 A IBT-2 H-bridge module rather than a single shared driver. This allows the firmware to throttle one actuator independently when the Hall-feedback synchronisation error between the two exceeds the tolerance defined by `CUTTER_SYNC_TOLERANCE_PULSES` (12 pulses, ≈0.018 inch), preventing any skewing. Each board's R_EN and L_EN pins are tied directly to +5 V (always-enabled), and the RPWM

and LPWM inputs are driven by the Arduino as plain digital outputs (D8/D9 for the left actuator, D10/D11 for the right). The firmware enforces a software interlock that never asserts both RPWM and LPWM on the same board.

Hall-effect feedback
The PA-09's two feedback wires (Yellow = Signal 2, White = Signal 1) are connected to Arduino digital inputs configured *INPUT_PULLUP*: D12/A0 for the left actuator, A1/A2 for the right. The firmware counts rising edges on the active-direction line only and uses the counts to (a) command extension to a fixed stroke target, (b) detect mechanical end-of-travel during retract-homing via a pulse-stall timeout (*CUTTER_HALL_STALL_MS*, 250 ms), and (c) keep the blade parallel between the two actuators throughout the stroke.

Power
A single 24 V, 20 A, 480 W enclosed switching power supply was selected to feed both the DM542T drivers (rated 18-50 VDC input) and the BTS7960 H-bridges (rated 6-27 VDC) from a common rail. Worst-case current draw is approximately 4.6 A for both steppers running simultaneously plus a measured $\sim$3 A per actuator under load, leaving substantial headroom inside the 20 A PSU rating. The Arduino is powered separately from the host laptop's USB so that logic-side ground is electrically isolated from motor-side ground up to the DM542T inputs and the IBT-2 logic-supply pins. The two grounds however are connected to give a common reference required for STEP/DIR signaling.

Operator interface
The operator interacts with the system through a USB-B cable from the Arduino to a laptop running the Arduino IDE Serial Monitor at 115200 baud. The following commands are available: `ZERO`, `LOAD` $x_1, x_2, \ldots$, `ADD` x, `CLEAR`, `LIST`, `MOVE` x, `TESTROT`, `TESTCUT`, `CUTEXTEND`, `CUTRETRACT`, `CUTSTOP`, `RUN`, `STOP`, `STATUS`, and `HELP`.

7 Engineering Analysis

7.1 Models and Simplifications

Block Pushing: Lead Screws and Motors

The block pushing mechanism is comprised of both a push plate that interfaces with the block and the lead screw and motor assembly that mechanically powers its motion. To size lead screws and motors, the force required to push the gel block along the stainless steel surface is relevant. Friction between the block and stainless steel base plate is modeled using Coulomb friction with a constant coefficient. The dual lead screw configuration is assumed to distribute the load evenly between both screws, such that each screw carries half of the total required force. The lead screws are modeled as ideal power transmission elements with constant mechanical efficiency, and losses due to backlash, misalignment, and compliance are neglected.

Block Pushing: Push Plate

The push plate within the block moving mechanism can be represented by a simply supported beam with the capacity to deform and experience stress. In early calculations, simple supports are sufficient to calculate necessary parameters; however finite element analysis utilizes a more realistic geometry. Additionally, in hand calculations, the distributed load of the block against the plate is simplified to a point load at the center due to the non-uniform distribution of load.

Block Slicing

The block slicing subassembly is comprised of dual linear actuators, supports, blade mounts, and the blade itself. The blade and actuators were assumed to be rigid components, while the blade mounts interfacing them were assumed to be capable of deformation.

Base Plate and Frame

The frame is modeled as a rigid, non-deforming structure, while the stainless steel plate is assumed to provide a flat, uniform surface with the capability to deform under the weight of the block. Dynamic effects such as vibration and frame compliance are neglected, and all loads are treated as quasi-static.

7.2 Assumptions

- The gelatin block behaves as a rigid body during translation.
- Friction between the gelatin and base plate is constant and velocity-independent.
- Load is evenly distributed between both lead screws.
- Lead screw efficiency is constant and known.
- Stepper motors operate within their rated torque range with no step loss.
- Cutting force can be represented as a single resultant force.
- Actuator loads are symmetric in the dual-actuator configuration.
- Frame is rigid and does not deform under load.
- Dynamic effects (vibration, impact, transient loads) are negligible.

7.3 Governing Equations

The following equations were used to model the translation and cutting behavior of the system.

Block Translation (Friction Force)

$$F_{\text{req}} = \mu m g \tag{1}$$

Force per Lead Screw

$$F_{\text{screw}} = \frac{F_{\text{req}}}{2} \tag{2}$$

Lead Screw Torque

$$T_s = \frac{F_{\text{screw}} L}{2\pi\eta} \tag{3}$$

Motor Torque (Direct Drive)

$$T_m = T_s \tag{4}$$

Power Required

$$P = T \cdot \omega \tag{5}$$

Cutting Force Distribution (Dual Actuator)

$$F_{\text{actuator}} = \frac{F_{\text{cut}}}{2} \tag{6}$$

Deflection of Simply Supported Beam with Center Load

$$y_{\max} = \frac{-Fl^3}{48EI} \tag{7}$$

7.4 Analysis Methods and Results

Block Pushing: Lead Screws and Motors

To analyze and size lead screws and motors, simple calculations are sufficient. After determining a coefficient of friction of 2.243 experimentally, the team was able to calculate the force required to slide the block across a metal plate:

$$F_{\text{req}} = \mu m = 2.243 \cdot 27.94 \text{ lb} = 62.664 \text{ lbf} \tag{8}$$

Since the block is pushed equally from two ends, this load can be split to apply 31.332 lbf of force to each lead screw assembly. From there, required lead screw torque can be calculated:

$$T_s = \frac{F_{\text{screw}} L}{2\pi\eta} = \frac{31.332 \cdot 0.0393701}{2\pi \cdot 0.4} = 0.491 \text{ lbf} \cdot \text{in} \tag{9}$$

where F is the lead of the screw and η is lead screw efficiency.

Finally, required mechanical power was solved for given previous calculations:

$$P = T \cdot \omega = 0.491 \cdot 63.83712825 \text{ rad/s} \cdot 0.113 = 3.540 \text{ watts} \tag{10}$$

where 0.113 is a conversion factor.

Force to push block	62.664 lbf
Torque	0.491 lbf·in
Mechanical power	3.540 watts

Table 3: Calculated Values for Lead Screw and Motor Analysis

Block Pushing: Push Plate

Push plate analysis includes both hand calculations and finite element simulations. One of the most important criteria to design around in this component was deflection, since precise locations are expected from this system. Deflection was initially calculated by hand:

For PLA, Young's modulus is estimated at 2 GPa, or 290075 psi

$$y_{\max} = \frac{-Fl^3}{48EI} = \frac{-63 \cdot 11.5^3}{48 \cdot 290075 \cdot 0.065} = 0.106 \text{ in} \tag{11}$$

For Aluminum 6061, a much lower deflection was calculated as expected:

$$y_{\max} = \frac{-Fl^3}{48EI} = \frac{-63 \cdot 11.5^3}{48 \cdot 10000000 \cdot 0.065} = 0.00307 \text{ in} \tag{12}$$

After preliminary hand calculations, finite element simulations were run to observe deflection and factor of safety.

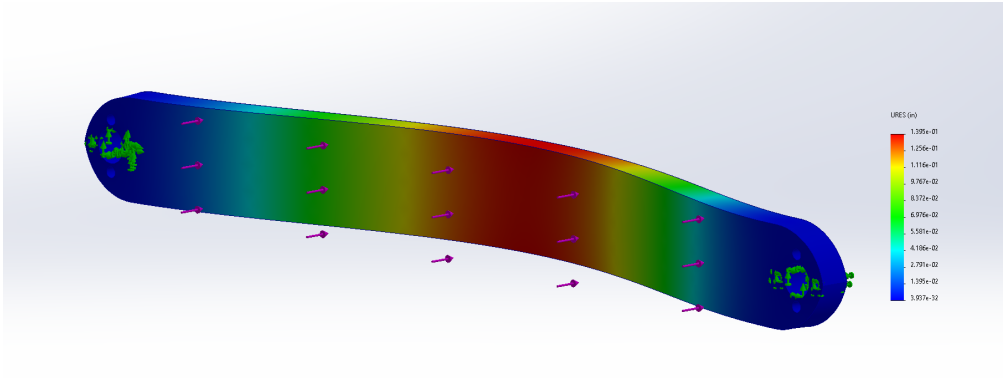


Figure 5: Static Study Results on 3D Printed Push Plate - Displacement

Simulations showed a maximum displacement in the PLA plate of 0.1395in, closely aligning with hand calculations.

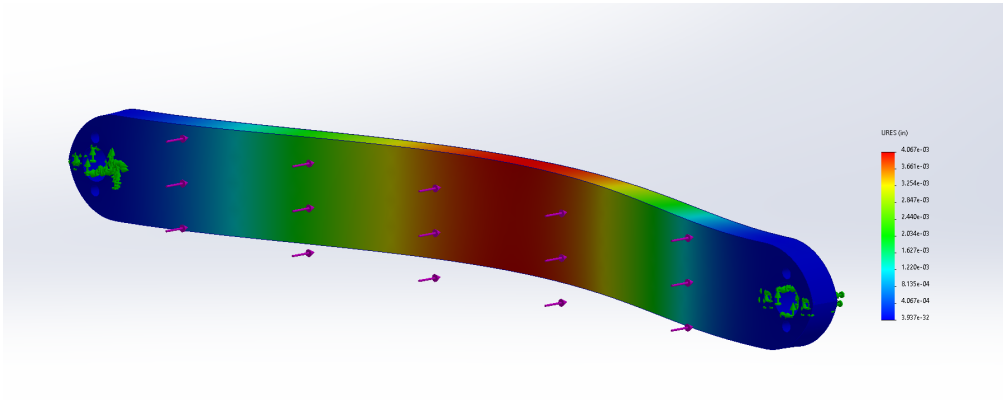


Figure 6: Static Study Results on Al 6061 Push Plate - Displacement

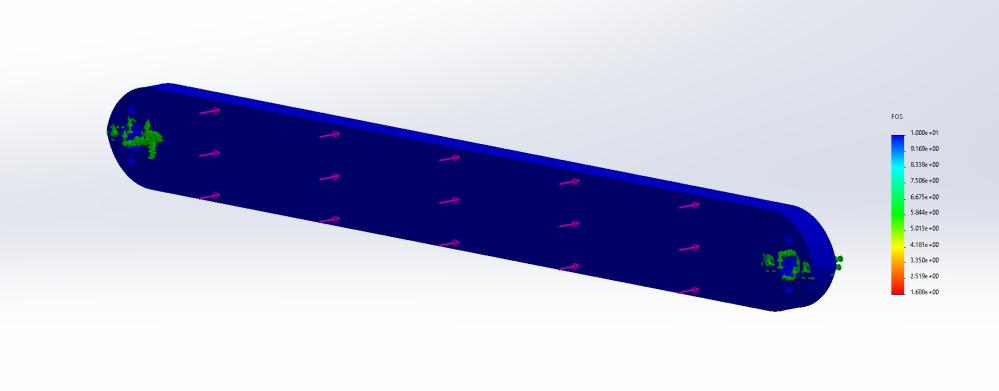


Figure 7: Static Study Results on Al 6061 Push Plate - Factor of Safety

Finite element analysis in the aluminum plate revealed a displacement of 0.00407 in, which is slightly larger than hand calculations but still acceptable. Additionally, simulations found a minimum factor of safety of 1.7. However, the area of failure in the static study was at the bolt holes which indicates a simulation setup shortcoming since yielding factor of safety is orders of magnitude higher at all other areas.

Calculated deflection (PLA)	0.106 in
Calculated deflection (Al 6061)	0.00307 in
Simulated deflection (PLA)	0.1395 in
Simulated deflection (Al 6061)	0.00407 in
Simulated factor of safety (Al 6061)	1.7 (see "Limitations")

Table 4: Push Plate Analysis

Block Cutting

Since the block slicing mechanism includes multiple components, several forms of analysis were utilized. For simplified cutting force and actuator sizing, hand calculations were sufficient.

Force on these components was calculated based on experimentally determined loads for block slicing. A required cutting force of 21 lbf was estimated from testing, which was translated into 11 lbf per actuator assuming evenly distributed load between left and right supports.

For blade mounts, the geometry was too complex to conduct hand calculations accurately. Instead, finite element simulations were conducted to observe deformation in both PLA and aluminum models.

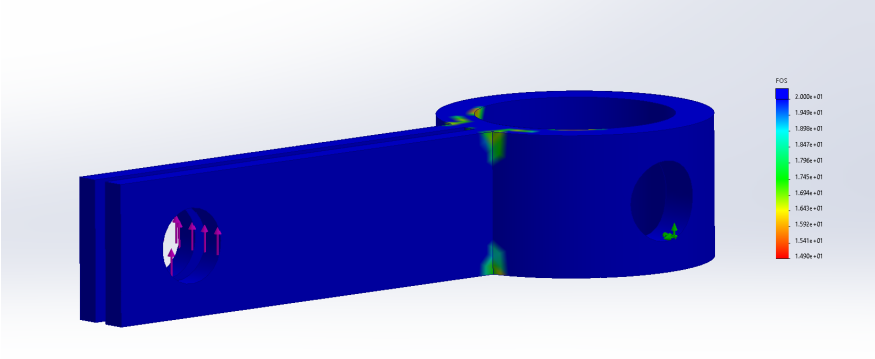


Figure 8: Static Study Results on Aluminium Blade Mount - Factor of Safety

Static simulation results on the aluminum blade mount resulted in a minimum factor of safety of 14.9 which was sufficient to account for any unknowns in cutting force given differing gel densities. Studies were also conducted on PLA mounts with varying levels of success due to uncertain material properties. Some simulations showed a maximum stress of 18.1 MPa, which given an estimated ultimate tensile stress of 30 MPa produced a factor of safety of 1.66.

Aluminum blade mount factor of safety	14.9
PLA blade mount factor of safety	1.66

Table 5: Block Slicing Analysis

Base Plate and Frame

Since most components in the frame are assumed to be rigid, finite element analysis of the base plate was sufficient to ensure excessive deformation would not occur.

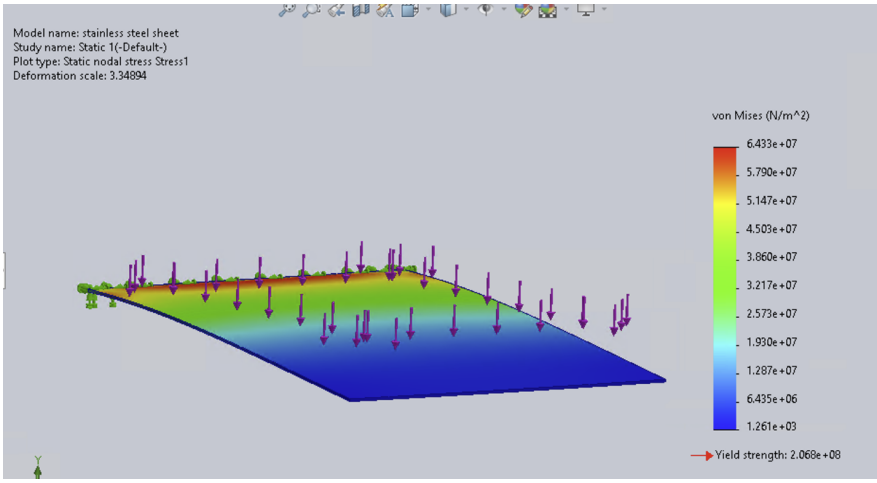


Figure 9: Static Study Results on Base Plate - Stress

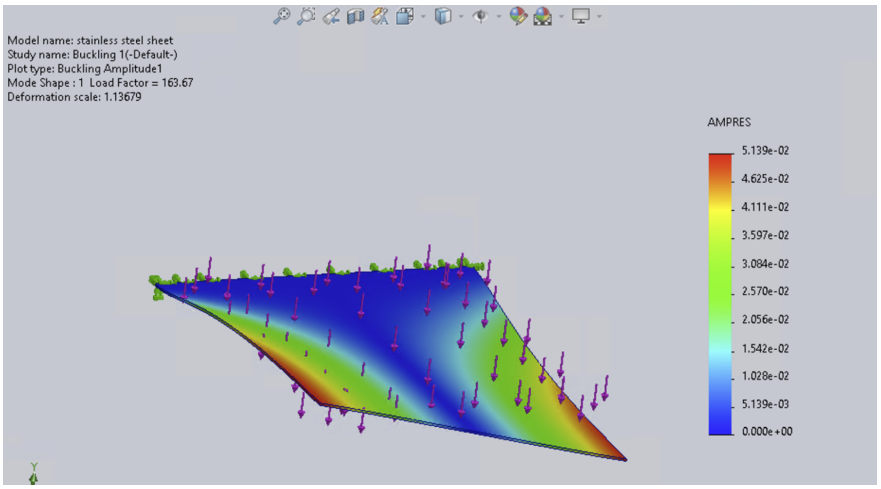


Figure 10: Static Study Results on Base Plate - Buckling

Base plate factor of safety	3.2
Buckling load factor	163

Table 6: Base Plate Analysis

7.5 Limitations

There were several notable limitations in component analysis including but not limited to simulation setup, modeling of PLA, and unknown loads.

Given that all loads were determined experimentally in testing, there was some uncertainty surrounding consistency between batches of gel. The team found that the second block of gel was noticeably tougher than the first, demonstrating some discrepancies between batches. To mitigate this, higher factors of safety were prioritized to account for outlier batches.

Beyond load uncertainty, some material properties were also difficult to simulate. PLA and other 3D print materials are considered non-uniform to the extent that moduli of elasticity are not standardized. Within Solidworks static simulations, factor of safety plots cannot be defined for some types of plastic due to these uncertainties limiting the team's ability to predict failure in that way. The team used a conservative estimate for these components and many of them ended up being replaced with metal regardless.

Within finite element simulations, bolted joints are notably difficult to simulate as well. Within the scope of this analysis, most joints were modeled as fixed geometry which gives rise to minor simulation failures at fixed faces. Because of this, some factors of safety such as that of the push plate were considered acceptable despite being relatively low since the expected failure point was unrealistic.

There were also some components such as the actuator mounts that were not thoroughly analyzed at all due to uncertainty in loading methods. In future work, these components should be both redesigned and re-analyzed for failure modes that came up later in the prototyping process.

7.6 Design Implications and Performance Predictions

This analysis had significant impacts on design choices, especially when paired with preliminary testing results.

Baseline calculations for torque required to move the block across a plate were directly used to select motors for use in that mechanism. Applying an additional design factor of safety of 3, the team was able to size and select the NEMA 17 motors and the 8mm lead screws featured in this subassembly.

A similar process was used to size actuators, with the caveat that stroke length was the most critical property of actuation. The Progressive Automations actuators featured have a load capacity of 330 lb which is far more than the required force, however no "weaker" options were available with the required 12 in stroke length.

Calculations and simulations on both the push plate and blade mounts showed that aluminum components would be more durable than PLA, which was to be expected. These calculations further predicted failure in PLA components which was critical to understand before final prototyping.

8 Engineering Standards

The design and operation of the automated ballistic gelatin slicing system is informed by several engineering standards spanning occupational safety, machinery guarding, hazardous energy control, and electrical safety. Because the system is intended for deployment at a U.S. Army DEVCOM facility, the applicable standards reflect both general industrial machinery requirements and the more stringent safety and health requirements imposed on Army and Department of Defense operations.

8.1 Occupational Safety and Health: EM 385-1-1

U.S. Army Corps of Engineers Safety and Health Requirements Manual, EM 385-1-1 (30 November 2014 Edition)
EM 385-1-1 is the primary occupational safety and health standard governing U.S. Army Corps of Engineers activities and operations. The manual states that it "prescribes the safety and health requirements for all Corps of Engineers activities and operations" and that its applicability "extends to occupational exposure for missions under the command of the Chief of Engineers, whether accomplished by military, civilian, or contractor personnel." As a system designed for deployment at an Army DEVCOM testing facility, the gelatin slicing machine falls within this scope. The most directly relevant provisions are found in Sections 11, and 18, each of which is addressed below.

8.2 Electrical Safety: EM 385-1-1, Section 11

EM 385-1-1, Section 11: Electrical
Section 11 governs all electrical wiring, equipment, and operations within the scope of EM 385-1-1.Section 11.A.02.d states that "live parts of wiring or equipment shall be guarded to protect all persons or objects from harm." In the current prototype, the stepper driver boards and actuator relay boards are not yet mounted within a dedicated electronics enclosure. Integration of these enclosures, which are also outstanding, is required to satisfy this provision.

8.3 Machinery Guarding: EM 385-1-1, Section 18

EM 385-1-1, Section 18: Vehicles, Machinery and Equipment
Section 18 of EM 385-1-1 governs the operation, guarding, and safety devices required for machinery and mechanized equipment. Section 18.B.03.a states that "all belts, gears, shafts, pulleys, sprockets, spindles, drums, flywheels, chains, or other reciprocating, rotating, or moving parts of equipment shall be guarded when exposed to contact by persons or when they otherwise create a hazard." This provision in particular directly applies to the guillotine blade assembly. The outstanding blade cover, in accordance to this provision, is a crucial requirement that must be fulfilled before this system can be cleared for deployment.

Additionally, Section 18.B.08 states that "no guard, safety appliance, or device shall be removed from machinery or equipment, or made ineffective, except for making immediate repairs, lubrications, or adjustments, and then only after the equipment has been de-energized and Hazardous Energy Control Program (lockout/-tagout procedures) implemented. All guards and devices shall be replaced immediately after completion of repairs and adjustments and before power is turned on." While this provision does not directly dictate the system design or components it is essential for the safe operation and maintenance of the ballistic gel slicer assembly by necessitating that the system be shut off before a safety can be removed for repair or lubrication which are critical for system function.

8.4 Environment and Sustainability: ISO 14006

ISO 14006, Environmental management systems: Guidelines for incorporating ecodesign
ISO 14006 provides a framework for integrating environmental considerations into product design and development in addition to emphasizing the incorporation of environmental criteria throughout the product lifecycle. This framework was directly applied to many of the design choices dictating material use and end of life paths for several components as part of the final prototype. For example,the aluminum extrusions for

the structural frame prioritizes durability and recyclability, and the system's electronic components demand proper disposal via certified e-waste streams.

9 Testing

9.1 Tests Conducted

Early Testing: Friction and Slicing

Testing was conducted throughout the entire design process, including during the preliminary phase. Once the team had access to a block of gel, the first step was experimenting with various cutting devices and observing frictional effects when moving it on a metal sheet.

Mockup Test

Once each subassembly was prototyped for the first time, testing was done on the total functionality and electronic integration. This involved moving the motors and lead screws to varying degrees and making initial cuts into a cured gel block. Several components were still in a mockup stage (3D printed rather than machined), so these tests were expected to be destructive. Mockup tests were not quantitative but rather qualitative; the team was able to observe the performance of each component and determine from observations whether additional improvements were necessary.

Electronics Testing

Electronics testing was used to verify that the wiring, actuator direction, motor direction, and Arduino command logic was all working before final mounting. The team tested the stepper motors with basic movement commands to ensure that both lead screws moved in the same direction, then tested actuator extension and retraction commands separately. The Arduino code was updated to make the command workflow more intuitive. This testing supported the low training requirement, range operability, and processing-time design criteria.

Final Testing and Demonstration

Once components were in their final form, additional testing was conducted to assess full functionality. This included running full execution processes composed of multiple cuts and determined coordinates. The team tested the prototype on both soft materials like Jell-O as well as the hardened gel block used by the client.

9.2 Results and Reliability

Frictional testing was conducted by observing motion of the block across a metal plate held at various angles. From multiple tests, a static coefficient of friction was determined from the average of processed results. Additionally, density of gel blocks was determined based on the mass and dimensions of this particular batch.

Early Testing: Friction and Slicing

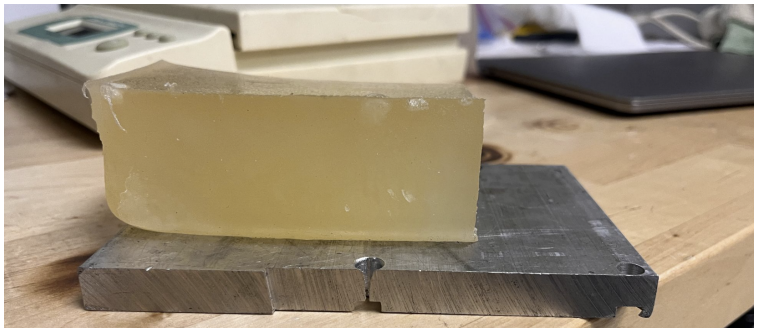


Figure 11: Friction Testing

Experimental static coefficient of friction	2.243
Gel block density	0.036 lb/in^3

Table 7: Friction and Density Test Results

Blade type	**Required cutting force (lbf)**
Paper cutter (guillotine)	21
Serrated knife (jagged side)	15
Serrated knife (wavy side)	16
Electric cutter	6

Table 8: Block Slicing Test Results

Frictional testing and density analysis provided the basis for many critical analysis and calculations for various components, however it proved to be slightly unreliable. The second batch of gel produced by the team was much tougher than the first one, likely requiring higher forces to cut. These discrepancies were mitigated by applying high factors of safety throughout analysis to ensure that blocks of varying toughness could still be moved and cut.

Mockup Testing

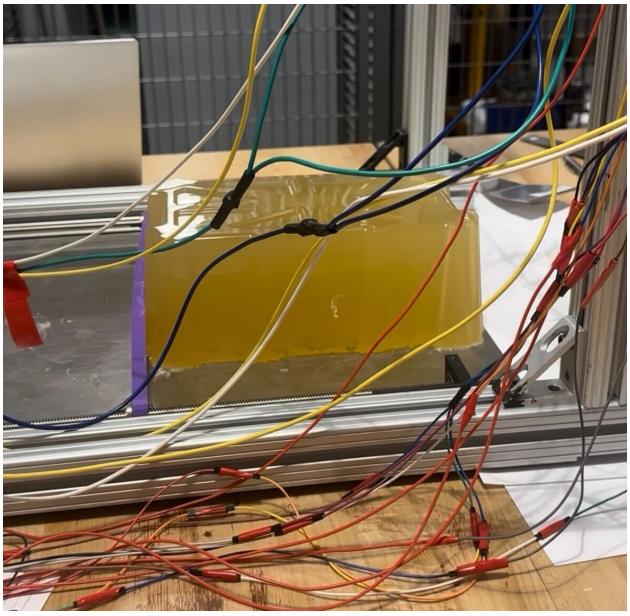


Figure 12: Mockup Push Plate Test

Testing of the push plate was initially conducted with a PLA mockup. As can be observed in the figure above, bending in this member was apparent. The block did slide across the steel plate without additional lubrication and did not pose a large challenge in its general motion. From this test, the team decided that replacement of the PLA push plate was critical.

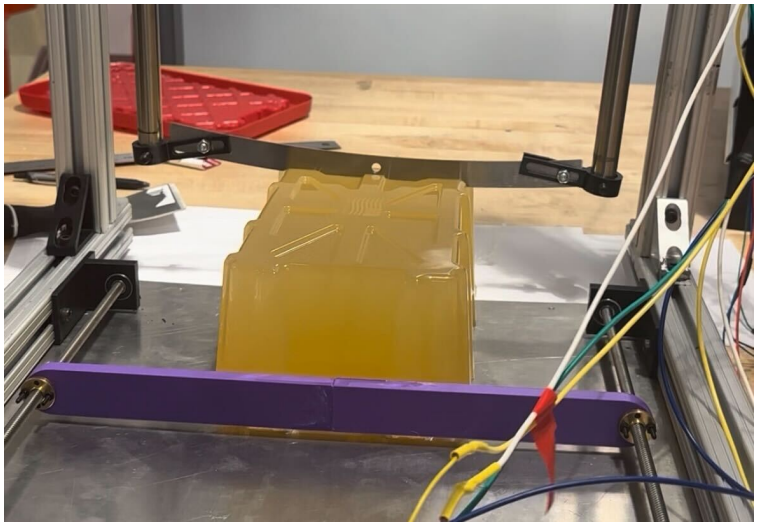


Figure 13: Mockup Slice Test

Testing of the cutting mechanism revealed similar results: PLA mounts experienced catastrophic tearout failure when high load was applied. The team intentionally pushed the prototype beyond force capabilities to reveal failures early on and replace necessary components. After observing almost immediate failure in the blade mounts, switching to aluminum was an obvious next step. This test was highly reliable as it closely followed the actual procedure for fixture usage.

Electronics Testing

One of the main electronics issues that came up was that one actuator would extend but not retract. After checking wiring and continuity, Arduino pins, command logic, and actuator behavior, it was determined that the actuator was functional but its controller was faulty. Replacing the controller fixed the issue, and the team ordered additional spare controllers to prevent a similar failure from delaying future testing.

Final Testing and Demonstration

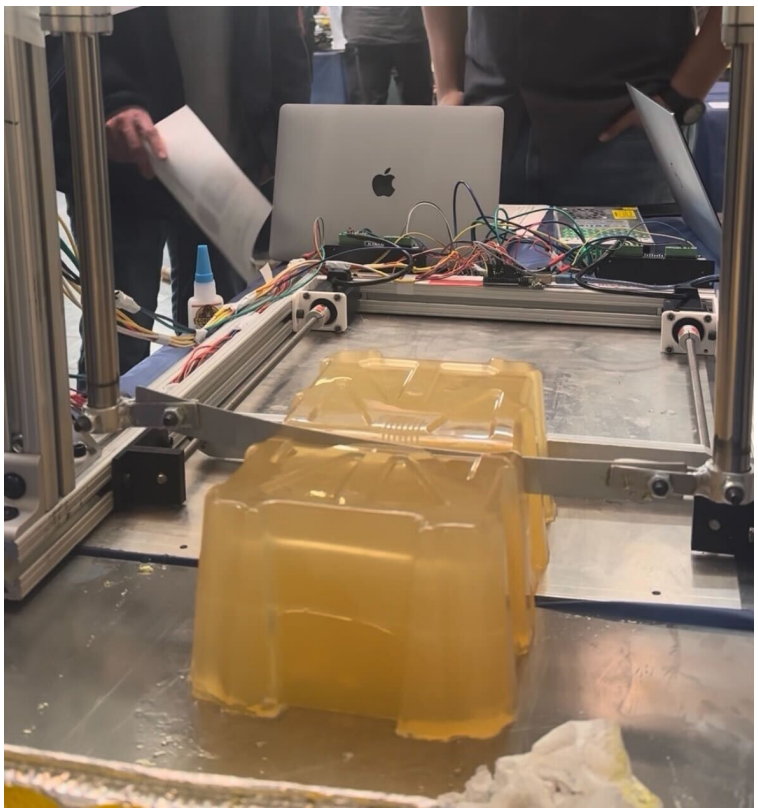


Figure 14: Final Prototype Testing

Final testing replicated the process of mockup testing with stronger machined components. The push plate and block motion mechanism worked seamlessly, demonstrating that an aluminum plate was far more capable of making precise translations. Blade mounts also worked with improved functionality, although bolted joints showed some rotation which was not preferred.

One large issue that came to light during final testing was in the actuator mounts themselves, which were still PLA. Testing with higher loads and fewer barriers to failure revealed that the actuator mounts were not capable of holding the actuators in place with friction alone, and that an additional component to support the actuators from the top was necessary. These mounts also experienced internal damage, with some small cracks appearing after repeated usage.

Overall, the prototype was able to successfully complete numerous cuts of fully cured gel in a row when supported slightly from the top. Additionally, the process for making cuts was timed, revealing an average full cycle run time of approximately 7 minutes which fulfilled the design criteria of at least 50% reduction in labor time.

9.3 Comparison with Analytical Predictions

Test results generally aligned with analytical predictions, especially in comparisons between 3D printed and metal components.

Since friction and cutting forces were used to determine specs for many components in the system, validation would be cyclical. However, through testing the team was able to determine that the selected motors, lead screws, and actuators were able to handle the applied loads without issue.

For the push plate and blade mounts, testing aligned with analysis in showing that aluminum components were much stronger and more reliable that PLA. However, it is interesting to observe that total failure was not predicted in the blade mounts analytically, but still did occur. This can be attributed to higher than expected loads and material property uncertainty in simulations.

Overall, trends between analysis and testing aligned closely including the general feasibility of cutting and moving a block with final components installed.

9.4 Design Improvements from Testing

Initial testing laid the foundation for component selection and additional analysis, with the measured coefficient of friction and density featured in all calculations as shown in previous sections.

Test results in the mockup phase were instrumental in demonstrating what improvements needed to be made in the final version to create a successful prototype. After experiencing failure in the 3D printed blade mounts, the team quickly switched to a welded aluminum solution with much higher load capacity. Additionally, after observing bending in the push plate the team replaced the component with an aluminum version, increasing the accuracy of block motion in the solution.

The testing of the electronics also led the team to improve the Arduino commands to make them more intuitive. The actuator controller failure further showed the need for having replacement electronics and reliable wiring, so spare controllers were ordered and the wiring was de-tangled and managed far better.

10 Recommendations and Conclusions

10.1 Design Improvements

Wire Management. The current electronics assembly relies on loose, unrouted wiring between the Arduino controller, stepper driver boards, actuator relay, and power supply. This creates a risk of accidental disconnection during operation and complicates troubleshooting. Implementing dedicated cable routing along the 80/20 extrusion channels, with bundled and labeled wire runs, would improve reliability and reduce setup time between tests.

Blade Guard. The prototype does not currently include a physical blade guard over the guillotine cutting mechanism. As discussed in Section 8, this guard is required to satisfy the machinery guarding provisions of EM 385-1-1 Section 18.B.03 and ANSI B11.19 before the system can be cleared for operational deployment. The guard must completely cover the blade while allowing for the full actuator stroke to be completed.

Electronics Enclosure. The Arduino controller, stepper driver boards, and actuator relay board are currently exposed. A dedicated electronics enclosure would protect these components from gelatin contamination and moisture during cleaning, and satisfy the live parts guarding requirement of EM 385-1-1 Section 11.A.02.d.

Actuator Vertical Constraint. During cutting, the reaction force transmitted upward through the linear actuators produces a tendency for the actuator bodies to lift vertically and slide against their mounts. In the current design, the actuators are retained primarily by friction at the mount interface, which may be insufficient under repeated high-force cuts. Adding a through-hole and fastener at the top of each actuator body to attach to the horizontal 80/20 cross member would mechanically constrain this vertical movement and improve the consistency and reliability of the cutting stroke after repeated uses.

Metal Actuator Mounts. The actuator mounts are currently fabricated from high-infill PLA via FDM 3D printing. While adequate for initial prototyping, PLA is susceptible to creep and fatigue under sustained cyclic mechanical loading, particularly at the fastener contact points where stress concentrations develop. Remanufacturing the actuator mounts in aluminum or steel would significantly extend service life, reduce the risk of failure, and better withstand the environmental conditions.

10.2 Future Work

To design the additional mounting components described above, additional prototyping would be necessary. Load analysis should be redone to determine the resultant force the actuators produce as the basis of new component design. The new part would likely be mounted at the top of the actuators as not to rely on friction to hold them in place like the current components.

The team currently utilizes a makeshift blade guard that would be improved given additional time. An ideal guard would model ice-skate covers and provide protection all throughout the blade's length.

An electronics enclosure and additional wire management would require a re-wiring of the system, including the installation of proper electronics connectors rather than splices. This system would allow wires to be unplugged and replaced as necessary, saving significant troubleshooting time. An enclosure would further protect the drivers, controllers, and arduino from gel residue or other wear and tear, improving the durability of the fixture.

10.3 Path to Mass Production

To mass produce this fixture, custom component manufacturing and assembly process should be streamlined. Many of the components are already mass produced such as 8020 extruded aluminum, frame hardware, and

mounting components. Since the push plate is a flat piece of aluminum, an automated waterjet or mill process would likely be the fastest and most cost efficient method of mass manufacturing. For more complex components such as the actuator mounts and blade mounts, a CNC program would make the most sense for mid-scale production. Depending on a cost analysis, die casting could be a more affordable alternative once production scales to the extent that additional automation is beneficial.

Additionally, improvements to the assembly process would be critical. While 8020 is an excellent material for modular design, a pre-made frame would optimize the assembly process for users or technicians. Additionally, improved wiring would make installation of electronic components much faster and easier to repair if necessary.

10.4 Lessons Learned

The biggest lesson learned by the team throughout this project was project management and completing tasks on short timelines. The ideation phase of this project took a significant amount of time and discussion with the client, meaning that analysis and prototyping did not begin until later in the second semester. Despite this setback, the team was able to rapidly prototype, test, and improve upon the solution in a short time frame.

The team split up sub-assemblies between members, giving each engineer a specialty to focus on in presentations and reports. Having ownership over certain components for analysis and manufacturing created expertise within the group and gave each member a leadership role within the team. Implementing this setup increased productivity as compared to earlier in the semester.

Communication was absolutely critical especially in the last few weeks of the project. Tasks were distributed by member specialty and workload capacity and all progress was communicated frequently.

Acknowledgments

- U.S. Army DEVCOM Armaments Center and the Test & Evaluation Division
- Duke University ME 424 Instructors

References

1. U.S. Army Corps of Engineers. *Safety and Health Requirements Manual*, EM 385-1-1. Washington, D.C.: U.S. Army Corps of Engineers, 30 November 2014.

2. International Organization for Standardization. *Environmental Management Systems — Guidelines for Incorporating Ecodesign*, ISO 14006:2020. Geneva: ISO, 2020.

A User Manual

This manual provides operating instructions for the automated ballistic gelatin slicing system. It covers system initialization, the cutting sequence, and routine maintenance.

A.1 System Overview

The system consists of four integrated subsystems: an 80/20 aluminum extrusion frame, a stainless steel gelatin tray driven by dual NEMA 17 stepper motors and T8 lead screws, a dual linear actuator guillotine slicing mechanism, and an Arduino-based electronics assembly controlled via USB serial connection from a laptop. The operator interacts with the system entirely through the Arduino IDE Serial Monitor; no manual intervention is required once the cutting sequence is initiated.

A.2 Required Equipment

Before beginning, confirm the following are available and in working condition:

- Assembled slicing machine with all wiring connected
- Laptop with the Arduino IDE installed
- USB cable connecting the laptop to the Arduino
- 24V DC power supply connected to the stepper driver boards and actuator relay
- Ballistic gelatin block to be processed
- Appropriate personal protective equipment (PPE): safety glasses and cut-resistant gloves

A.3 System Initialization and Homing

Step 1: Make wired connections. Connect the 24V DC power supply to the electronics assembly. Confirm that the stepper driver boards and actuator relay board indicator lights are active. Connect the USB cable from the laptop to the Arduino.

Step 2: Launch the Arduino IDE. Open the Arduino IDE on the laptop. Navigate to `Tools` → `Port` and confirm that the correct serial port corresponding to the Arduino is selected.

Step 3: Open the Serial Monitor. Open the Serial Monitor via `Tools` → `Serial Monitor` or by pressing `Ctrl+Shift+M`. Set the baud rate in the Serial Monitor dropdown to `115200`. Confirm that the monitor is receiving output from the Arduino before proceeding.

Step 4: Manually home the pushing plate. Before issuing any software commands, manually slide the pushing plate to the end of the tray furthest from the blade — this is the physical home position. Ensure the plate is fully seated at this end before continuing.

Step 5: Set software zero. In the Serial Monitor, type `zero()` and press Enter. This registers the current pushing plate position as the zero reference for all subsequent move and cut coordinates. Confirm that the Serial Monitor returns an acknowledgment before proceeding.

WARNING: Do not place the gelatin block in the tray until after `zero()` has been confirmed. Keep hands clear of all moving components at all times when the system is powered.

Step 6: Load the gelatin block. Once `zero` is confirmed, place the ballistic gelatin block onto the fixture. Ensure the block is flush against the pushing plate and aligned with the lateral edges of the tray.

TIP: At any point during operation, type `help` in the Serial Monitor and press Enter to display a comprehensive list of all available commands, including test run commands and emergency stop commands.

A.4 Operating the Cutting Sequence

Step 1: Load cut coordinates. In the Serial Monitor, use the `load` command to enter the x-coordinates at which cuts are to be made, measured from the zero position along the axis of block travel. The Serial Monitor will display a confirmation message indicating that the specified coordinates have been successfully loaded. Verify the listed coordinates before proceeding.

Step 2: Execute the cutting sequence. Type `run` in the Serial Monitor and press Enter to begin operation. The system will automatically translate the block to each loaded coordinate in sequence and actuate the guillotine blade to execute each cut. No further operator input is required during the run. Monitor the Serial Monitor for status updates and any error messages.

WARNING: Never place hands or any other objects in the tray area or near the blade path after issuing `run`. The actuator stroke is forceful and will not stop mid-cycle once initiated. Use `help` to identify the emergency stop command before beginning any cutting session.

A.5 Shutdown

After all cuts are complete and fragments have been extracted, type `move 0` in the Serial Monitor and press Enter. This returns the pushing plate to the zero position at the end of the tray furthest from the blade, ready for the next block. Once the plate has returned, power down the 24V supply and disconnect the USB cable from the Arduino.

A.6 Maintenance and Cleaning

After each use. Remove all gelatin residue from the stainless steel tray, lead screws, and blade using a warm water rinse and a soft cloth or sponge. Do not use abrasive cleaning tools on the tray surface or blade edge. Dry all metal surfaces thoroughly after cleaning to prevent surface corrosion on the carbon steel lead screws.

Component lubrication. The T8 lead screws, the blade, and the base plate requires periodic lubrication to maintain smooth and consistent block translation. Apply a light machine oil or dry PTFE lubricant to the screw threads after every 10 to 15 operating sessions, or sooner if increased motor resistance or irregular motion is observed. Wipe away excess lubricant before the next use to prevent gelatin contamination.

PLA actuator mounts. Inspect the 3D-printed PLA actuator mounts periodically for signs of cracking, deformation, or loosening at the fastener points. PLA components under sustained mechanical load can develop fatigue over time. If deformation is observed, replace the affected mount using the original print file before resuming operation. Replacement mounts should be printed at high infill to match the structural performance of the originals.

B Fabrication Instructions

This appendix section provides step-by-step fabrication instructions for the prototype assembly.

B.1 Tools and Hardware Required

- Hex key set (metric and imperial)
- Drop-in T-nuts sized to the 80/20 extrusion slot profile
- Corner brackets compatible with the extrusion profile
- Torque screwdriver or wrench

B.2 Frame and Horizontal Base Assembly

Step 1: Attach motor mounts to horizontal 80/20 bars. Insert drop-in T-slot nuts into the slots of the two parallel horizontal 80/20 extrusion bars at the locations specified in the engineering drawings. Align the motor mounts over the drop-in pins and fasten securely with the appropriate hex fasteners. Confirm that each mount is flush and square to the bar before fully tightening.

Step 2: Install motors onto motor mounts. Seat each NEMA 17 stepper motor onto its corresponding motor mount. Align the motor shaft with the mount opening and fasten the motor to the mount using M3 bolts. Confirm that the motor shaft protrudes sufficiently to accept the motor coupler in a later step.

Step 3: Attach the perpendicular cross bar to form the base U-shape. Install the perpendicular 80/20 bar across the ends of the two parallel horizontal bars to form a U-shaped base.Confirm that all three bars are coplanar and that the assembly is square before fully tightening all fasteners.

Step 4: Attach the lead screws to the pushing plate. Connect each T8 lead screw to the pushing plate using the appropriate threaded fittings. Ensure both screws are attached symmetrically so that the pushing plate remains parallel to the cross bar during translation. Do not fully constrain the free ends of the lead screws at this stage.

Step 5: Connect the lead screws to the motors via couplers. Slide a motor coupler onto the shaft of each NEMA 17 motor and onto the corresponding end of each lead screw. Tighten the coupler set screws to secure both connections. Confirm that each coupler is centered on the joint between the motor shaft and lead screw with no axial gap.

Step 6: Seat bearings in the end mounts. Press a bearing into each end mount , taking care not to damage the bearing race. The bearings should be fully seated and flush with the mount face.

Step 7: Attach end mounts to the 80/20 and connect to the free ends of the lead screws. Insert drop-in T-slot buts into the horizontal bars at the end mount locations specified in the engineering drawings. Align each end mount over its pins and fasten in place. Feed the free end of each lead screw through the corresponding bearing in the end mount. Confirm that each lead screw rotates freely without binding before fully tightening the end mount fasteners.

Step 8: Attach the base plate. Place the stainless steel base plate on the underside of the horizontal frame assembly. Fasten the base plate to the frame using the specified screws at all designated hole locations. Confirm that the plate lies flat and does not introduce any twist or bow into the frame.

B.3 Vertical Members and Slicing Mechanism Assembly

Step 9: Attach the vertical 80/20 members to the base frame. Position the two vertical 80/20 uprights at the blade end of the base frame. Use corner brackets and drop-in T-nut pins to connect each vertical member to the horizontal frame. Confirm that both uprights are square to the base before fully tightening all bracket fasteners.

Step 10: Attach linear actuator mounts to the vertical members. Insert drop-in T-nut pins into the vertical 80/20 uprights at the actuator mount locations specified in the engineering drawings. Align each linear actuator mount over its pins and fasten securely. Confirm that both mounts are at the same height on their respective uprights before tightening.

Step 11: Install linear actuators into their mounts. Seat each linear actuator into its corresponding mount. Fasten the actuators to the mounts, confirming that each actuator shaft is oriented vertically and aligned with the blade path below.

Step 12: Attach the top cross member. Using corner brackets and drop-in T-nut pins, attach a horizontal 80/20 cross member across the tops of the two vertical uprights. This member provides lateral rigidity to the vertical assembly and prevents splaying of the uprights under actuator load. Confirm the cross member is level and all bracket fasteners are fully tightened.

Step 13: Attach blade mounts to the linear actuators. Connect the custom blade mount to the shaft of each linear actuator using the specified fasteners. Confirm that both mounts are at the same height and

that the combined mount assembly is level across the full blade width, ensuring uniform force distribution across the blade during actuation.

Step 14: Mount the blade and blade cover. Attach the guillotine blade to the blade mounts using the specified fasteners. Confirm that the blade is secure, level, and centered over the tray. Once the blade is mounted, attach the blade cover. Confirm that the cover does not interfere with the actuator stroke before proceeding to electronics installation.

C Engineering Drawings

This appendix contains detailed engineering drawings for each custom designed component of the system assembly. Dimensions are given in inches unless otherwise noted. For purchased part specifications, refer to Appendix F.

C.1 Push Plate

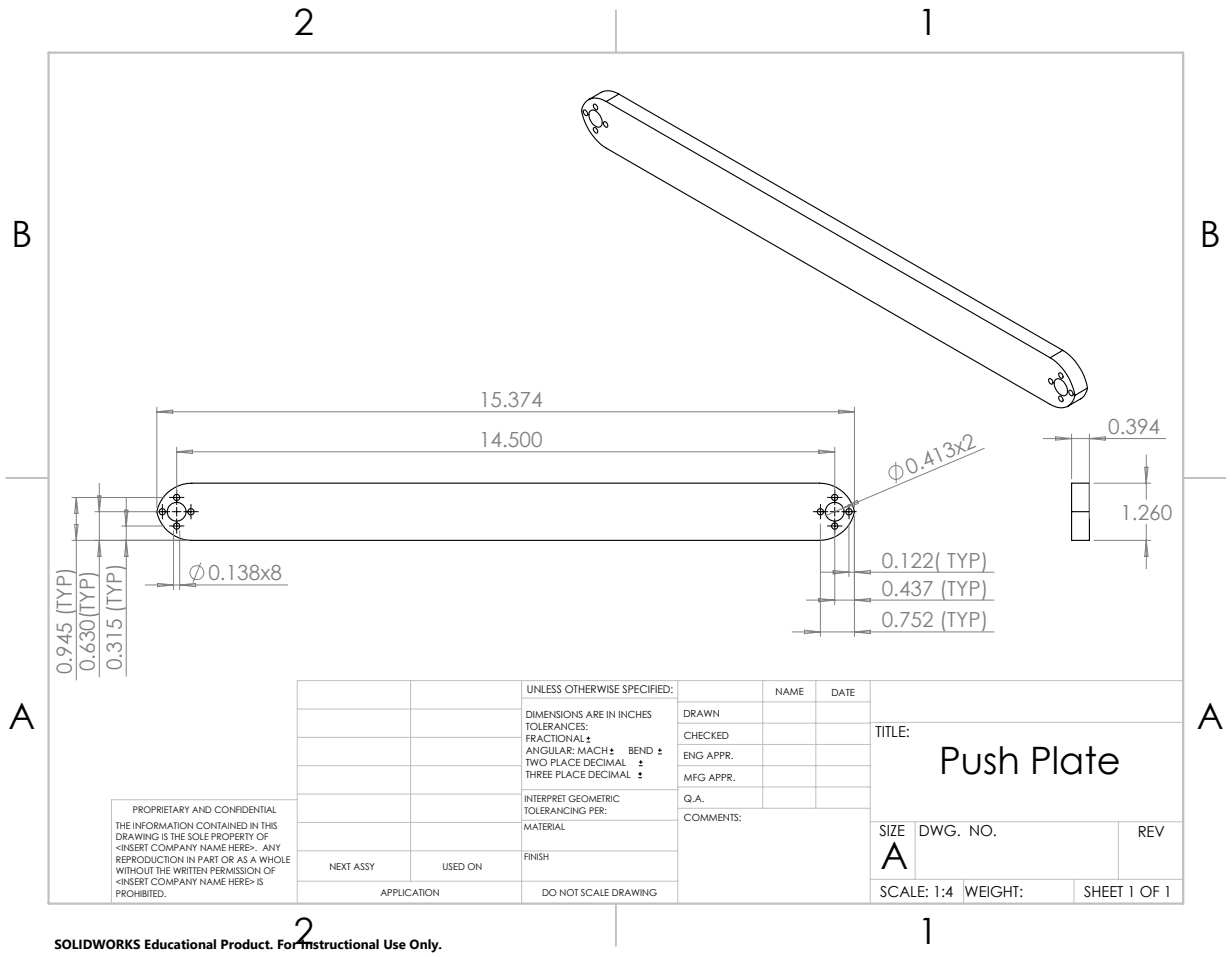


Figure 15: Push Plate Drawing

C.2 Motor Mount

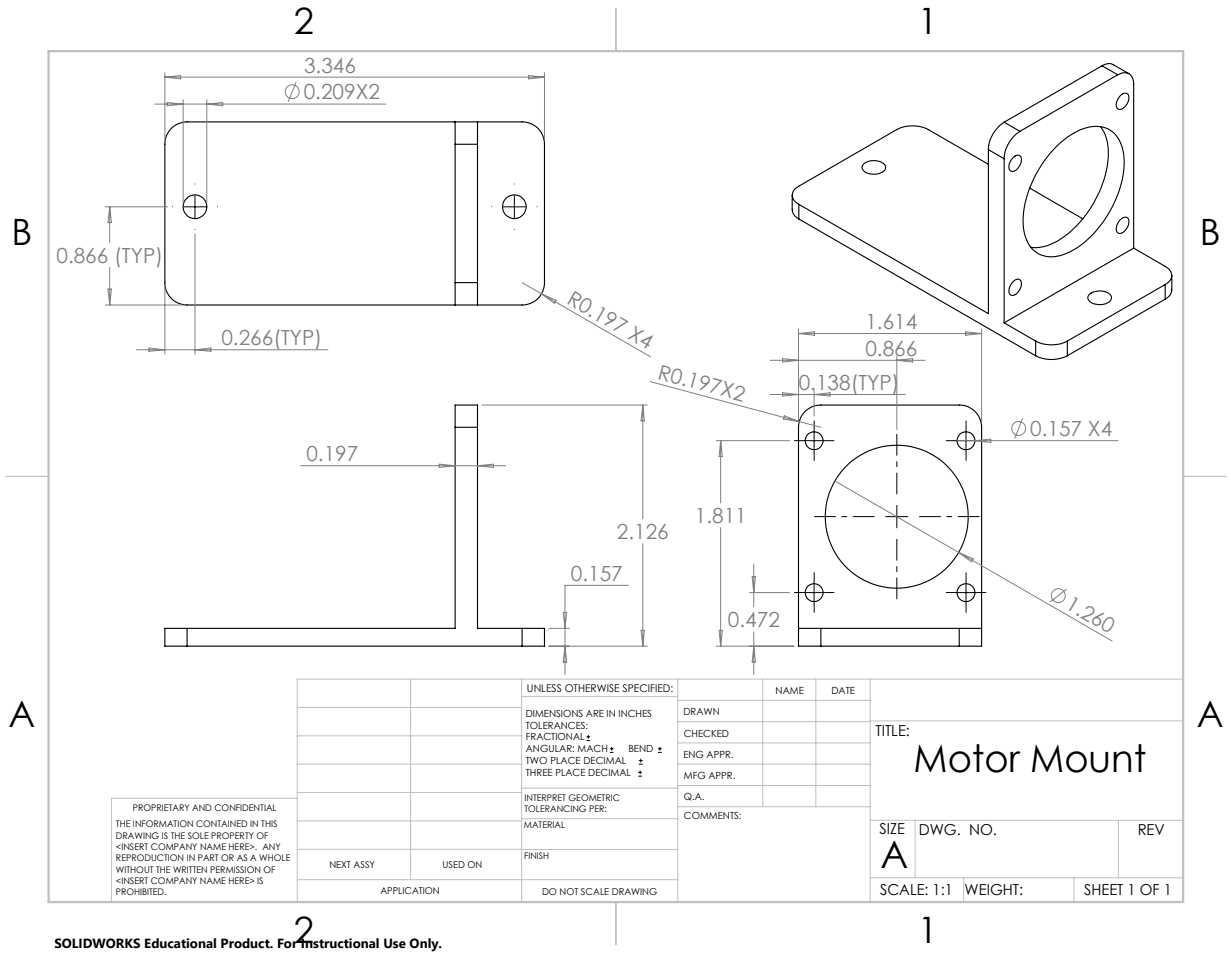


Figure 16: Motor Mount Drawing

C.3 End Mount

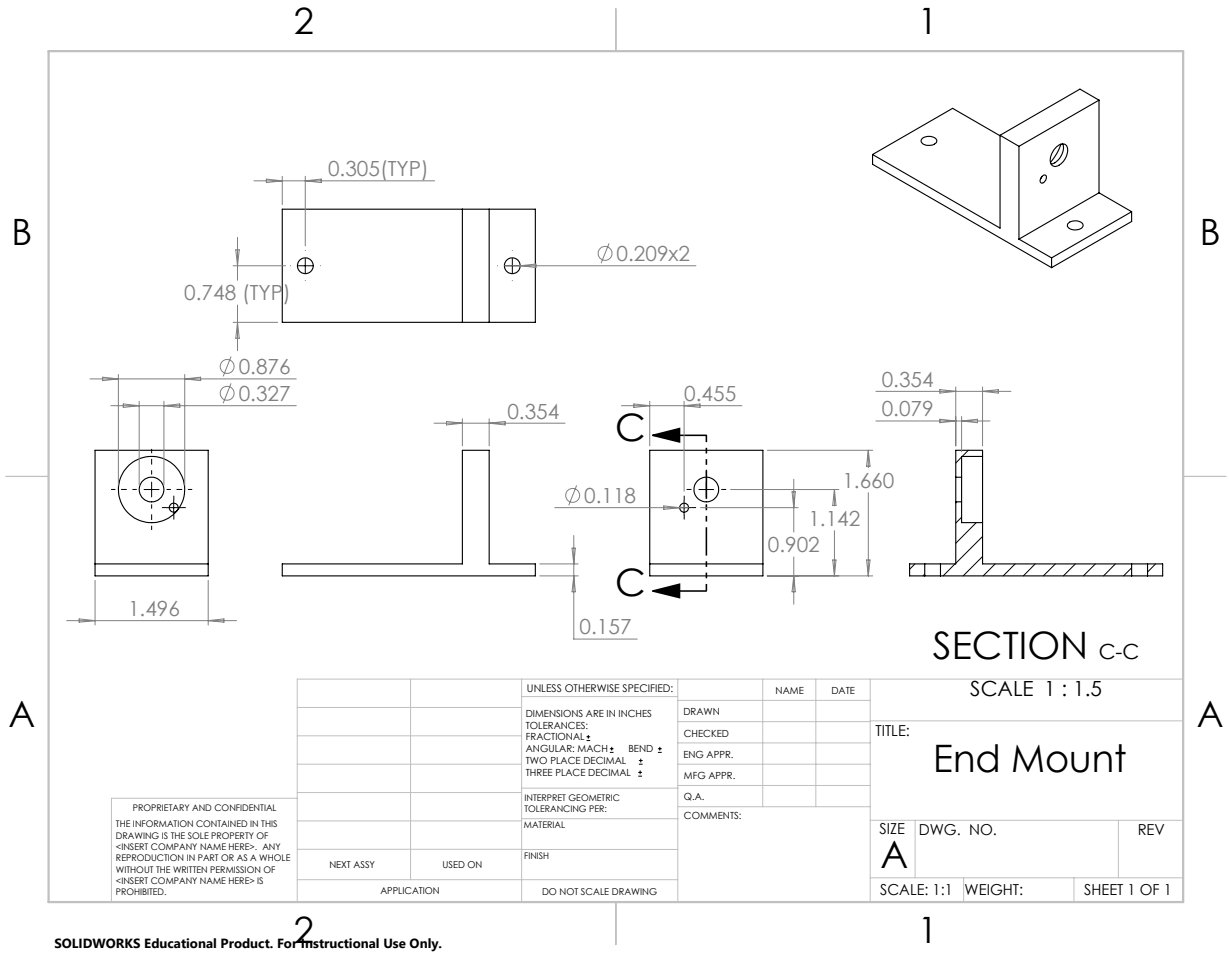


Figure 17: End Mount Drawing

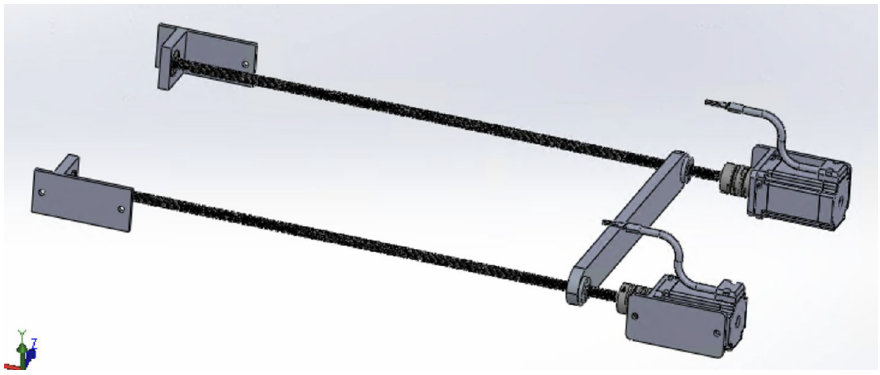


Figure 18: Assembly drawing of block-moving mechanism

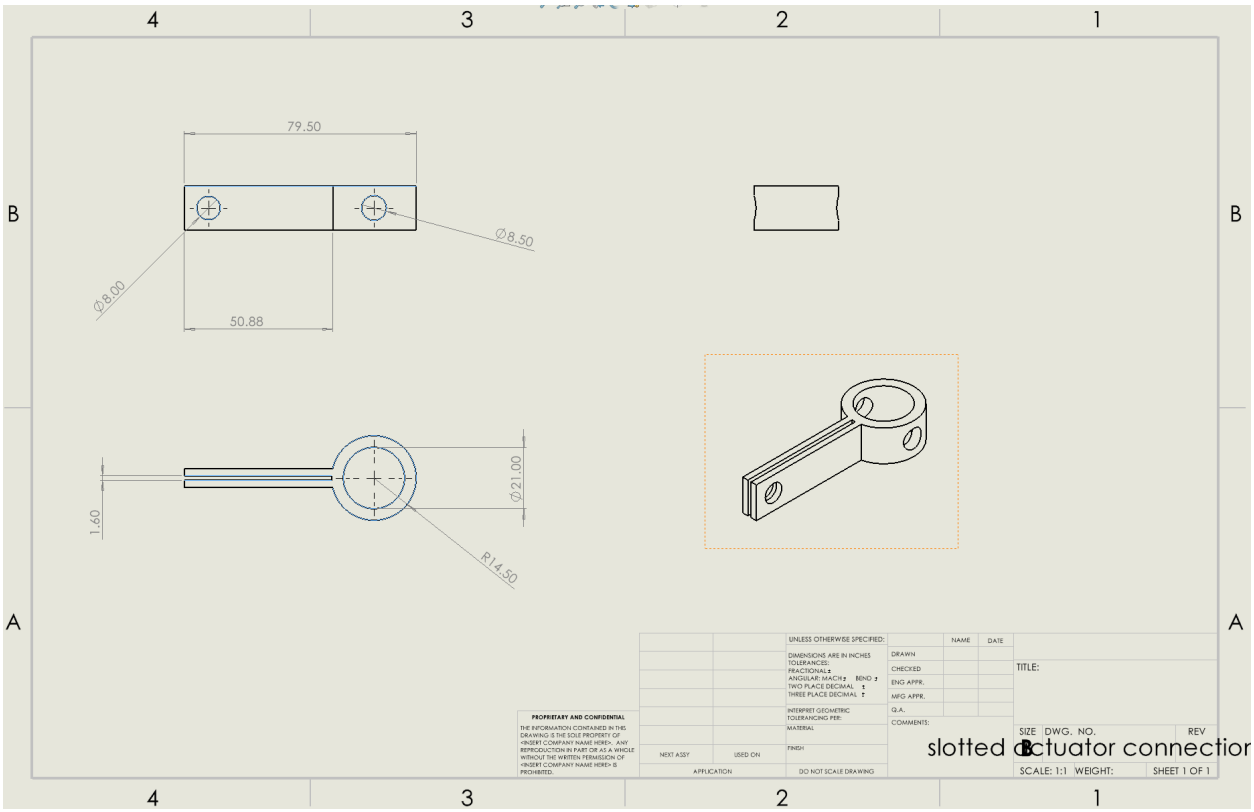


Figure 19: Dimensional Drawing of Blade Mount

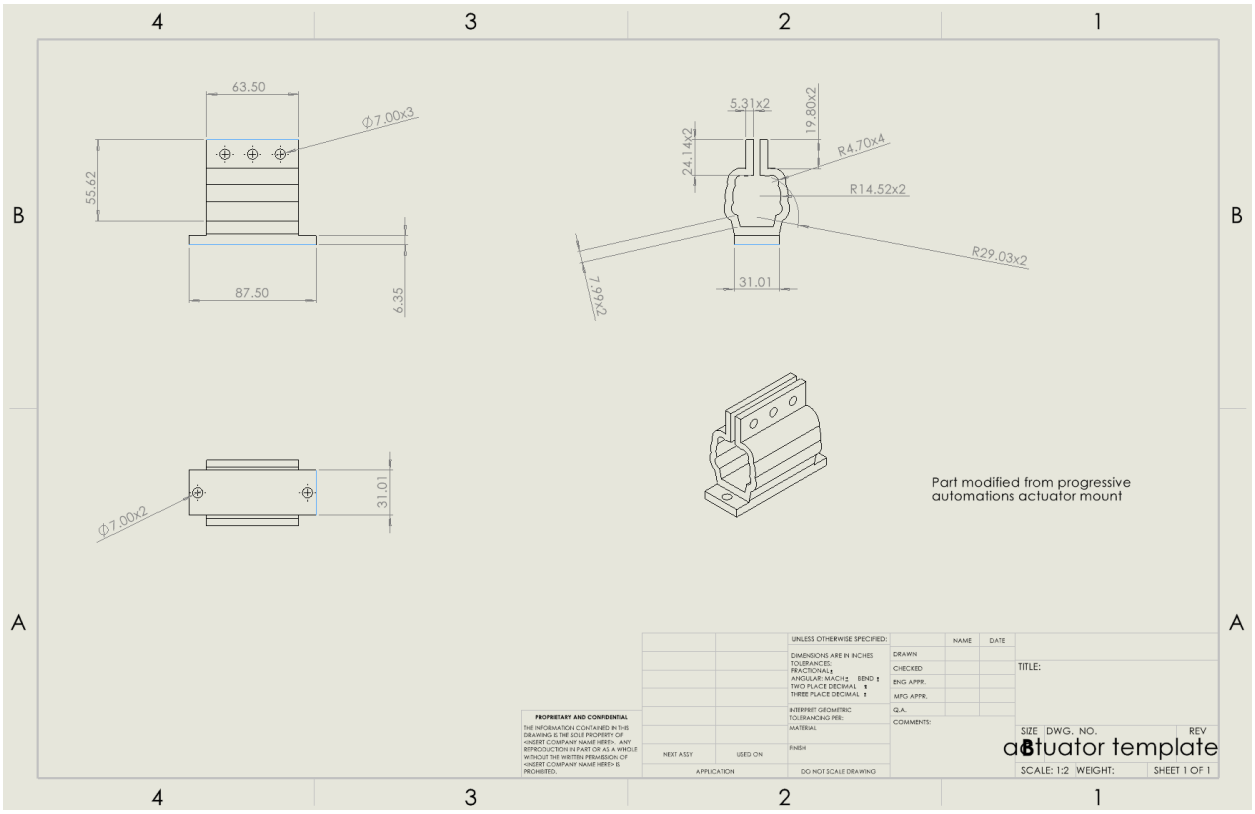


Figure 20: Dimensional Drawing of Actuator Mount

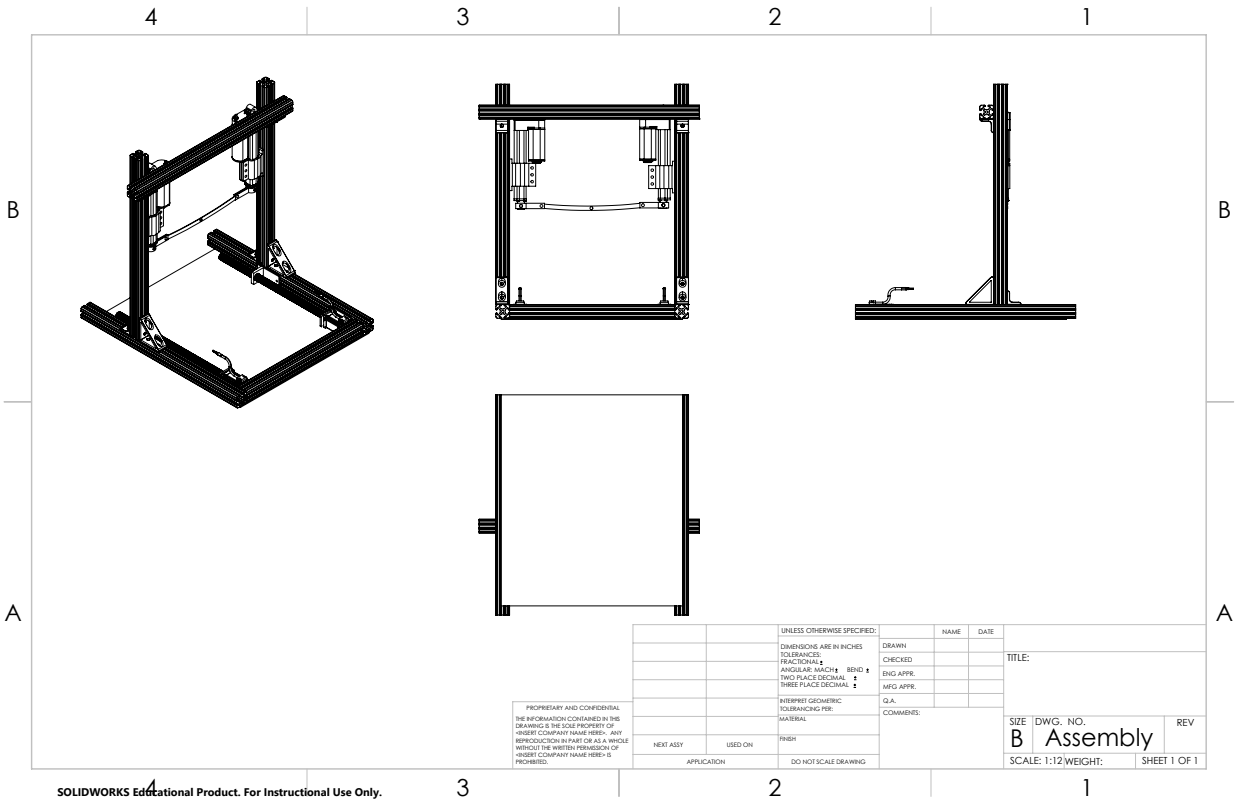


Figure 21: Assembly Dimensional Drawings

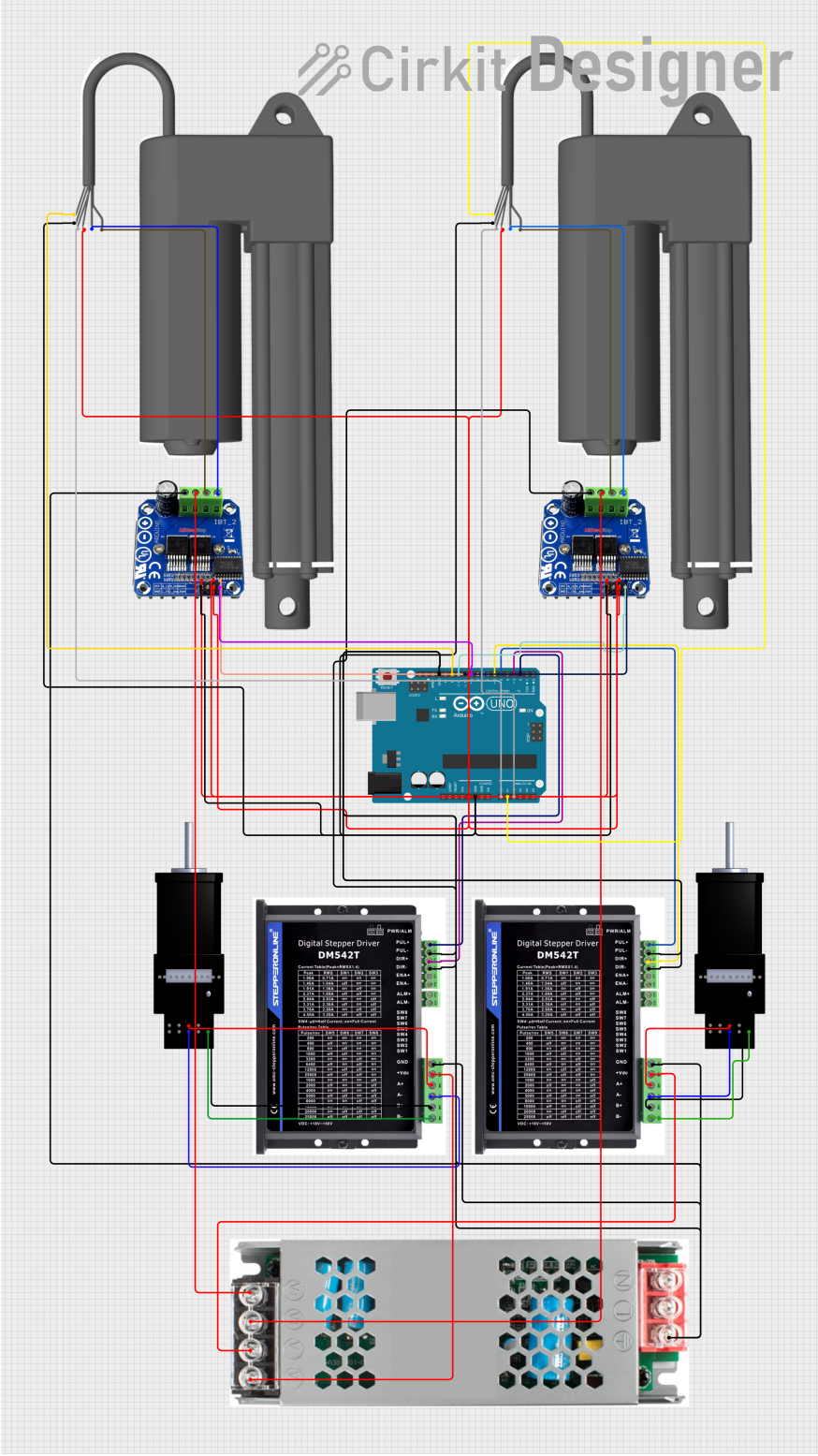


Figure 22: System-level wiring diagram for the slicing apparatus. Pin assignments correspond to the constants defined at the top of `slicer_arm_controller.ino` (Appendix G).

D Bill of Materials

Table 9: Bill of Materials

Item	Description	Qty	Unit Cost ($)	Total Cost ($)	Source
Linear Actuators	Feedback PA-09 Mini Industrial Actuator, 12 in stroke, 24 VDC	2	173.99	347.98	Progressive Automations
Lead Screw Set	Tr8x8 400 mm lead screw with T8 brass nut (2-pack)	1	14.99	14.99	Amazon
H-Bridge Driver	43 A high-power H-bridge DC motor driver	1	20.99	20.99	Amazon
Stepper Motors	NEMA 17 bipolar, 1.8°, 79 N · cm (17HS26-2304S)	2	23.12	46.24	StepperOnline
Flexible Couplings	8 mm–8 mm jaw coupling, 20×25 mm	2	3.26	6.52	StepperOnline
Power Supply	24 V, 20 A, 480 W DC switching power supply	1	31.99	31.99	Amazon
Stepper Drivers	DM542T digital stepper driver, 1.0–4.5 A, 18–50 VDC	2	24.26	48.52	StepperOnline
Arduino Uno R3	Microcontroller running motion-control firmware	1	27.60	27.60	Provided
Wiring & Connectors	22 AWG hookup wire, Dupont/JST connectors, terminal blocks	–	25.00	25.00	Sourced from Duke stock
80/20 Extrusion Frame	Aluminum T-slot extrusion, brackets, and fasteners for chassis	–	100.00	100.00	Sourced from Duke stock
3D-Printed Components	PLA motor mounts, actuator brackets, and guides	–	15.00	15.00	Fabricated
		Total Project Cost		**684.83**	

E Purchased Part Specification Sheets

This appendix contains the manufacturer specification sheets for all major purchased components used in the automated gelatin slicing system. Specifications for each part informed component selection and were used to validate motor torque, actuator force, and power supply capacity in engineering analfysis.

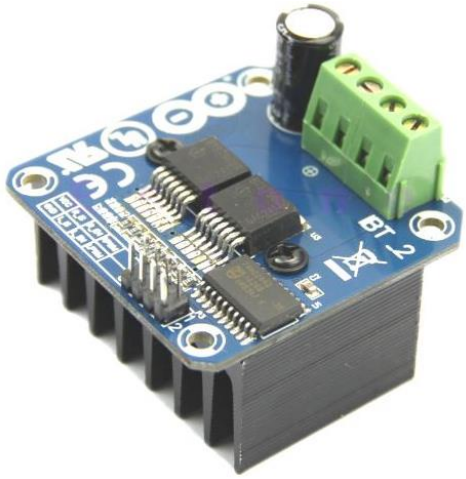
E.1 H-Bridge Motor Driver

Handson Technology

User Guide

BTS7960 High Current 43A H-Bridge Motor Driver

The BTS7960 is a fully integrated high current H bridge module for motor drive applications. Interfacing to a microcontroller is made easy by the integrated driver IC which features logic level inputs, diagnosis with current sense, slew rate adjustment, dead time generation and protection against overtemperature, overvoltage, undervoltage, overcurrent and short circuit. The BTS7960 provides a cost optimized solution for protected high current PWM motor drives with very low board space consumption.



SKU: DRV-1012

Brief Data:

- Input Voltage: 6 ~ 27Vdc.
- Driver: Dual BTS7960 H Bridge Configuration.
- Peak current: 43-Amp.
- PWM capability of up to 25 kHz.
- Control Input Level: 3.3~5V.
- Control Mode: PWM or level
- Working Duty Cycle: 0 ~100%.
- Over-voltage Lock Out.
- Under-voltage Shut Down.
- Board Size (LxWxH): 50mm x 50mm x 43mm.
- Weight: ~66g.

Figure 23: HandsOn Technology specification sheet — H-Bridge Motor Driver

E.2 PA-09 Mini Industrial Linear Actuator

Specifications

Load (LBS)		No Load Current (A)				Full Load Current (A)				Speed (inch/sec)	
Dynamic	Static	12VDC	24VDC	36VDC	48VDC	12VDC	24VDC	36VDC	48VDC	No Load	Full Load
225	225	1.5	0.6	0.5	0.3	4.0	3.0	2.0	1.5	0.59	0.43
330	330	1.5	0.6	0.4	0.3	4.0	3.0	2.0	1.5	0.39	0.27
450	450	1.2	0.5	0.4	0.3	4.0	3.5	2.5	1.5	0.24	0.20

Input Voltage	12VDC
Stroke	1" to 40"
Feedback	Hall Effect, Limit Switch Feedback, and Potentiometer (See Page 6)
Duty Cycle	25% (5 minutes on, 15 minutes off)
Weather Protection	IP66
Overload Protection	NA
Operational Temperature	-25ºC to 65ºC (-13ºF to 149ºF)
Operating Noise	<45 dBA
Limit Switch	Built-In (Non-Adjustable)
Cable Length	40"
Connector	Molex Mini-Fit Jr 2-Pin
Front Mounting Hole Size	0.32"
Rear Mounting Hole Size	0.32"
Actuator Type	Mini, Industrial
Motor Type	Brushed DC Motor
Screw Type	ACME
Stroke Rod Material	Stainless Steel
Housing Material	Aluminum Alloy 6062
Gear Material	Polyformaldehyde
Compatible Mounting Brackets	BRK-09, BRK-10
Warranty	18 Months

Figure 24: Progressive Automations specification sheet — PA-09 Mini Industrial Linear Actuator

E.3 DM542T Digital Stepper Driver

1. Introductions, Features and Applications

Introductions

The DM542T is a fully digital stepper drive developed with advanced DSP control algorithm based on the latest motion control technology. It has achieved a unique level of system smoothness, providing optimal torque and nulls mid-range instability. Its motor auto-identification and parameter auto-configuration feature offers quick setup to optimal modes with different motors. Compared with traditional analog drives, DM542T can drive a stepper motor at much lower noise, lower heating, and smoother movement. Its unique features make DM542T an ideal choice for high requirement applications.

Features

- Anti-Resonance provides optimal torque and nulls mid-range instability
- Motor auto-identification and parameter auto-configuration when power on, offer optimal responses with different motors
- Multi-Stepping allows a low resolution step input to produce a higher microstep output, thus offers smoother motor movement
- 15 selectable microstep resolutions including 400, 800, 1600, 3200, 6400, 12800, 25600, 1000, 2000, 4000, 5000, 8000, 10000, 20000, 25000
- Soft-start with no “jump” when powered on
- Input voltage 20-50VDC
- 8 selectable peak current including 1.00A, 1.46A, 1.91A, 2.37A, 2.84A, 3.31A, 3.76A, 4.20A
- Pulse input frequency up to 200 KHz, TTL compatible and optically isolated input
- Automatic idle-current reduction
- Suitable for 2-phase and 4-phase motors
- Support PUL/DIR mode
- Over-voltage and over-current protections

Applications

Suitable for a wide range of stepper motors, size from NEMA17 to 24. It can be used in various kinds of machines, such as X-Y tables, engraving machines, labeling machines, laser cutters, pick-place devices, and so on. Particularly adapt to the applications with low noise, low heating, high speed and high precision.

2. Specifications

Electrical Specifications (T_j = 25℃/77℉)

Parameters	DM542T			
	Min	Typical	Max	Unit
Output Peak Current	1.0	-	4.2 (3.0 RMS)	A
Input Voltage Logic	+20	+36	+50	VDC
Signal Current Pulse	7	10	16	mA
input frequency Pulse	0	-	200	kHz
Width	2.5	-	-	uS
Isolation resistance	500			MΩ

Figure 25: Stepper Online specification sheet — DM542T Digital Stepper Driver

E.4 NEMA 17 Stepper Motor

Product Type	
Product Type	Hybrid Stepper Motor
Electrical Specification	
Bipolar/Unipolar	Bipolar
Has Torque/Thrust Curve?	Yes
Holding Torque(Ncm)	79
Holding Torque(oz.in)	111.896
Inductance(mH)	2.8
Phase Resistance(ohms)	1.45
Rated Current(A)	2.3
Step Angle(deg.)	1.8
Physical Specification	
Body Length(mm)	67
Frame Size(mm)	Nema 17 (42 x 42)
Inventory Method	Dedicated Inventory
IP Rating	40
Lead Length(mm)	300
No. of Lead	4
Output Shaft Type	D
Product Series	S
Shaft Diameter(mm)	8
Shaft Length(mm)	21
Single Shaft/Dual Shaft	Single Shaft
Weight(g)	550

Figure 26: Stepper Online specification sheet — NEMA 17 Stepper Motor

E.5 24V 20A DC Switching Power Supply

Specifications

Parameter	Value
Input Voltage	AC 110V/220V (85-145V / 185-265V)
Output Voltage	DC 24V (Adjustable range ±10%, 21.6V-26.4V)
Current Range	0 to 20A
Rated Power	480W (max)
Efficiency	≥80%
Frequency	50-60Hz
Operating Temperature	-10°C to 60°C (14°F to 140°F)
Operating Humidity	20-90%RH (non-condensing)
Storage Temperature	-20°C to 85°C
Storage Humidity	10%-95%RH
Dimensions (L*B*H)	215mm * 115mm * 50mm
Net Weight	0.75 kg
Certifications	RoHS, CE, LVD, EMC

Figure 27: BOYSTRO specification sheet — 24V 20A DC Switching Power Supply

E.6 Arduino UNO R3

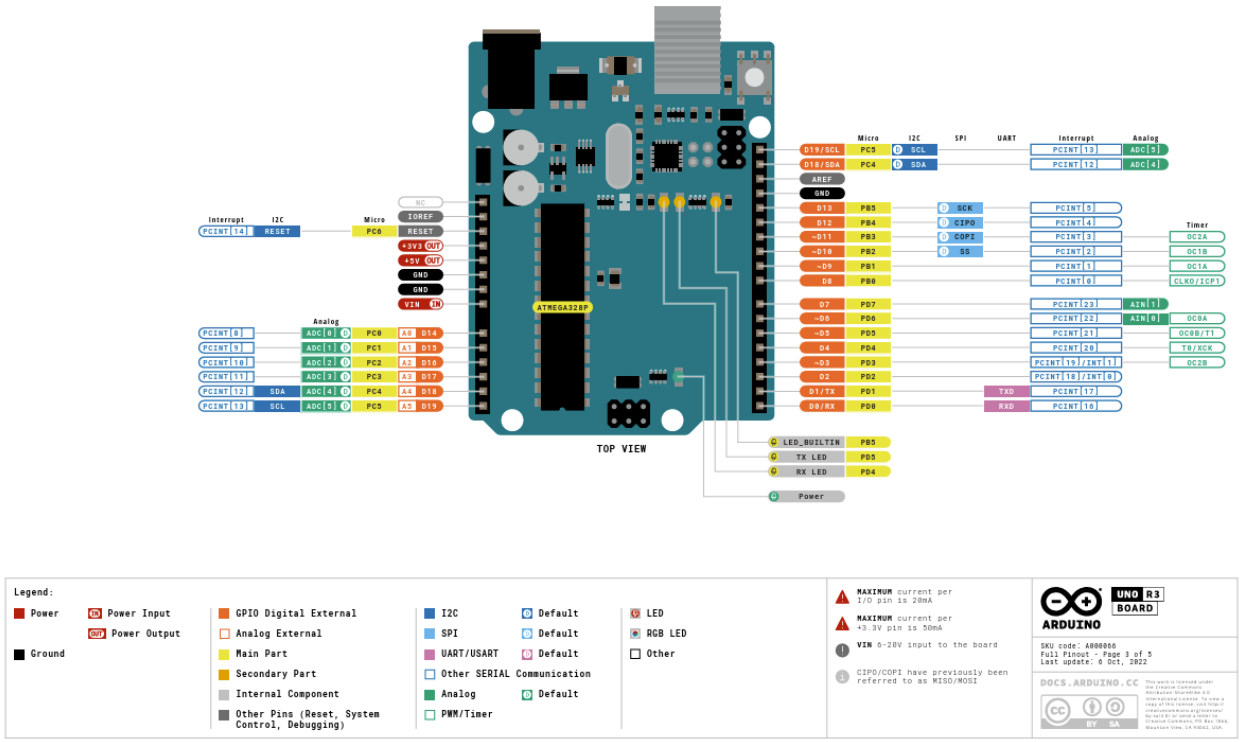


Figure 28: Arduino UNO R3 Full Pinout

F Detailed Analysis

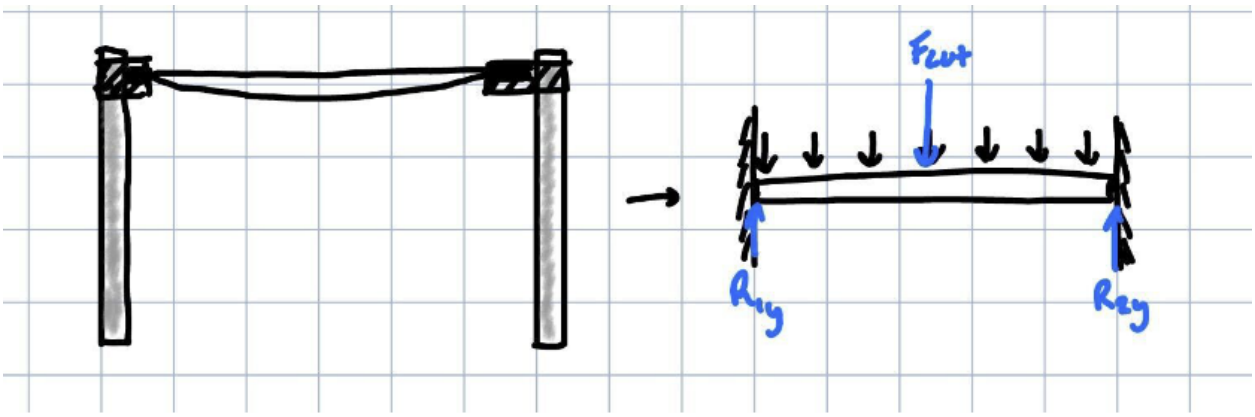


Figure 29

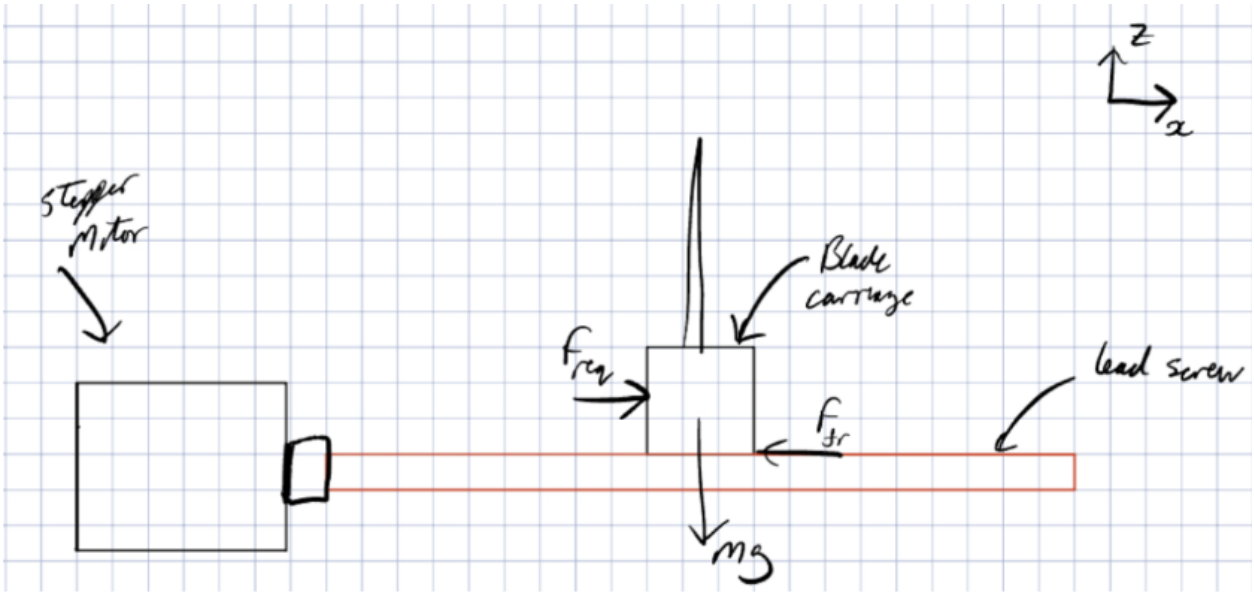


Figure 30

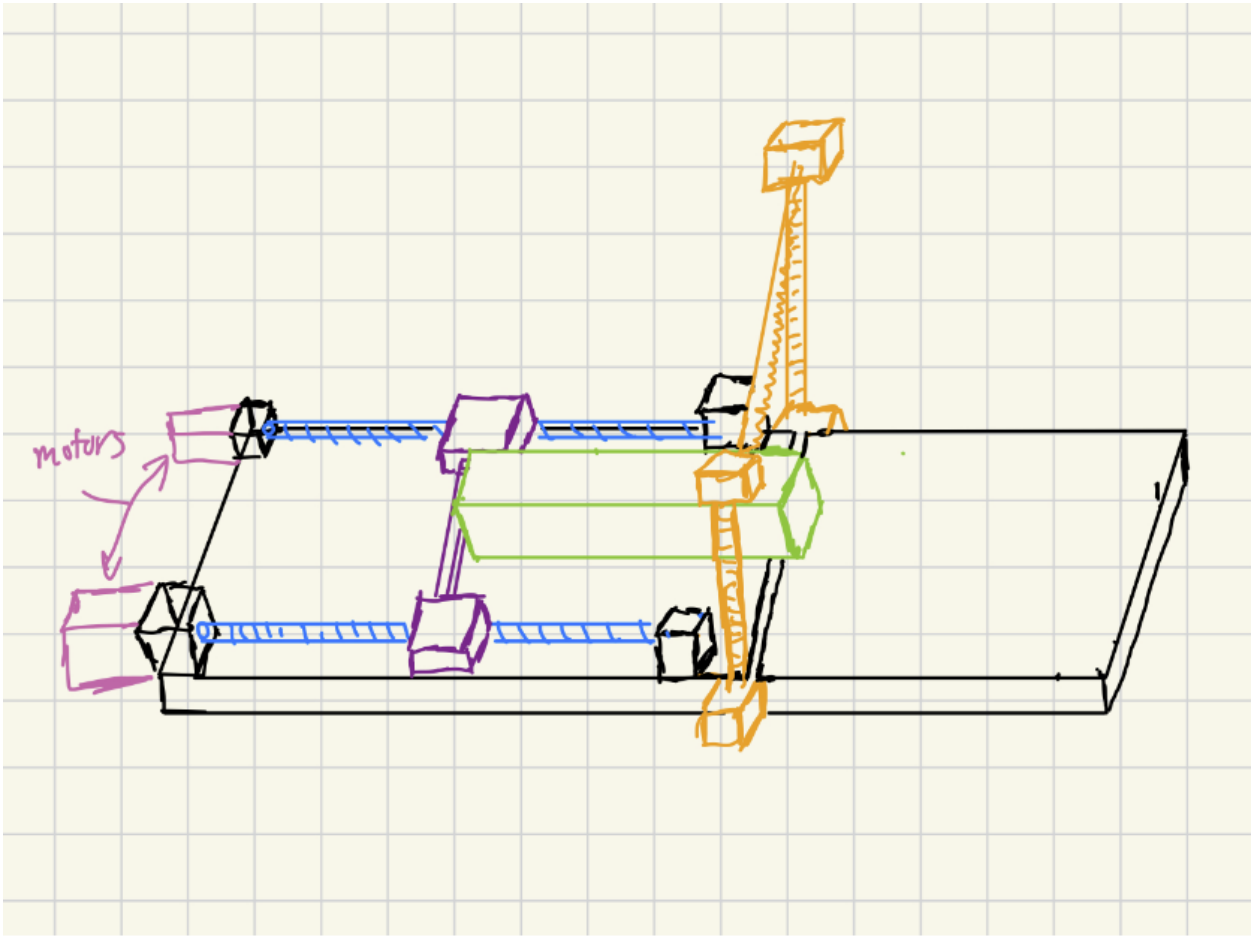


Figure 31

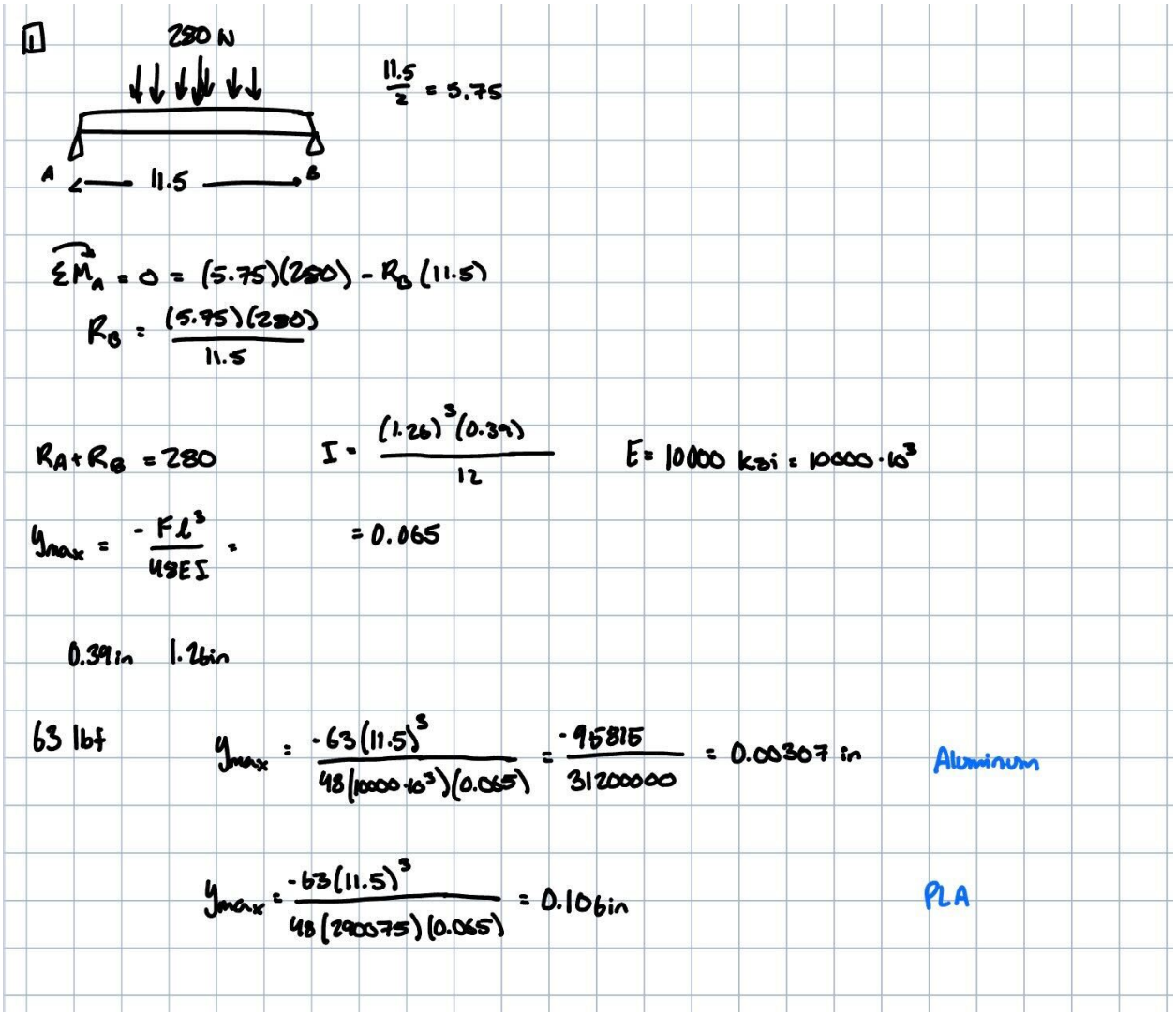


Figure 32

G Design Documentation

Meeting Documentation

Client Meetings

- November 13: first meeting and introduction
- November 20: reviewed previous work on project and client ideation
- February 10: discussed design updates, client shipped gel materials
- February 17: preliminary design finalization and testing plan
- February 24: pre-testing meeting
- April 6: early prototyping update, client recommendations for support structure
- April 13: prototype update via pictures, advice for final prototype, help with debugging actuator issues

- April 20: final client demonstration

G.0.1 Idea Descriptions

1. **Iterative Reference-Guided Slicing System**
   - A guided blade cuts through the gelatin block one layer at a time, allowing sequential inspection of each slice.
   - After each cut, fragment locations are identified and recorded before proceeding to the next layer.
   - A fixed reference frame keeps the blade aligned with the block for consistent cuts across multiple tests.
   - A touchscreen guides each step and controls blade positioning, while non-stick coating minimizes cleanup.
2. **Real-Time Metal Detection Slicing System**
   - Eddy current sensors detect metal fragments inside the gelatin block while the blade cuts simultaneously.
   - A guided blade dissects the block to expose and recover fragments for weighing.
   - Simultaneous detection and cutting eliminate separate scanning steps, speeding up processing.
   - Simple start/stop buttons and LED indicators make operation straightforward, while a streamlined design minimizes cleanup.
3. **Thermal Imaging Slicing System**
   - Infrared/thermal imaging detects fragments inside the gelatin by identifying temperature differences before cutting.
   - An oscillating blade slices with less force, improving precision and reducing gelatin deformation.
   - The system displays the planned cut path before execution for placement verification.
   - A fixed reference frame and smooth surfaces ensure consistent positioning and easy cleanup.
4. **X-Ray Guided Slicing System**
   - Orthogonal X-rays locate fragments inside the gelatin block and provide precise cutting coordinates.
   - A motorized blade cuts along a single guided axis, allowing one operator to expose and recover fragments.
   - Physical start/stop buttons, LED indicators, and on-device instructions ensure safe, straightforward operation.
   - A pre-aligned reference frame and removable cutting head simplify setup and cleaning; non-stick and corrosion-resistant materials reduce maintenance.
5. **Basic Blade Slicing System**
   - A physical reference helps estimate fragment x-location and select cut positions.
   - A guided blade makes straight, controlled cuts through the gelatin.
   - Preset prompts minimize operator decision-making during operation.
   - Non-stick coated surfaces reduce gelatin adhesion for faster cleanup.
6. **Basic Wire Slicing System**
   - A physical reference helps estimate fragment x-location and choose where to cut.
   - A tensioned wire saw makes straight, controlled cuts with less force and improved safety compared to blades.

- Preset modes minimize operator input and streamline the cutting process.
- Corrosion-resistant materials throughout ensure durability and reduce long-term maintenance.

7. **Real-Time Feedback Slicing System**
   - The gelatin block is sliced incrementally with fragment detection after each layer using real-time feedback.
   - Slice depth and progression are dynamically controlled to improve efficiency.
   - Visual progress indicators keep the operator informed without interrupting operation.
   - A quick-release catch basin continuously removes sliced material, enabling rapid scan–slice cycles with minimal manual handling.
8. **Integrated Metal Detector Waterjet System**
   - A metal detector coil integrated into the cutting head locates fragments in real time during slicing.
   - Water-jet assisted cutting reduces mechanical force on the gelatin.
   - Planned cut paths are visually confirmed before execution.
   - Combining detection and cutting increases automation and throughput, with added system complexity.
9. **Dual-Band CT Scanning System**
   - A dual-band CT scan images the entire gelatin block without cutting, identifying and segmenting all embedded fragments.
   - Each fragment is labeled as a 3D object with location, orientation, and volume.
   - Dual-band energy calibration estimates material density, which combines with volume to calculate fragment mass.
   - The system generates a complete per-fragment data table for automated analysis and documentation.
10. **Automated Slicing & Robotic Extraction System**
    - An automated slicer sections the gelatin block into thin layers.
    - A robotic arm probes each layer to locate fragments in 3D and extracts them using a small gripper.
    - Extracted fragments are placed on an integrated scale for automatic mass measurement.
    - Fragment location and weight are recorded automatically, creating a repeatable, low-labor workflow.

Option	Precision of Data Collection	Reduce Labor Time	Ease of Operation	Safety	Feasibility	Total
Weight	0.6	0.8	0.4	0.7	0.5	
Current	0	0	0	0	0	**0**
Alt #1	0	1	2	1	-1	**1.8**
Alt #2	-1	-1	0	1	-2	**-1.7**
Alt #3	-3	1	2	1	-3	**-1.0**
Alt #4	3	2	1	0	-2	**2.8**
Alt #5	0	1	2	1	0	**2.3**
Alt #6	0	1	2	0	-1	**1.1**
Alt #7	1	2	1	0	-2	**1.6**
Alt #8	1	0	2	3	-3	**2.0**
Alt #9	3	3	-2	0	-3	**1.9**
Alt #10	3	3	-2	1	-3	**2.6**

Figure 33: Pugh Matrix

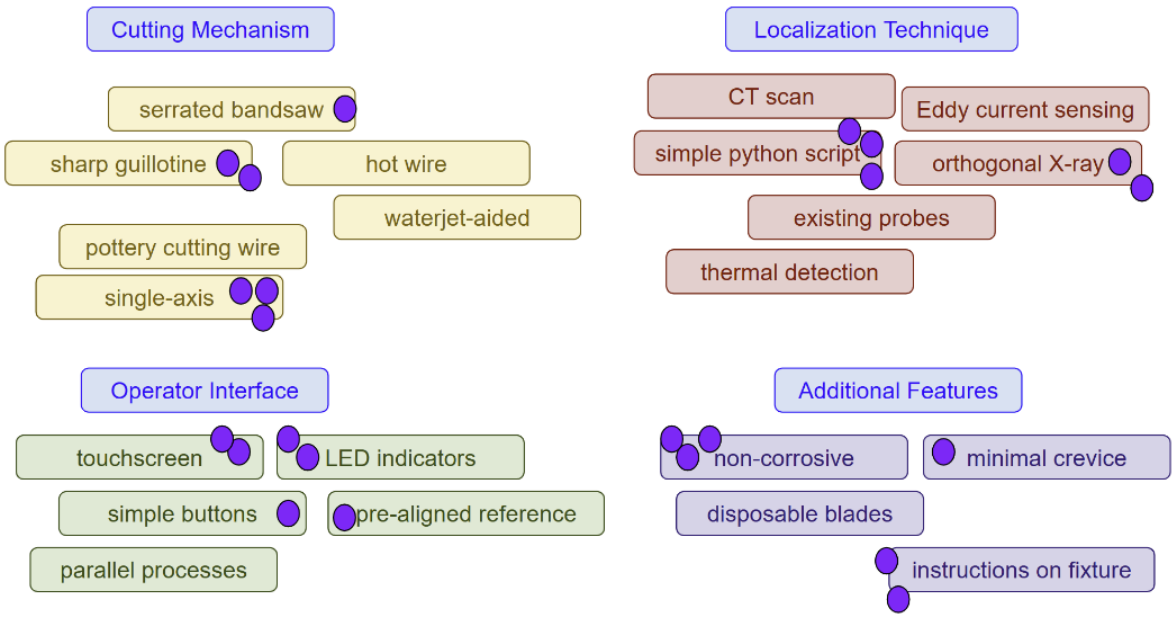


Figure 34: Dot Voting

		lbm Required To Cut Gel Block			
		Paper Cutter (Guillotine)	Serated Knife (Jagged Side	Serated Knife (Wavy Side	Electric Cutter
Test Trials	Trial 1	21	15	16	5
	Trial 2	17	14	13	4
	Trial 3	17	13	13	5
	Trial 4	19	13	12	6
	Trial 5	15	14	12	6
	Average	17.8	13.8	13.2	5.2
	Notes				- Definitely will get hot after repeated usage
	Additional Notes	- There seems to be a force spike on both the initial penetration and the exit positions of the cut with a more consistent force during the cut - fairly certain theres a relationship between the legnth of the cut an the force required - unless this gel wasnt cured correctly this definitely did not feel like it was comparable to cutting through a tire- think the cutter shouldnt be too bad			

Figure 35

t (s)	Tilt up/down (deg)	Tilt up/down (deg)	Tilt up/down (deg)					
-2.75E-04	3.49E+00	-4.37E-01	-2.38E-01		Trial 1	Trial 2	Trial 3	
5.00E-01	-2.95E-01	-2.29E-01	-2.91E-01	Max Angle of Tilt Before Slip (Degrees)	6.34E+01	6.63E+01	6.83E+01	
9.99E-01	-2.72E-01	-4.34E-01	-1.52E-01	Average Maximum Tilt Before Slip	6.60E+01			
1.50E+00	-2.41E-01	-3.41E-01	-2.02E-01	Estimation For The Static Coefficient Of Friction	2.243			
2.00E+00	-2.23E-01	-4.54E-01	-2.39E-01	Notes	This is rather high and, should we emply this			
2.50E+00	-1.46E-01	-2.95E-01	-1.99E-01	Test Gel Block Mass (lbm)	2			
3.00E+00	-2.76E-01	-1.86E-01	-2.65E-01	Test Gel Block Volume (in^3)	54.984375			
3.50E+00	-1.78E-01	-2.45E-01	-2.23E-01	Gel Block Density (lbm/in^3)	0.036			
4.00E+00	-2.18E-01	-9.21E-02	-2.15E-01					
4.50E+00	4.37E-01	-2.73E-01	-2.63E+00	Assumptions: - block is 8x8x12				
5.00E+00	3.33E+00	-1.59E+00	-8.90E-01					
5.50E+00	-1.88E-01	6.61E-01	-4.04E-01					
6.00E+00	-2.18E-01	-4.05E-01	-2.43E-01					
6.50E+00	-2.16E-01	1.19E-01	-2.45E-01	**OPERATIONAL CALCULATIONS**				
7.00E+00	-2.34E-01	7.23E+00	-1.97E-01	Linear Force Required To Move Block (lbf)	62.664		Volume (in^3)	
7.50E+00	-2.29E-01	8.44E+00	-1.94E-01	Force Per Lead Screw	31.332		Weight (lb)	
8.00E+00	-8.30E-01	1.15E+01	-2.32E-01	Torque (lbf*in)	0.491		Lead Screw Lead	
8.50E+00	-1.96E-01	1.37E+01	-2.47E-01	Mechanical Power (Watts)	3.540		Lead Screw Efficiency	
9.00E+00	-2.54E-01	1.38E+01	-3.01E-01				Block speed (Linear) in/s	
9.50E+00	-1.97E-01	1.70E+01	-2.13E-01				Actuator speed (rad/sec)	
1.00E+01	-1.51E-01	1.99E+01	-1.20E-01	**Motor Selection**				
				Accounting for:				

Figure 36

H Final Budget

All in, the team used $517.23 of the $2,000 budget. Significant savings came from sourcing materials from Duke stock.

I Software Code

```
#include <AccelStepper.h>

#include <ctype.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

/*
  Slicer Arm Controller

  Operator workflow:
    1. Manually place the carriage at the physical reference mark.
    2. Send ZERO to define that location as X=0.
    3. Load cut coordinates with ADD or LOAD, then send RUN.

  The travel axis uses two synchronized stepper motors. The cutter uses two
  PA-09-HALL actuators driven by IBT-2 boards, with hall feedback used to keep
  both actuator positions aligned during the cut cycle.
*/

// ---------------------------
// Config constants
// ---------------------------
// Travel-axis stepper driver pins.
constexpr uint8_t TRAVEL_LEFT_STEP_PIN = 2;
constexpr uint8_t TRAVEL_LEFT_DIR_PIN = 3;
constexpr uint8_t ENABLE_PIN = 4;       // Shared active-LOW enable for travel
    drivers if used
constexpr uint8_t TRAVEL_RIGHT_STEP_PIN = 5;
```

```
29 constexpr uint8_t TRAVEL_RIGHT_DIR_PIN = 6;
30
31 // IBT-2 motor-driver inputs. Wire each board's R_EN and L_EN to +5V.
32 constexpr uint8_t CUTTER_LEFT_EXTEND_PIN = 8;
33 constexpr uint8_t CUTTER_LEFT_RETRACT_PIN = 9;
34 constexpr uint8_t CUTTER_RIGHT_EXTEND_PIN = 10;
35 constexpr uint8_t CUTTER_RIGHT_RETRACT_PIN = 11;
36 // PA-09-HALL wire colors: Yellow = Signal 2 (leads while extending), White =
       Signal 1 (leads while retracting).
37 constexpr uint8_t CUTTER_LEFT_EXTEND_FEEDBACK_PIN = 12;
38 constexpr uint8_t CUTTER_LEFT_RETRACT_FEEDBACK_PIN = A0;
39 constexpr uint8_t CUTTER_RIGHT_EXTEND_FEEDBACK_PIN = A1;
40 constexpr uint8_t CUTTER_RIGHT_RETRACT_FEEDBACK_PIN = A2;
41
42 constexpr uint8_t STATUS_LED_PIN = 13;
43
44 constexpr bool TRAVEL_LEFT_DIR_INVERTED = true;
45 constexpr bool TRAVEL_RIGHT_DIR_INVERTED = true;
46
47 // TR8x8 lead screws: 8 mm travel per revolution. DM542T drivers are set to 3200
       pulses/rev.
48 constexpr float TRAVEL_LEAD_MM_PER_REV = 8.0f;
49 constexpr float MM_PER_INCH = 25.4f;
50 constexpr float TRAVEL_PULSES_PER_REV = 3200.0f;
51 constexpr float MAX_TRAVEL_IN = 14.0f;
52 constexpr float TRAVEL_STEPS_PER_INCH = TRAVEL_PULSES_PER_REV * (MM_PER_INCH /
       TRAVEL_LEAD_MM_PER_REV);
53 constexpr float TRAVEL_MAX_SPEED_IN_S = 1.0f;
54 constexpr float TRAVEL_ACCEL_IN_S2 = 2.0f;
55 constexpr uint8_t MAX_CUTS = 50;
56 constexpr long TRAVEL_TEST_ROTATION_STEPS =
       static_cast<long>(TRAVEL_PULSES_PER_REV);
57
58 // PA-09-HALL, 330 lb model: 660 pulses/inch per signal.
59 constexpr float CUTTER_STROKE_IN = 11.4f;
60 constexpr float CUTTER_FEEDBACK_PULSES_PER_IN = 660.0f;
61 constexpr long CUTTER_FULL_STROKE_PULSES =
62     static_cast<long>((CUTTER_STROKE_IN * CUTTER_FEEDBACK_PULSES_PER_IN) + 0.5f);
63 constexpr long CUTTER_SYNC_TOLERANCE_PULSES = 12L;
64 constexpr uint16_t CUTTER_EXTEND_SETTLE_MS = 150;
65 constexpr uint16_t CUTTER_RETRACT_SETTLE_MS = 150;
66 constexpr uint16_t CUTTER_NO_PULSE_STOP_MS = 250;
67 constexpr uint16_t CUTTER_RETRACT_GRACE_MS = 750;
68 constexpr unsigned long CUTTER_MOVE_TIMEOUT_MS = 90000UL;
69 constexpr unsigned long CUTTER_RETRACT_TIMEOUT_MS = 90000UL;
70
71 constexpr unsigned long SERIAL_BAUD = 115200UL;
72 constexpr size_t SERIAL_LINE_MAX = 96;
73 constexpr unsigned long ABORT_DRAIN_TIMEOUT_MS = 5000UL;
74 constexpr uint8_t STEPPER_MIN_PULSE_WIDTH_US = 2;
75
76 // ---------------------------
77 // Globals/types
78 // ---------------------------
79 enum MachineState { UNZEROED, IDLE, MOVING, CUTTING, ABORTED };
80
81 AccelStepper travelLeftStepper(AccelStepper::DRIVER, TRAVEL_LEFT_STEP_PIN,
       TRAVEL_LEFT_DIR_PIN);
```

```
AccelStepper travelRightStepper(AccelStepper::DRIVER, TRAVEL_RIGHT_STEP_PIN,
    TRAVEL_RIGHT_DIR_PIN);

float cutQueue[MAX_CUTS];
uint8_t queueCount = 0;
volatile bool stopRequested = false;

MachineState machineState = UNZEROED;
bool isZeroed = false;

char serialLine[SERIAL_LINE_MAX];
size_t serialLen = 0;
bool serialOverflow = false;

long cutterLeftPositionPulses = 0;
long cutterRightPositionPulses = 0;
bool cutterPositionKnown = false;
int8_t cutterMotionDirection = 0;
bool cutterLeftFeedbackState = false;
bool cutterRightFeedbackState = false;
unsigned long cutterLeftLastPulseMs = 0;
unsigned long cutterRightLastPulseMs = 0;

// ---------------------------
// Forward declarations
// ---------------------------
void serviceSerial(bool stopOnly);
void handleCommand(char *line, bool stopOnly);
void printHelp();
void printStatus();
void setMachineState(MachineState nextState);
const char *stateToString(MachineState state);
void setCurrentPositionAsZero();

bool moveToInches(float xIn, bool announce);
bool runTravelRotationTest();
bool runCutQueue();
bool performCutCycle();
bool delayWithStop(uint16_t delayMs);
bool waitForTravelMotion();

bool retractCuttersUntilStopped();
bool moveCuttersToTarget(long targetPulses);
void initializeCutterFeedbackTracking(int8_t direction);
void updateCutterFeedback();
void driveCutters(int8_t leftDirection, int8_t rightDirection);
void stopCutters();

void handleStopCommand();
bool checkAndHandleAbort();
void abortActiveOperation(bool announce);
void drainMotionToStop();

bool parseStrictFloat(const char *text, float &outValue);
bool parseSingleCoordinate(char *args, float &outCoord);
bool loadQueueFromCsv(char *args);
bool isCoordinateInRange(float xIn);
bool isManualMoveInRange(float xIn);

```

```
char *trimInPlace(char *text);
void uppercaseInPlace(char *text);

long inchesToSteps(float inches);
float stepsToInches(long steps);
long currentTravelPositionSteps();
long currentTravelSyncErrorSteps();
long clampCutterPosition(long pulses);
long cutterAveragePositionPulses();
bool isCutterAtTarget(long currentPulses, long targetPulses, int8_t direction);
uint8_t leftActiveFeedbackPin();
uint8_t rightActiveFeedbackPin();

void configureDriver(AccelStepper &stepper, bool directionInverted);
void applyNominalMotionProfile();

void moveTravelRelativeSteps(long deltaSteps);
void moveTravelToSteps(long targetSteps);
void setTravelCurrentPosition(long positionSteps);
bool isTravelAxisBusy();
void runTravelAxis();
void stopTravelAxis();

// ---------------------------
// Setup/loop
// ---------------------------
void setup() {
  Serial.begin(SERIAL_BAUD);

  pinMode(ENABLE_PIN, OUTPUT);
  digitalWrite(ENABLE_PIN, LOW);

  pinMode(STATUS_LED_PIN, OUTPUT);

  pinMode(CUTTER_LEFT_EXTEND_PIN, OUTPUT);
  pinMode(CUTTER_LEFT_RETRACT_PIN, OUTPUT);
  pinMode(CUTTER_RIGHT_EXTEND_PIN, OUTPUT);
  pinMode(CUTTER_RIGHT_RETRACT_PIN, OUTPUT);
  pinMode(CUTTER_LEFT_EXTEND_FEEDBACK_PIN, INPUT_PULLUP);
  pinMode(CUTTER_LEFT_RETRACT_FEEDBACK_PIN, INPUT_PULLUP);
  pinMode(CUTTER_RIGHT_EXTEND_FEEDBACK_PIN, INPUT_PULLUP);
  pinMode(CUTTER_RIGHT_RETRACT_FEEDBACK_PIN, INPUT_PULLUP);

  stopCutters();
  cutterLeftFeedbackState = (digitalRead(leftActiveFeedbackPin()) == HIGH);
  cutterRightFeedbackState = (digitalRead(rightActiveFeedbackPin()) == HIGH);
  cutterLeftLastPulseMs = millis();
  cutterRightLastPulseMs = millis();

  configureDriver(travelLeftStepper, TRAVEL_LEFT_DIR_INVERTED);
  configureDriver(travelRightStepper, TRAVEL_RIGHT_DIR_INVERTED);

  applyNominalMotionProfile();
  setTravelCurrentPosition(0);

  setMachineState(UNZEROED);

  Serial.println(F("INFO: Slicer Arm Controller ready"));
```

```
198   Serial.println(F("INFO: State=UNZEROED. Move to the reference mark, then run
          ZERO."));
199   Serial.println(F("INFO: Type HELP for command list."));
200 }
201
202 void loop() {
203   if (cutterMotionDirection != 0) {
204     updateCutterFeedback();
205   }
206   serviceSerial(false);
207 }
208
209 // ---------------------------
210 // Command handling
211 // ---------------------------
212 void serviceSerial(bool stopOnly) {
213   while (Serial.available() > 0) {
214     char c = static_cast<char>(Serial.read());
215
216     if (c == '\r') {
217       continue;
218     }
219
220     if (c == '\n') {
221       if (!serialOverflow) {
222         serialLine[serialLen] = '\0';
223         if (serialLen > 0) {
224           handleCommand(serialLine, stopOnly);
225         }
226       } else {
227         Serial.println(F("ERR: Command too long"));
228       }
229       serialLen = 0;
230       serialOverflow = false;
231       continue;
232     }
233
234     if (serialOverflow) {
235       continue;
236     }
237
238     if (serialLen >= (SERIAL_LINE_MAX - 1)) {
239       serialOverflow = true;
240       continue;
241     }
242
243     serialLine[serialLen++] = c;
244   }
245 }
246
247 void handleCommand(char *line, bool stopOnly) {
248   char *input = trimInPlace(line);
249   if (input[0] == '\0') {
250     return;
251   }
252
253   char *cursor = input;
254   while (*cursor != '\0' && !isspace(static_cast<unsigned char>(*cursor))) {
255     cursor++;
```

```
256   }
257
258   char *args = nullptr;
259   if (*cursor != '\0') {
260     *cursor = '\0';
261     args = trimInPlace(cursor + 1);
262     if (args[0] == '\0') {
263       args = nullptr;
264     }
265   }
266
267   uppercaseInPlace(input);
268
269   if (stopOnly && strcmp(input, "STOP") != 0) {
270     Serial.println(F("ERR: Busy (STOP only)"));
271     return;
272   }
273
274   if (strcmp(input, "HELP") == 0) {
275     printHelp();
276     return;
277   }
278
279   if (strcmp(input, "STATUS") == 0) {
280     printStatus();
281     return;
282   }
283
284   if (strcmp(input, "ZERO") == 0) {
285     if (args != nullptr) {
286       Serial.println(F("ERR: ZERO takes no arguments"));
287       return;
288     }
289     setCurrentPositionAsZero();
290     Serial.println(F("OK: Zero set"));
291     return;
292   }
293
294   if (strcmp(input, "LOAD") == 0) {
295     if (args == nullptr) {
296       Serial.println(F("ERR: Invalid coordinate list"));
297       return;
298     }
299     if (loadQueueFromCsv(args)) {
300       Serial.print(F("OK: Loaded "));
301       Serial.print(queueCount);
302       Serial.println(F(" coordinates"));
303     }
304     return;
305   }
306
307   if (strcmp(input, "ADD") == 0) {
308     float coord = 0.0f;
309     if (!parseSingleCoordinate(args, coord)) {
310       Serial.println(F("ERR: Invalid coordinate"));
311       return;
312     }
313     if (!isCoordinateInRange(coord)) {
314       Serial.println(F("ERR: X out of range"));
```

```
315       return;
316     }
317     if (queueCount >= MAX_CUTS) {
318       Serial.println(F("ERR: Queue full"));
319       return;
320     }
321     cutQueue[queueCount++] = coord;
322     Serial.print(F("OK: Added x="));
323     Serial.print(coord, 3);
324     Serial.println(F(" in"));
325     return;
326   }
327
328   if (strcmp(input, "CLEAR") == 0) {
329     queueCount = 0;
330     Serial.println(F("OK: Queue cleared"));
331     return;
332   }
333
334   if (strcmp(input, "LIST") == 0) {
335     if (queueCount == 0) {
336       Serial.println(F("OK: Queue empty"));
337       return;
338     }
339     Serial.print(F("OK: Queue count="));
340     Serial.println(queueCount);
341     for (uint8_t i = 0; i < queueCount; i++) {
342       Serial.print(F("INFO: ["));
343       Serial.print(i);
344       Serial.print(F("] "));
345       Serial.print(cutQueue[i], 3);
346       Serial.println(F(" in"));
347     }
348     return;
349   }
350
351   if (strcmp(input, "MOVE") == 0) {
352     float coord = 0.0f;
353     if (!parseSingleCoordinate(args, coord)) {
354       Serial.println(F("ERR: Invalid coordinate"));
355       return;
356     }
357     if (!moveToInches(coord, true)) {
358       return;
359     }
360     Serial.println(F("OK: Move complete"));
361     return;
362   }
363
364   if (strcmp(input, "TESTROT") == 0) {
365     if (args != nullptr) {
366       Serial.println(F("ERR: TESTROT takes no arguments"));
367       return;
368     }
369     if (!runTravelRotationTest()) {
370       return;
371     }
372     Serial.println(F("OK: Rotation test complete"));
373     return;
```

```
374   }
375
376   if (strcmp(input, "TESTCUT") == 0) {
377     if (args != nullptr) {
378       Serial.println(F("ERR: TESTCUT takes no arguments"));
379       return;
380     }
381     if (!performCutCycle()) {
382       return;
383     }
384     Serial.println(F("OK: Cutter test complete"));
385     return;
386   }
387
388   if (strcmp(input, "CUTEXTEND") == 0) {
389     if (args != nullptr) {
390       Serial.println(F("ERR: CUTEXTEND takes no arguments"));
391       return;
392     }
393     stopRequested = false;
394     setMachineState(CUTTING);
395     initializeCutterFeedbackTracking(1);
396     driveCutters(1, 1);
397     Serial.println(F("OK: Cutter manual extend"));
398     return;
399   }
400
401   if (strcmp(input, "CUTRETRACT") == 0) {
402     if (args != nullptr) {
403       Serial.println(F("ERR: CUTRETRACT takes no arguments"));
404       return;
405     }
406     stopRequested = false;
407     setMachineState(CUTTING);
408     initializeCutterFeedbackTracking(-1);
409     driveCutters(-1, -1);
410     Serial.println(F("OK: Cutter manual retract"));
411     return;
412   }
413
414   if (strcmp(input, "CUTSTOP") == 0) {
415     if (args != nullptr) {
416       Serial.println(F("ERR: CUTSTOP takes no arguments"));
417       return;
418     }
419     stopCutters();
420     if (machineState == CUTTING) {
421       setMachineState(IDLE);
422     }
423     Serial.println(F("OK: Cutter manual stop"));
424     return;
425   }
426
427   if (strcmp(input, "RUN") == 0) {
428     if (!runCutQueue()) {
429       return;
430     }
431     Serial.println(F("OK: Run complete"));
432     return;
```

```
  }

  if (strcmp(input, "STOP") == 0) {
    handleStopCommand();
    return;
  }

  Serial.println(F("ERR: Unknown command"));
}

void printHelp() {
  Serial.println(F("OK: Commands"));
  Serial.println(F("INFO: HELP"));
  Serial.println(F("INFO: STATUS"));
  Serial.println(F("INFO: ZERO"));
  Serial.println(F("INFO: LOAD x1,x2,x3"));
  Serial.println(F("INFO: ADD x"));
  Serial.println(F("INFO: CLEAR"));
  Serial.println(F("INFO: LIST"));
  Serial.println(F("INFO: MOVE x   (manual move supports negative values)"));
  Serial.println(F("INFO: TESTROT"));
  Serial.println(F("INFO: TESTCUT"));
  Serial.println(F("INFO: CUTEXTEND"));
  Serial.println(F("INFO: CUTRETRACT"));
  Serial.println(F("INFO: CUTSTOP"));
  Serial.println(F("INFO: RUN"));
  Serial.println(F("INFO: STOP"));
}

void printStatus() {
  Serial.print(F("OK: STATE="));
  Serial.print(stateToString(machineState));
  Serial.print(F(" ZEROED="));
  Serial.print(isZeroed ? 1 : 0);
  Serial.print(F(" X_IN="));
  Serial.print(stepsToInches(currentTravelPositionSteps()), 3);
  Serial.print(F(" SYNC_ERR_STEPS="));
  Serial.print(currentTravelSyncErrorSteps());
  Serial.print(F(" CUT_L_PULSES="));
  Serial.print(cutterLeftPositionPulses);
  Serial.print(F(" CUT_R_PULSES="));
  Serial.print(cutterRightPositionPulses);
  Serial.print(F(" CUT_KNOWN="));
  Serial.print(cutterPositionKnown ? 1 : 0);
  Serial.print(F(" QUEUE="));
  Serial.println(queueCount);
}

// ---------------------------
// Motion and cut helpers
// ---------------------------
bool moveToInches(float xIn, bool announce) {
  if (!isZeroed) {
    Serial.println(F("ERR: Must ZERO first"));
    return false;
  }
  if (!isManualMoveInRange(xIn)) {
    Serial.println(F("ERR: X out of range"));
    return false;
```

```
  }

  if (announce) {
    Serial.print(F("INFO: Moving to "));
    Serial.print(xIn, 3);
    Serial.println(F(" in"));
  }

  setMachineState(MOVING);
  moveTravelToSteps(inchesToSteps(xIn));

  if (!waitForTravelMotion()) {
    return false;
  }

  setMachineState(IDLE);
  return true;
}

bool runTravelRotationTest() {
  if (!isZeroed) {
    Serial.println(F("ERR: Must ZERO first"));
    return false;
  }

  Serial.print(F("INFO: Rotation test steps="));
  Serial.println(TRAVEL_TEST_ROTATION_STEPS);

  setMachineState(MOVING);
  moveTravelRelativeSteps(TRAVEL_TEST_ROTATION_STEPS);

  if (!waitForTravelMotion()) {
    return false;
  }

  setMachineState(IDLE);
  return true;
}

bool runCutQueue() {
  if (!isZeroed) {
    Serial.println(F("ERR: Must ZERO first"));
    return false;
  }
  if (queueCount == 0) {
    Serial.println(F("ERR: Queue empty"));
    return false;
  }

  stopRequested = false;

  for (uint8_t i = 0; i < queueCount; i++) {
    Serial.print(F("INFO: Target "));
    Serial.print(i + 1);
    Serial.print(F("/"));
    Serial.print(queueCount);
    Serial.print(F(" x="));
    Serial.print(cutQueue[i], 3);
    Serial.println(F(" in"));
```

```
551
552     if (!moveToInches(cutQueue[i], false)) {
553       return false;
554     }
555
556     Serial.print(F("INFO: Cutting at "));
557     Serial.print(cutQueue[i], 3);
558     Serial.println(F(" in"));
559     if (!performCutCycle()) {
560       return false;
561     }
562   }
563
564   setMachineState(IDLE);
565   return true;
566 }
567
568 bool performCutCycle() {
569   setMachineState(CUTTING);
570
571   if (!retractCuttersUntilStopped()) {
572     return false;
573   }
574
575   Serial.println(F("INFO: Cutter extend"));
576   if (!moveCuttersToTarget(CUTTER_FULL_STROKE_PULSES)) {
577     return false;
578   }
579   if (!delayWithStop(CUTTER_EXTEND_SETTLE_MS)) {
580     return false;
581   }
582
583   Serial.println(F("INFO: Cutter retract"));
584   if (!retractCuttersUntilStopped()) {
585     return false;
586   }
587   if (!delayWithStop(CUTTER_RETRACT_SETTLE_MS)) {
588     return false;
589   }
590
591   setMachineState(IDLE);
592   return true;
593 }
594
595 bool delayWithStop(uint16_t delayMs) {
596   unsigned long start = millis();
597   while ((millis() - start) < delayMs) {
598     serviceSerial(true);
599     if (checkAndHandleAbort()) {
600       return false;
601     }
602     delay(2);
603   }
604   return true;
605 }
606
607 bool waitForTravelMotion() {
608   while (isTravelAxisBusy()) {
609     serviceSerial(true);
```

```
610     if (checkAndHandleAbort()) {
611       return false;
612     }
613     runTravelAxis();
614   }
615   return true;
616 }
617
618 bool retractCuttersUntilStopped() {
619   if (cutterPositionKnown && cutterLeftPositionPulses == 0 &&
          cutterRightPositionPulses == 0) {
620     return true;
621   }
622
623   Serial.println(F("INFO: Cutter retract reference"));
624   initializeCutterFeedbackTracking(-1);
625
626   unsigned long start = millis();
627   unsigned long graceDeadline = start + CUTTER_RETRACT_GRACE_MS;
628   bool leftRunning = true;
629   bool rightRunning = true;
630
631   while (true) {
632     serviceSerial(true);
633     if (checkAndHandleAbort()) {
634       return false;
635     }
636
637     updateCutterFeedback();
638
639     unsigned long now = millis();
640     if (now >= graceDeadline) {
641       if (leftRunning && (now - cutterLeftLastPulseMs) > CUTTER_NO_PULSE_STOP_MS)
              {
642         leftRunning = false;
643       }
644       if (rightRunning && (now - cutterRightLastPulseMs) >
              CUTTER_NO_PULSE_STOP_MS) {
645         rightRunning = false;
646       }
647     }
648
649     driveCutters(leftRunning ? -1 : 0, rightRunning ? -1 : 0);
650
651     if (!leftRunning && !rightRunning) {
652       stopCutters();
653       // The actuators have stopped producing retract pulses, so treat this as
              the cutter reference position.
654       cutterLeftPositionPulses = 0;
655       cutterRightPositionPulses = 0;
656       cutterPositionKnown = true;
657       return true;
658     }
659
660     if ((now - start) > CUTTER_RETRACT_TIMEOUT_MS) {
661       stopCutters();
662       cutterPositionKnown = false;
663       Serial.println(F("ERR: Cutter retract timeout"));
664       return false;
```

```
    }
  }
}

bool moveCuttersToTarget(long targetPulses) {
  targetPulses = clampCutterPosition(targetPulses);

  if (!cutterPositionKnown) {
    if (!retractCuttersUntilStopped()) {
      return false;
    }
  }

  long averagePosition = cutterAveragePositionPulses();
  if (averagePosition == targetPulses && cutterLeftPositionPulses == targetPulses &&
      cutterRightPositionPulses == targetPulses) {
    return true;
  }

  int8_t direction = (targetPulses > averagePosition) ? 1 : -1;
  initializeCutterFeedbackTracking(direction);

  unsigned long start = millis();

  while (true) {
    serviceSerial(true);
    if (checkAndHandleAbort()) {
      return false;
    }

    updateCutterFeedback();

    bool leftDone = isCutterAtTarget(cutterLeftPositionPulses, targetPulses, direction);
    bool rightDone = isCutterAtTarget(cutterRightPositionPulses, targetPulses, direction);

    if (leftDone && rightDone) {
      stopCutters();
      cutterLeftPositionPulses = targetPulses;
      cutterRightPositionPulses = targetPulses;
      cutterPositionKnown = true;
      return true;
    }

    long syncError = (cutterLeftPositionPulses - cutterRightPositionPulses) * direction;
    int8_t leftDirection = leftDone ? 0 : direction;
    int8_t rightDirection = rightDone ? 0 : direction;

    if (syncError > CUTTER_SYNC_TOLERANCE_PULSES) {
      leftDirection = 0;
    } else if (syncError < -CUTTER_SYNC_TOLERANCE_PULSES) {
      rightDirection = 0;
    }

    driveCutters(leftDirection, rightDirection);

```

```
720     if ((millis() - start) > CUTTER_MOVE_TIMEOUT_MS) {
721       stopCutters();
722       cutterPositionKnown = false;
723       Serial.println(F("ERR: Cutter move timeout"));
724       return false;
725     }
726   }
727 }
728
729 void initializeCutterFeedbackTracking(int8_t direction) {
730   cutterMotionDirection = direction;
731   cutterLeftFeedbackState = (digitalRead(leftActiveFeedbackPin()) == HIGH);
732   cutterRightFeedbackState = (digitalRead(rightActiveFeedbackPin()) == HIGH);
733   cutterLeftLastPulseMs = millis();
734   cutterRightLastPulseMs = millis();
735 }
736
737 void updateCutterFeedback() {
738   bool leftState = (digitalRead(leftActiveFeedbackPin()) == HIGH);
739   if (leftState != cutterLeftFeedbackState) {
740     cutterLeftFeedbackState = leftState;
741     cutterLeftLastPulseMs = millis();
742     // Count one edge per hall cycle so the datasheet pulses/inch value remains
          valid.
743     if (leftState == HIGH) {
744       if (cutterMotionDirection > 0) {
745         cutterLeftPositionPulses = clampCutterPosition(cutterLeftPositionPulses +
              1L);
746       } else if (cutterMotionDirection < 0 && cutterPositionKnown) {
747         cutterLeftPositionPulses = clampCutterPosition(cutterLeftPositionPulses -
              1L);
748       }
749     }
750   }
751
752   bool rightState = (digitalRead(rightActiveFeedbackPin()) == HIGH);
753   if (rightState != cutterRightFeedbackState) {
754     cutterRightFeedbackState = rightState;
755     cutterRightLastPulseMs = millis();
756     if (rightState == HIGH) {
757       if (cutterMotionDirection > 0) {
758         cutterRightPositionPulses = clampCutterPosition(cutterRightPositionPulses
              + 1L);
759       } else if (cutterMotionDirection < 0 && cutterPositionKnown) {
760         cutterRightPositionPulses = clampCutterPosition(cutterRightPositionPulses
              - 1L);
761       }
762     }
763   }
764 }
765
766 uint8_t leftActiveFeedbackPin() {
767   return (cutterMotionDirection < 0) ? CUTTER_LEFT_RETRACT_FEEDBACK_PIN :
        CUTTER_LEFT_EXTEND_FEEDBACK_PIN;
768 }
769
770 uint8_t rightActiveFeedbackPin() {
771   return (cutterMotionDirection < 0) ? CUTTER_RIGHT_RETRACT_FEEDBACK_PIN :
        CUTTER_RIGHT_EXTEND_FEEDBACK_PIN;
```

```
}

void driveCutters(int8_t leftDirection, int8_t rightDirection) {
  digitalWrite(CUTTER_LEFT_EXTEND_PIN, (leftDirection > 0) ? HIGH : LOW);
  digitalWrite(CUTTER_LEFT_RETRACT_PIN, (leftDirection < 0) ? HIGH : LOW);
  digitalWrite(CUTTER_RIGHT_EXTEND_PIN, (rightDirection > 0) ? HIGH : LOW);
  digitalWrite(CUTTER_RIGHT_RETRACT_PIN, (rightDirection < 0) ? HIGH : LOW);
}

void stopCutters() {
  driveCutters(0, 0);
  cutterMotionDirection = 0;
}

void handleStopCommand() {
  stopRequested = true;
  stopTravelAxis();
  stopCutters();
  Serial.println(F("OK: Stop requested"));

  if (machineState != MOVING && machineState != CUTTING) {
    abortActiveOperation(false);
    Serial.println(F("INFO: State set to ABORTED. Move to reference, then run
        ZERO."));
  }
}

bool checkAndHandleAbort() {
  if (!stopRequested) {
    return false;
  }
  abortActiveOperation(true);
  return true;
}

void abortActiveOperation(bool announce) {
  stopTravelAxis();
  stopCutters();
  drainMotionToStop();

  cutterPositionKnown = false;
  isZeroed = false;
  stopRequested = false;
  setMachineState(ABORTED);

  if (announce) {
    Serial.println(F("ERR: Operation aborted"));
  }
}

void drainMotionToStop() {
  unsigned long start = millis();
  while (isTravelAxisBusy()) {
    runTravelAxis();
    if ((millis() - start) > ABORT_DRAIN_TIMEOUT_MS) {
      break;
    }
  }
}
```

```
830
831 // ---------------------------
832 // Parsing/validation helpers
833 // ---------------------------
834 bool parseStrictFloat(const char *text, float &outValue) {
835   if (text == nullptr || *text == '\0') {
836     return false;
837   }
838
839   char *endPtr = nullptr;
840   double parsed = strtod(text, &endPtr);
841   if (endPtr == text) {
842     return false;
843   }
844   while (*endPtr != '\0' && isspace(static_cast<unsigned char>(*endPtr))) {
845     endPtr++;
846   }
847   if (*endPtr != '\0') {
848     return false;
849   }
850   if (isnan(parsed) || isinf(parsed)) {
851     return false;
852   }
853
854   outValue = static_cast<float>(parsed);
855   return true;
856 }
857
858 bool parseSingleCoordinate(char *args, float &outCoord) {
859   if (args == nullptr) {
860     return false;
861   }
862
863   char *token = trimInPlace(args);
864   if (token[0] == '\0') {
865     return false;
866   }
867
868   return parseStrictFloat(token, outCoord);
869 }
870
871 bool loadQueueFromCsv(char *args) {
872   float tempQueue[MAX_CUTS];
873   uint8_t tempCount = 0;
874
875   char *segmentStart = args;
876   while (true) {
877     char *comma = strchr(segmentStart, ',');
878     if (comma != nullptr) {
879       *comma = '\0';
880     }
881
882     char *token = trimInPlace(segmentStart);
883     if (token[0] == '\0') {
884       Serial.println(F("ERR: Invalid coordinate list"));
885       return false;
886     }
887
888     float coord = 0.0f;
```

```
889       if (!parseStrictFloat(token, coord)) {
890         Serial.println(F("ERR: Invalid coordinate list"));
891         return false;
892       }
893       if (!isCoordinateInRange(coord)) {
894         Serial.println(F("ERR: X out of range"));
895         return false;
896       }
897       if (tempCount >= MAX_CUTS) {
898         Serial.println(F("ERR: Queue full"));
899         return false;
900       }
901       tempQueue[tempCount++] = coord;
902
903       if (comma == nullptr) {
904         break;
905       }
906
907       segmentStart = comma + 1;
908       if (*segmentStart == '\0') {
909         Serial.println(F("ERR: Invalid coordinate list"));
910         return false;
911       }
912     }
913
914     if (tempCount == 0) {
915       Serial.println(F("ERR: Invalid coordinate list"));
916       return false;
917     }
918
919     for (uint8_t i = 0; i < tempCount; i++) {
920       cutQueue[i] = tempQueue[i];
921     }
922     queueCount = tempCount;
923     return true;
924   }
925
926   bool isCoordinateInRange(float xIn) {
927     return (xIn >= 0.0f && xIn <= MAX_TRAVEL_IN);
928   }
929
930   bool isManualMoveInRange(float xIn) {
931     return (xIn >= -MAX_TRAVEL_IN && xIn <= MAX_TRAVEL_IN);
932   }
933
934   char *trimInPlace(char *text) {
935     if (text == nullptr) {
936       return text;
937     }
938
939     while (*text != '\0' && isspace(static_cast<unsigned char>(*text))) {
940       text++;
941     }
942
943     if (*text == '\0') {
944       return text;
945     }
946
947     char *end = text + strlen(text) - 1;
```

```
948   while (end > text && isspace(static_cast<unsigned char>(*end))) {
949     *end = '\0';
950     end--;
951   }
952
953   return text;
954 }
955
956 void uppercaseInPlace(char *text) {
957   while (*text != '\0') {
958     *text = static_cast<char>(toupper(static_cast<unsigned char>(*text)));
959     text++;
960   }
961 }
962
963 // ---------------------------
964 // Units/state utilities
965 // ---------------------------
966 long inchesToSteps(float inches) {
967   return lroundf(inches * TRAVEL_STEPS_PER_INCH);
968 }
969
970 float stepsToInches(long steps) {
971   return static_cast<float>(steps) / TRAVEL_STEPS_PER_INCH;
972 }
973
974 long currentTravelPositionSteps() {
975   long leftPosition = travelLeftStepper.currentPosition();
976   long rightPosition = travelRightStepper.currentPosition();
977   return (leftPosition + rightPosition) / 2L;
978 }
979
980 long currentTravelSyncErrorSteps() {
981   return travelLeftStepper.currentPosition() -
          travelRightStepper.currentPosition();
982 }
983
984 long clampCutterPosition(long pulses) {
985   if (pulses < 0L) {
986     return 0L;
987   }
988   if (pulses > CUTTER_FULL_STROKE_PULSES) {
989     return CUTTER_FULL_STROKE_PULSES;
990   }
991   return pulses;
992 }
993
994 long cutterAveragePositionPulses() {
995   return (cutterLeftPositionPulses + cutterRightPositionPulses) / 2L;
996 }
997
998 bool isCutterAtTarget(long currentPulses, long targetPulses, int8_t direction) {
999   return (direction > 0) ? (currentPulses >= targetPulses) : (currentPulses <=
          targetPulses);
1000 }
1001
1002 void configureDriver(AccelStepper &stepper, bool directionInverted) {
1003   stepper.setMinPulseWidth(STEPPER_MIN_PULSE_WIDTH_US);
1004   stepper.setPinsInverted(directionInverted, false, false);
```

```
}

void applyNominalMotionProfile() {
  float maxSpeedSteps = TRAVEL_MAX_SPEED_IN_S * TRAVEL_STEPS_PER_INCH;
  float accelSteps = TRAVEL_ACCEL_IN_S2 * TRAVEL_STEPS_PER_INCH;
  travelLeftStepper.setMaxSpeed(maxSpeedSteps);
  travelLeftStepper.setAcceleration(accelSteps);
  travelRightStepper.setMaxSpeed(maxSpeedSteps);
  travelRightStepper.setAcceleration(accelSteps);
}

void moveTravelRelativeSteps(long deltaSteps) {
  travelLeftStepper.move(deltaSteps);
  travelRightStepper.move(deltaSteps);
}

void moveTravelToSteps(long targetSteps) {
  travelLeftStepper.moveTo(targetSteps);
  travelRightStepper.moveTo(targetSteps);
}

void setTravelCurrentPosition(long positionSteps) {
  travelLeftStepper.setCurrentPosition(positionSteps);
  travelRightStepper.setCurrentPosition(positionSteps);
}

void setCurrentPositionAsZero() {
  // ZERO is the only travel-axis reference operation in this prototype.
  stopRequested = false;
  setTravelCurrentPosition(0);
  isZeroed = true;
  setMachineState(IDLE);
}

bool isTravelAxisBusy() {
  return (travelLeftStepper.distanceToGo() != 0 ||
      travelRightStepper.distanceToGo() != 0);
}

void runTravelAxis() {
  travelLeftStepper.run();
  travelRightStepper.run();
}

void stopTravelAxis() {
  travelLeftStepper.stop();
  travelRightStepper.stop();
}

void setMachineState(MachineState nextState) {
  machineState = nextState;
  digitalWrite(STATUS_LED_PIN, (machineState == IDLE) ? HIGH : LOW);
}

const char *stateToString(MachineState state) {
  switch (state) {
  case UNZEROED:
    return "UNZEROED";
  case IDLE:
```

```
    return "IDLE";
  case MOVING:
    return "MOVING";
  case CUTTING:
    return "CUTTING";
  case ABORTED:
    return "ABORTED";
  default:
    return "UNKNOWN";
  }
}
```

Listing 1: Arduino sketch used for the prototype