



Fullum Polisher Machine Control and Firmware Specification

Cédric implementation reference - Draft v1.3

Antoine Letarte, Eng., M.A.Sc.

Project	Polisher Software Suite
Client	Optiques Fullum
Document	ATM-2026-POL-CTRL-001
Date	2026-07-06
Revision	Draft v1.3 - AK60/RPi5/BiSS-C source sync

Contents

1	Executive Summary	1
1.1	Scope at a glance	1
1.2	v1 build order (read this first)	1
1.3	How to read this document	1
1.4	Architecture overview	3
1.5	Revision and distribution	4
2	Fullum Polisher - Machine Control & Firmware Specification v1	5
2.1	Revision history	5
2.2	Purpose and Context	5
2.2.1	People	6
2.2.2	The Machine	6
2.2.3	Read This First - Cédric's v1 Build Target	6
2.2.4	Normand Manual-Mode Quick Start	7
2.3	Overview and Methodology	8
2.3.1	The Problem We Are Solving	8
2.3.2	What the Machine Does	8
2.3.3	The Three Truths	10
2.3.4	Force Modulation - Why It Matters	11
2.3.5	The Journey: Manual First, Programs Later	12
2.3.6	What This Means for the Machine Control Stack	13
2.3.7	How Manual and Auto Mode Fit Together	14
2.4	Hardware Basis and Integration Ownership	15
2.4.1	Sensors	15
2.4.2	Actuators	16
2.4.3	Cam Mechanism	19
2.4.4	Controller Hardware	19
2.4.5	Safety Hardware	20
2.4.6	Operator Control Panel (HOA - Hand / Off / Auto)	21
2.4.7	Power	21
2.4.8	Transmission Summary	21
2.5	System Boundary	22
2.5.1	What You Own (In Scope)	22
2.5.2	What You Do NOT Own (Out of Scope)	22
2.6	Architecture Context	23
2.6.1	Three Truths — Implementation Constraint	24
2.7	Runtime Split	24
2.7.1	Host Controller (Python on RPi or PC)	24
2.7.2	Teensy Firmware (C++ on Teensy 4.x)	24
2.7.3	What the Teensy Computes Locally	25
2.7.4	What the Teensy Must NOT Decide	25
2.8	Input Contract: Controller-Job	25
2.8.1	What You Receive	25
2.8.2	Job Intake Validation (4-Gate Sequence)	26
2.8.3	Controller-Job Top-Level Structure	26
2.8.4	Segment Structure	27
2.8.5	Force Modulation Block	28
2.9	Output Contract: Run Artifacts	29

2.9.1	Required Outputs	29
2.9.2	Run-Log Top-Level Structure	29
2.9.3	Executed Segment Structure	30
2.9.4	Result States	31
2.9.5	Event Record Format	31
2.9.6	Alarm Promotion	32
2.10	Telemetry Specification	32
2.10.1	Channel Table	32
2.10.2	Telemetry Principles	34
2.10.3	Telemetry CSV Format	34
2.10.4	KWR75B-CAN Transport – Dedicated CAN Link	34
2.10.5	Arm Kinematic Linearization (Firmware Obligation)	36
2.11	State Machine	37
2.11.1	States	37
2.11.2	Transition Table	37
2.11.3	State Machine Rules	38
2.12	Safety, Interlocks, and Fault Handling	38
2.12.1	Interlocks and Fault Responses	38
2.12.2	Safety Enforcement Split	39
2.12.3	Watchdog / Heartbeat	41
2.12.4	Startup Safety	41
2.13	Segment Execution Lifecycle	42
2.13.1	Pause During Segment	43
2.13.2	Abort Sequence	44
2.14	Manual Operation Mode	44
2.14.1	Purpose	44
2.14.2	How It Works	44
2.14.3	Manual Mode vs Job Mode	45
2.14.4	What the Operator Controls	45
2.14.5	Manual Mode State Rules	47
2.14.6	Manual Session Log	47
2.14.7	Manual Session Telemetry	48
2.14.8	Host ↔ Teensy in Manual Mode	49
2.14.9	Tool-Weight Compensation and Tool Removal Workflow	49
2.15	Host ↔ Teensy Protocol	50
2.15.1	Philosophy	51
2.15.2	Message Classes	51
2.15.3	Protocol Requirements	52
2.15.4	Example Frame Concepts	52
2.16	Machine Capability Descriptor	53
2.17	Timing Summary	55
2.18	Data Storage, Export, and Remote Access	56
2.18.1	Local Storage	56
2.18.2	File Structure	57
2.18.3	Remote Access (Tailscale)	58
2.18.4	Machine Status Endpoint	58
2.18.5	Auto-Sync (Post-Run)	58
2.18.6	Telemetry Retention and Post-Processing Contract	59
2.18.7	What This Does NOT Include	60

2.19	First-Proving Scope (v1)	60
2.19.1	In Scope for v1	60
2.19.2	Explicitly Optional for v1	61
2.20	Acceptance Criteria	62
2.20.1	Input Contract	62
2.20.2	Execution	62
2.20.3	Telemetry	62
2.20.4	State Machine	62
2.20.5	Safety	62
2.20.6	Output Contract	63
2.20.7	Manual Mode	63
2.20.8	Data Storage and Remote Access	63
2.20.9	Boundary Discipline	63
2.20.10	Kinematics and Geometric Gate	64
2.20.11	KWR75B-CAN Transport	64
2.20.12	Tool-Weight Compensation and Safe Removal Workflow	64
2.21	Recommended Follow-On Documents	64
2.22	Open Questions for Cédric (Control & Firmware Side Only)	65
2.22.1	Zone 1 — Table (Control Interface)	65
2.22.2	Zone 2 — Spindle / Toolhead Drive (Control Interface)	66
2.22.3	Zone 4 — Z-Axis / Force Actuator (Control Interface)	66
2.22.4	Cross-Cutting Items	66
2.22.5	Future Scope (post-v1)	66
2.23	Related Documents	67
2.24	Appendix A: Large-Format Control Figures	67
2.24.1	Appendix A.1: Figure 4 Landscape Copy	67
2.24.2	Appendix A.2: Figure 5 Landscape Copy	68
2.24.3	Appendix A.3: Figure 6 Landscape Copy	68

List of Figures

Figure 1	Machine delivery boundary and data flow. Controller jobs enter the machine; run logs and telemetry leave for downstream analysis. Source: Fullum-Polisher-Delivery-Scope.mmd	3
Figure 2	— Machine delivery boundary and data flow. Controller jobs enter the machine; run logs and telemetry leave for downstream analysis. Source: Fullum-Polisher-Delivery-Scope.mmd.	9
Figure 3	— The three truths preserved end-to-end. Commanded fields are flat scalars; actual fields always use the <code>_mean</code> / <code>_min</code> / <code>_max</code> triplet. The two never share a field name — that structural separation is what makes confusion impossible and what makes Phase 1 manual telemetry directly comparable to Phase 3 program runs. Mermaid source: Fullum-Polisher-Three-Truths.mmd.	11
Figure 4	— Phase roadmap. Cédric ships Phase 1 (manual + telemetry). Phases 2–4 reuse the same firmware, same hardware, same data contracts — they are feature work on top of v1, not redesigns. Mermaid source: Fullum-Polisher-Phase-Roadmap.mmd.	12
Figure 5	— Full system architecture. Sensors feed the Teensy real-time loop; Teensy drives actuators and exchanges setpoints/telemetry with the host; the host owns state machine, UI, and persistence. The hardware safety chain (E-stop, brake, HOA Hand fallback) runs independently of software and can cut all drives directly. See Appendix A.1 for the full-size landscape copy. Mermaid source: Fullum-Polisher-System-Architecture.mmd. (The older Fullum-Polisher-Control-Stack.* files are deprecated.)	23
Figure 6	— Controller state machine. 11 states; FAULTED only exits via explicit operator reset. Note that <code>JOB_LOADED</code> , <code>READY</code> , <code>RUNNING</code> , <code>PAUSED</code> , and segment-related transitions exist in v1 firmware even though production v1 runs are MANUAL-only. Preserving these states is what allows Phase 3 program execution to drop in without rework. See Appendix A.2 for the full-size landscape copy. Mermaid source: Fullum-Polisher-State-Machine.mmd.	37
Figure 7	— Three-layer safety hierarchy. See Appendix A.3 for a readable landscape copy. Mermaid source: Fullum-Polisher-Safety-Hierarchy.mmd.	40
Figure 8	— Segment execution sequence. Full-page layout in the PDF. The host orchestrates at segment granularity; the Teensy runs the inner loop at 1 kHz. Mermaid source: Fullum-Polisher-Segment-Lifecycle.mmd.	42
Figure 9	— Loop and stream rates. The single Teensy monotonic clock anchors every channel; host UTC is bound to that clock at run start and re-anchored each segment (or every 5 min during open-ended manual sessions). Mermaid source: Fullum-Polisher-Timing-Rates.mmd.	55
Figure 10	Large-format landscape copy of Figure 4, full system architecture.	67
Figure 11	Readable large-format copy of Figure 5, controller state machine. This copy uses the same state content as Fullum-Polisher-State-Machine.mmd, arranged vertically so transition labels remain readable on the landscape PDF page.	68
Figure 12	Large-format landscape copy of Figure 6, same safety hierarchy content arranged horizontally for readability.	68

List of Tables

Table 1	Revision and distribution metadata for the Cédric review package.	4
Table 2	Source document revision history.	5
Table 3	Project roles and responsibilities for the Fullum polisher control implementation.	6
Table 4	Planned, commanded, and actual data layers preserved by the controller.	10
Table 5	Electrical supply rails required by the machine-side control hardware.	21
Table 6	Firmware and host-controller responsibilities in Cédric’s implementation scope.	22
Table 7	Upstream planning and analysis functions excluded from the machine firmware delivery.	23
Table 8	Host-controller responsibilities and non-real-time orchestration duties.	24
Table 9	Teensy firmware responsibilities executed in the hard-real-time loop.	25
Table 10	Required run artifacts emitted by the machine controller.	29
Table 11	Canonical run result states recorded in run and manual-session logs.	31
Table 12	Dedicated KWR75B-CAN sensor-link topology and protocol assumptions.	35
Table 13	Mechanical geometry inputs consumed by the firmware linearization model.	36
Table 14	Safety enforcement layers and their expected response times.	39
Table 15	Operational differences between planned job execution and touchscreen manual mode. .	45
Table 16	Manual-mode setpoints and defaults exposed on the operator touchscreen.	45
Table 17	Host-to-Teensy command messages used during software manual mode.	49
Table 18	Serial protocol requirements for deterministic host-to-Teensy communication.	52
Table 19	Timing targets for control loops, command streams, telemetry, and events.	56
Table 20	Local storage requirements for reliable telemetry and run-artifact retention.	57
Table 21	Remote-access requirements for commissioning, diagnostics, and telemetry retrieval. . .	58
Table 22	Post-run synchronization requirements and operator fallback path.	59
Table 23	Baseline telemetry retention policy for long-duration polisher operation.	60
Table 24	v1 firmware and controller capabilities mapped to acceptance criteria.	61
Table 25	Recommended follow-on control documents after the v1 firmware specification.	65

1 Executive Summary

TL;DR This document is the **machine-side implementation target** for the Fullum swing-arm polisher overhaul. It defines what Cédric builds on the firmware and host-controller side so the machine integrates cleanly with the broader Atomaste Polisher Software Suite: scope boundary, sensor/actuator interfaces, host ↔ Teensy protocol, telemetry contract, safety model, state machine, manual-mode workflow, and acceptance criteria.

1.1 Scope at a glance

In scope for Cédric — Teensy firmware (1 kHz inner loop, force PID, modulation, safety), host controller on the RPi (state machine, UI, job intake, telemetry persistence, run reporting), sensor/actuator integration, safety chain, data storage and remote access on the machine.

Out of scope for this delivery — figuring strategy, metrology interpretation, dwell-map generation, Preston-coefficient calibration, pass planning, post-run normalization. Those live in polisher-sim / polisher-post and are Antoine's responsibility.

1.2 v1 build order (read this first)

1. **Safe manual operation with telemetry** — Normand can set force, table RPM, spindle RPM, toolhead spindle rotation direction (clockwise/counter-clockwise), and optional modulation from the touchscreen, with every sample logged. This is the day-one mode and the data foundation for Phase 2/3.
2. **Tool-weight compensation and safe removal workflow** — force control subtracts the known tool/head weight from the KWR75B-CAN reading during normal contact operation. For removal, the operator stops the machine, locks the swing arm/arc, removes the cam arm nut to free arm rotation, moves the arm aside, and removes the tool from the blank by hand.
3. **Host ↔ Teensy setpoint / telemetry protocol** — host sends setpoints at 10–50 Hz, Teensy runs the inner loop, both sides ACK/NACK explicitly.
4. **Controller-job intake** — accept only controller-job.v1 packages that pass the 4-gate validation (schema, machine ID, controller version, capability check).
5. **Run artifacts** — emit run-logs, manual-session logs, timestamped telemetry CSVs, event logs, alarms, and integrity hashes in a predictable /data/ layout on a USB SSD.

1.3 How to read this document

- §1–§2 — context, people, the phase roadmap (manual → programs), why the telemetry matters.
- §3–§5 — hardware basis, scope boundary, architecture.
- §6–§9 — runtime split, I/O contracts, telemetry specification, KWR75B-CAN transport, arm kinematic linearization.
- §10–§12 — state machine, safety / interlocks, segment lifecycle.
- §13 — manual mode (first-class citizen, not an afterthought).

- §14–§15 — host ↔ Teensy protocol constraints, machine capability descriptor.
- §16–§17 — timing, data storage, remote access.
- §18–§19 — v1 scope checklist and acceptance criteria.
- §20–§22 — follow-on documents, open questions for Cédric, related material.

1.4 Architecture overview

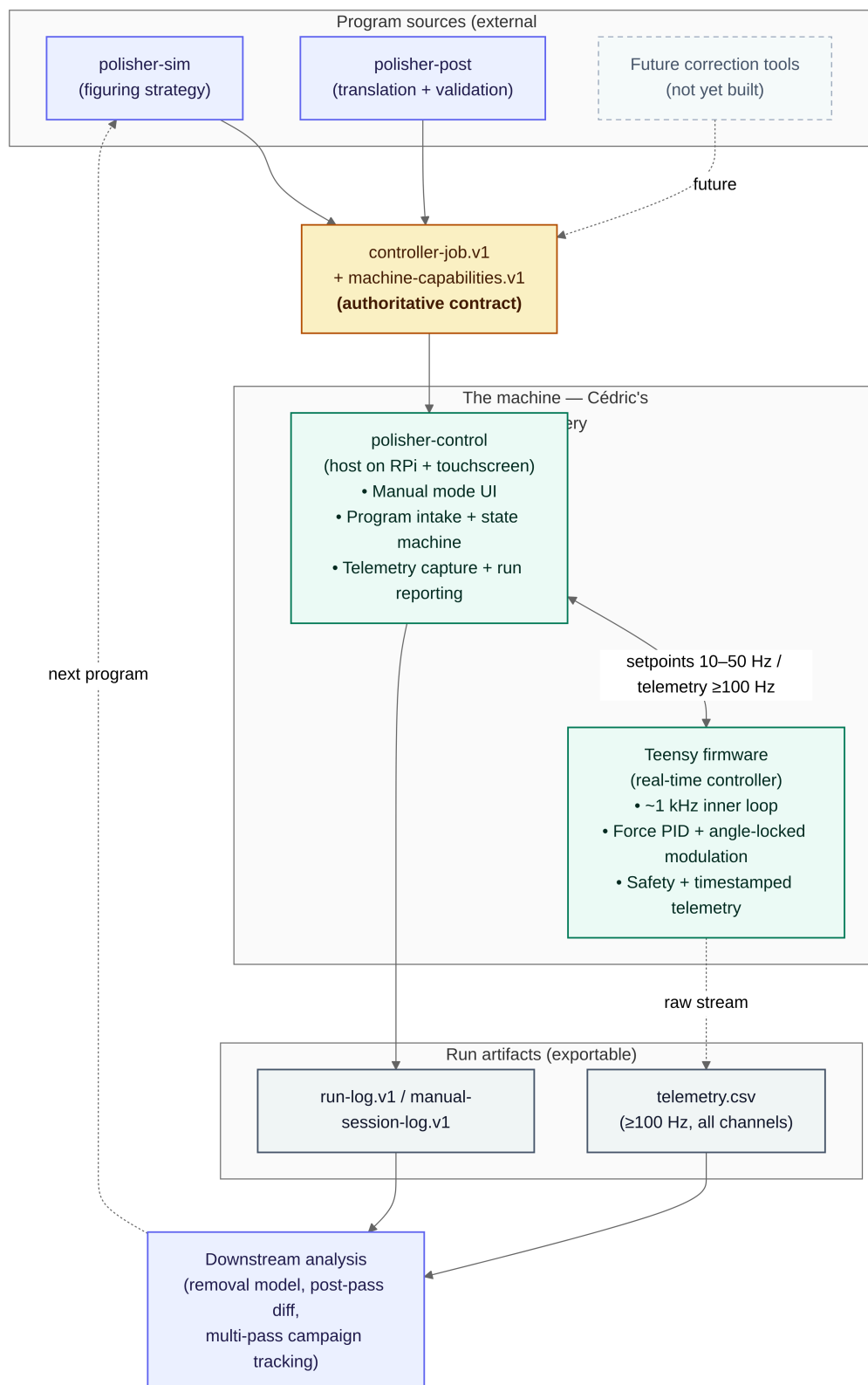


Figure 1 — Machine delivery boundary and data flow. Controller jobs enter the machine; run logs and telemetry leave for downstream analysis. Source: [Fullum-Polisher-Delivery-Scope.mmd](#).

1.5 Revision and distribution

Table 1 — Revision and distribution metadata for the Cédric review package.

Field	Value
Document number	ATM-2026-POL-CTRL-001
Revision	Draft v1.3 - AK60/RPi5/BiSS-C source sync
Date	2026-07-06
Author	Antoine Letarte (Atomaste)
Recipient	Cédric Leclerc
Status	For Cédric’s review — comments requested before lock

Changes since previous internal draft:

- *2026-07-06 source sync*: promotes Antoine-confirmed Raspberry Pi 5 host, AK60 spindle path, and ordered Renishaw/RLS BiSS-C encoders into the firmware source/report package.
- *Hardware sync with Cédric’s current BOM*: primary F/T sensor updated from the retired SRI M3703C + M8228S1 path to the selected **Kunwei KWR75B-CAN** integrated 6-axis load cell. The transport section is now CAN-based and the telemetry status channel is generalized to `ft_status`.
- *Toolhead spindle drive source sync*: Zone 2 now treats the selected **CubeMars AK60-6 V1.1 KV80** integrated actuator as the current spindle path, replacing ODrive S1 + M8325s. Firmware-facing work is now AK60 command/telemetry mapping, CAN topology, limits, fault mapping, and safety power-removal validation.
- *Z-axis drive selection update*: Zone 4 now reflects Cédric’s selected **AutomationDirect SV2L-210B / SV2A-2150** servo package, with remaining work focused on torque/current command scaling, enable/fault wiring, and current/Iq monitor mapping.
- *Open questions refreshed*: replaced stale ODrive and SSI encoder questions with AK60 CAN/telemetry/safety-stop items and ordered Renishaw/RLS BiSS-C encoder frame/pinout items.
- *Formatting*: ToC / figure lists enabled; exec summary expanded; Excalidraw architecture diagram retired, Mermaid is now the sole source of truth for all three diagrams (delivery-scope, control-stack, runtime-split).
- *Structure*: §5.1 Three-Truths duplicate removed (canonical definition in §2.3); schema references cleaned up to logical names; §18.1 v1 scope and §19 acceptance criteria cross-referenced; §21 open-questions list audited.
- *Telemetry*: `z_servo_iq_v` and `z_brake_engaged` channels added to §9.1 and §15.
- *Zero-G / manipulation mode cancelled for MVP and deferred*: tool-side force sensing cannot support contact-force control while the tool is in air, because no load path passes through the tool/blank interface. Powered Zero-G/admittance manipulation is now a secondary-priority future-feature bucket only, not a v1 requirement. The v1 basis is tool-weight compensation during force-control operation plus a mechanical lock / cam-arm-nut removal / manual-removal workflow.

- *Telemetry retention contract added*: raw ≥ 100 Hz capture remains the machine-side audit source, while a host-side post-run routine derives compressed/downsampled telemetry, segment statistics, dwell/work maps, anomaly windows, hashes, and retention status after each run or manual session.
- *Figure readability update*: The tall in-body safety and segment-lifecycle figures are reduced to clear the footer. Figures 4, 5, and 6 now point to Appendix A, where landscape large-format copies are provided for review.
- *Open questions closed*: arm encoder (Orbis), Z force actuator architecture (counterweight+servo), force-actuator feedback method (Iq), table motor retention (Baldor DC kept).
- *Normand operator request preserved*: toolhead spindle rotation direction remains an explicit operator/job setpoint (cw / ccw) logged separately from non-negative spindle RPM; AK60 physical direction mapping is a commissioning item.

2 Fullum Polisher - Machine Control & Firmware Specification v1

2.1 Revision history

Table 2 — Source document revision history.

Date	Revision	Update
2026-07-06	Draft v1.3 - AK60/RPi5/BiSS-C source sync	Promotes Antoine's accepted hardware corrections into the firmware source: Raspberry Pi 5 host, CubeMars AK60-6 V1.1 KV80 spindle path replacing ODrive/M8325s, ordered Renishaw/RLS BiSS-C encoder part numbers, and AK60 safety/CAN commissioning items.

2.2 Purpose and Context

This document is the **implementation target** for the machine-side stack of the Fullum swing-arm polisher. It tells you exactly:

- what code and firmware you need to build,
- what data you receive and what data you emit,
- what you must compute in real time and what you must not invent,
- how to prove that the implementation is aligned with the suite.

The machine is designed to be a complete, standalone polishing system. It operates in manual mode (operator sets parameters directly) and can also execute pre-computed correction programs. Program generation - figuring strategy, metrology analysis, pass planning - is a separate engineering scope and

is **not part of this machine delivery**. The machine simply receives program files and executes them. **Your job is execution, telemetry, and safety.**

“**Read this before everything else — Cédric’s v1 in one paragraph.** Your finish line for v1 is **Normand running the polisher manually from the touchscreen, producing clean synchronized telemetry, with safety enforced.** Nothing more is expected of v1. No program execution, no figuring strategy, no metrology — those belong to later phases. But the data contracts in this document look heavier than v1 strictly needs, and that is **deliberate**. The schemas (controller-job.v1, run-log.v1, manual-session-log.v1), the telemetry channel names, and the host↔Teensy protocol are shaped so that the same wires that carry today’s manual sessions will carry tomorrow’s program execution and multi-pass correction campaigns *without renaming a single field*. Phase 1 manual telemetry is the calibration dataset the entire downstream pipeline (polisher-sim, polisher-post) is built on (§2.5). Renaming a channel or reshaping a schema after that data exists is expensive; carrying a few “unused-in-v1” fields now is free. **Rule of thumb:** if something in this spec looks over-engineered for v1, check §2.5, §2.6, and §5.1 before simplifying — the structure is almost always there for the next phase. When in doubt, ask Antoine. The mental model is: build the v1 surface, but commit to the contracts that survive Phase 2–4 unchanged.”

2.2.1 People

Table 3 — Project roles and responsibilities for the Fullum polisher control implementation.

Role	Person	Responsibility
Machine operator	Normand Fullum	15+ years polishing production. Primary manual-mode user.
Mechanical / electrical / firmware / control	Cédric Leclerc	Design, wiring, firmware, sensor integration. This spec is for you.
Software / system integration	Antoine Letarte (Atomaste)	Program generation tooling, data schemas, system integration.

2.2.2 The Machine

The Fullum polisher is a **swing-arm polisher** used for optical figuring of large mirrors (currently the GigaBIT M1 - 1.2 m class Zerodur primary). The machine has been in production for 15+ years under Normand’s manual operation. This overhaul adds closed-loop force control, synchronized telemetry, and software integration while preserving Normand’s manual workflow.

2.2.3 Read This First - Cédric’s v1 Build Target

For the first implementation pass, build the machine-side stack in this order:

1. **Safe manual operation with telemetry** - Normand can set force, table RPM, spindle RPM, toolhead spindle rotation direction (clockwise/counter-clockwise), and optional force modulation from the touchscreen, with all samples logged.

2. **Tool-weight compensation and safe removal workflow** - normal force control subtracts the known tool/head weight from the KWR75B-CAN Fz reading while the tool is in contact. Tool removal is handled mechanically: stop the machine, lock the swing arm/arc, remove the cam arm nut to free arm rotation, move the arm aside, and remove the tool from the blank by hand.
3. **Host ↔ Teensy setpoint/telemetry protocol** - the host sends setpoints, the Teensy executes the inner loop, and both sides ACK/NACK faults explicitly.
4. **Controller-job intake** - accept only controller-job.v1 packages that pass schema, machine ID, controller version, and capability checks.
5. **Run artifacts** - emit run logs, manual-session logs, telemetry CSVs, event logs, alarms, and hashes in a predictable /data/ layout.

Do **not** implement figuring strategy, metrology interpretation, optimization, calibration logic, or controller-side replanning. Those belong to polisher-sim and polisher-post.

Authoritative contracts for this revision:

- controller-job.v1 — JSON Schema (Draft 2020-12), maintained in the polisher-sim repository under schemas/controller-job.schema.json.
- machine-capabilities.v1 — JSON Schema, same repo, schemas/machine-capabilities.schema.json.
- run-log.v1 — JSON Schema, same repo, schemas/run-log.schema.json.
- This document — machine-side implementation behavior, manual mode, safety, telemetry, and operator workflow.

The schema bundle is distributed with each revision of this spec. Antoine owns the schemas; the file paths above are given relative to the repo root and are not Cédric-local paths.

2.2.4 Normand Manual-Mode Quick Start

Manual mode is the day-one operating mode. The intended shop-floor sequence is:

1. Put the hardware HOA selector in **Auto** so the Teensy and RPi are active.
2. Mechanically set the arm amplitude and center on the machine.
3. Open **Manual Mode** on the touchscreen.
4. Confirm the geometric gate: r_menante, L_menee, R_tool, configured arm amplitude, and configured arm center.
5. Enter force, table RPM, spindle RPM, spindle rotation direction (clockwise/counter-clockwise), and optional modulation.
6. Press **Start**. The machine ramps smoothly, logs telemetry, and enforces software interlocks.
7. Adjust setpoints live as needed. Every change is timestamped.
8. Press **Stop**. The system ramps down and emits a manual-session log plus telemetry CSV.
9. For tool removal or repositioning, use the mechanical safe-removal workflow: stop the machine, confirm zero commanded force, lock the swing arm/arc, remove the cam arm nut to free arm rotation, move the arm aside, and remove the tool from the blank by hand.

Hardware **Hand** mode remains a fallback/maintenance path outside the software loop. Software manual mode is the preferred production and calibration mode because it produces synchronized telemetry.

2.3 Overview and Methodology

This section explains the bigger picture: what the machine overhaul achieves, why the architecture is structured this way, and how the work phases build on each other.

2.3.1 The Problem We Are Solving

Optical figuring is the process of shaping a mirror surface to nanometer-level accuracy. The physics of material removal in polishing follow the **Preston equation**:

$$\text{Removal} \propto \text{Pressure} \times \text{Velocity} \times \text{Time}$$

On a swing-arm polisher, the tool sweeps across the mirror as the table rotates and the arm oscillates back and forth. The amount of material removed at each point on the mirror depends on how much force the tool applies (pressure), how fast the tool moves across that point (velocity from table rotation + arm oscillation), and how long the tool spends over that point (dwell time).

The key insight: if we can precisely control force and measure the tool's position on the mirror, we can predict how much material we remove - and therefore shape the mirror surface intentionally rather than by trial and error.

Today, Normand does this by feel, experience, and periodic measurement with an interferometer. The goal of this overhaul is to make the machine **observable** (synchronized telemetry) and **precisely controllable** (closed-loop force), which opens the door to computer-assisted figuring in the future.

2.3.2 What the Machine Does

The machine is a **complete, standalone polishing system** with two operating modes:

1. **Manual mode** - Normand sets force, RPMs, and other parameters directly through the touchscreen UI. This is how the machine will be used from day one.
1. **Program mode** - The machine loads and executes a pre-computed correction program (controller-job.v1 file). These programs specify force, speed, duration, and modulation parameters for each polishing pass.

Program generation, including figuring strategy, metrology analysis, and pass planning, is not part of this machine delivery. It is a separate engineering scope that requires specialized simulation and optimization tools. The machine architecture is designed so that programs can be provided from any external source in the future. The machine simply receives a program file, validates it, executes it, and exports the results.

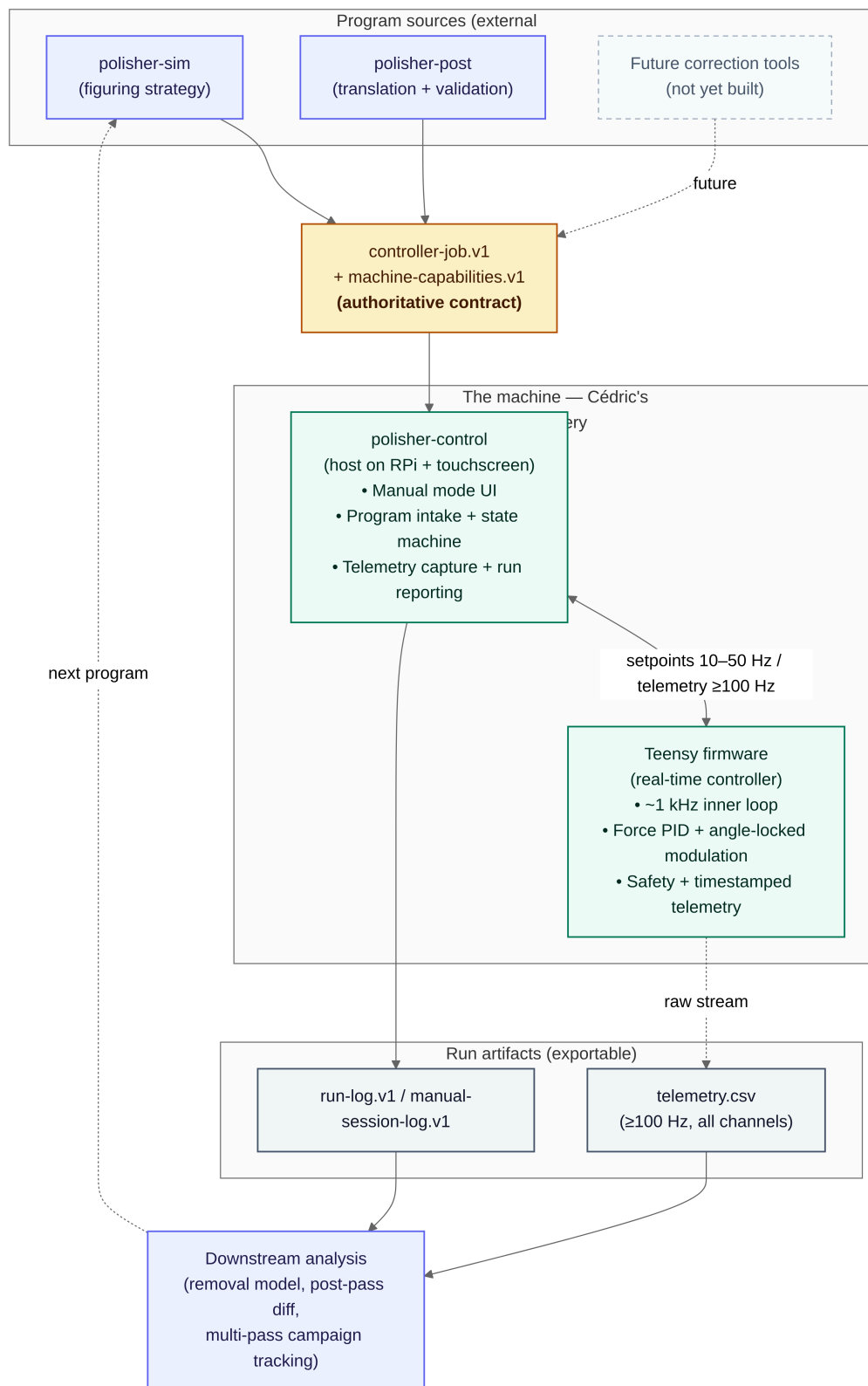


Figure 2 — Machine delivery boundary and data flow. Controller jobs enter the machine; run logs and telemetry leave for downstream analysis. Source: Fullum-Polisher-Delivery-Scope.mmd.

From the controller's perspective: a controller-job.v1 is a self-contained, validated package. The controller does not need network access, does not need to talk to external services in real time, and does not need to understand figuring strategy. It just executes what it's given.

2.3.3 The Three Truths

The system preserves three distinct layers of data, and they are never collapsed:

Table 4 — Planned, commanded, and actual data layers preserved by the controller.

Truth	What It Means	Example
Planned	What a correction program requested	force_n: 50
Commanded	What was actually sent to the machine (after capability checks)	commanded_force_n: 45 (clipped to machine max)
Actual	What physically happened (measured by sensors)	force_n_mean: 44.2 (from F/T sensor via Teensy)

The gap between commanded → actual tells us about control accuracy. If a program was loaded, the gap between planned → actual tells us how well the whole pipeline worked. **This feedback loop is what makes it possible to improve correction programs over time.**

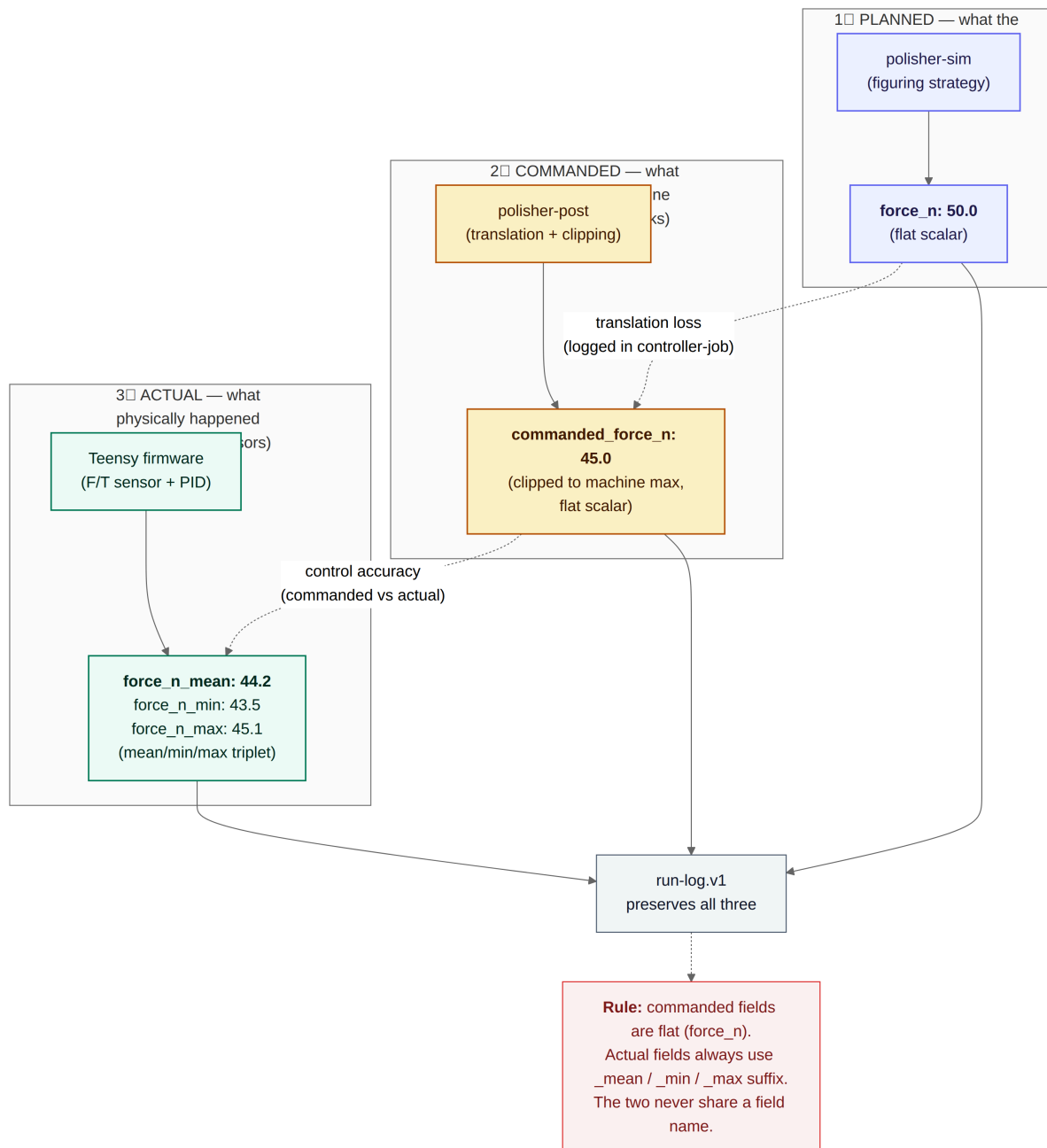


Figure 3 — — The three truths preserved end-to-end. Commanded fields are flat scalars; actual fields always use the `_mean / _min / _max` triplet. The two never share a field name — that structural separation is what makes confusion impossible and what makes Phase 1 manual telemetry directly comparable to Phase 3 program runs. Mermaid source: *Fullum-Polisher-Three-Truths.mmd*.

2.3.4 Force Modulation - Why It Matters

On a swing-arm polisher, the tool naturally removes more material in some zones than others due to the geometry of the sweep pattern. This creates systematic errors on the mirror surface that show up as Zernike aberrations (coma, astigmatism, trefoil, etc.).

Force modulation is the key correction mechanism. By varying the polishing force as a function of the table angle, we can intentionally remove more material in specific azimuthal zones:

$$F(t) = F_base + A \times \sin(m \times \theta_table(t) + \varphi)$$

- **m = 1** corrects coma (once-per-revolution pattern)
- **m = 2** corrects astigmatism (twice-per-revolution)
- **m = 3** corrects trefoil (three times per revolution)

The Teensy reads the table encoder angle in real time and modulates the force actuator output at 1 kHz. The amplitude (A) and phase (φ) come from the correction program - the firmware just executes the formula.

This is why synchronized, timestamped telemetry from the table encoder and force sensor is so critical. Without precise angle-force correlation, the modulation is useless.

2.3.5 The Journey: Manual First, Programs Later

The overhaul follows a deliberate multi-phase approach. We do **not** start with program execution. We start by making the machine observable and controllable, then progressively enable program-driven operation.

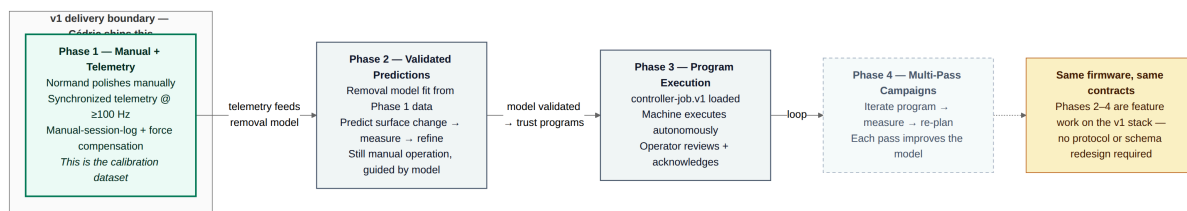


Figure 4 — Phase roadmap. *Cédric ships Phase 1 (manual + telemetry). Phases 2–4 reuse the same firmware, same hardware, same data contracts — they are feature work on top of v1, not redesigns. Mermaid source: Fullum-Polisher-Phase-Roadmap.mmd.*

2.3.5.1 Phase 1 - Manual Operation + Telemetry (where we start)

Normand polishes manually via the touchscreen UI
 ↓
 Machine records synchronized telemetry:
 force, table angle, arm angle, RPM, derived oscillation amplitude/center, timestamps
 ↓
 After the run: Normand measures the mirror with interferometer
 ↓
 Telemetry + interferograms are used to characterize the machine:
 "when X force was applied at Y angle for Z seconds,
 this much material was removed at this location"
 ↓
 This builds a removal model for this specific machine,
 tool, and mirror material

This is the most important phase. We learn the actual removal function of this specific machine. Normand's 15 years of experience get encoded into telemetry data that calibrates the model.

What Cédric's system must deliver for Phase 1:

- Clean, timestamped telemetry (force, angles, RPMs) at ≥ 100 Hz
- Manual mode that Normand can use comfortably
- Session logs that capture what setpoints Normand used and when he changed them

- Reliable, repeatable sensor readings
- Exportable data files (CSV, JSON)

2.3.5.2 Phase 2 - Validated Predictions

Using the removal model, predict: "if you run these parameters for this long, the surface should change by approximately this much"



Normand runs the machine (manual mode, following guidance)



Measure with interferometer



Compare prediction vs reality



Refine the model until predictions are reliable

We keep iterating until the removal model consistently predicts surface changes within acceptable uncertainty. Only then do we start trusting full correction programs.

2.3.5.3 Phase 3 - Program Execution

Measure mirror with interferometer



Correction program computed from surface error + removal model



Program file (controller-job.v1) loaded on machine



Normand reviews parameters, acknowledges, starts the run



Machine executes the program autonomously



Export run data, measure again, compute next correction



Iterate until mirror meets specification

This is where the machine runs programs end-to-end. But it only works because Phase 1 and 2 gave us a calibrated, trusted model of this specific machine.

2.3.5.4 Phase 4 - Multi-Pass Campaigns

Multi-pass correction campaign planned



Execute pass → measure → update model → compute next program



Repeat until mirror meets spec



Each iteration improves the model

2.3.6 What This Means for the Machine Control Stack

Cédric's work is **load-bearing for every phase**:

- **Phase 1** needs manual mode + telemetry. This is the minimum viable deliverable.

- **Phase 2** needs the same, plus reliable commanded-vs-actual reporting.
- **Phase 3** needs job intake, state machine, segment execution — the full spec in this document.
- **Phase 4** is just Phase 3 in a loop.

The quality of the telemetry data from Phase 1 directly determines how fast the machine's removal model can be calibrated. Bad timestamps, noisy force data, or missing angle readings will slow everything down. **Clean, synchronized telemetry is the single most valuable output of the machine control stack.**

The v1 build target is Phase 1. Cédric is not asked to deliver Phases 2–4; he is asked to deliver Phase 1 *in a way that does not block Phases 2–4*. Concretely:

- The host↔Teensy protocol must carry segment-style frames (SEGMENT_START, SETPOINT, SEGMENT_DONE) even though v1 only ever uses the manual subset of those frames. Otherwise Phase 3 forces a protocol redesign while Cédric's firmware is already deployed.
- Telemetry channels must already use the `_mean / _min / _max` triplet convention for actuals (§5.1) and stay flat for commanded, even though v1 has no controller-job to compare against. Otherwise Phase 1 telemetry is unusable as a Phase 2 baseline.
- The state machine must already include JOB_LOADED, READY, RUNNING, and segment lifecycle states, even if v1 production use mainly exercises IDLE → MANUAL → IDLE. Otherwise the gate that makes program execution safe (the mandatory operator-acknowledge transition) has to be retrofitted later.

In short: **v1 deliverables = manual force control + safe tool-removal procedure + telemetry.**
v1 contracts = the full suite contracts. Build the v1 surface against the full contracts and the next three phases are firmware feature work on top of the same hardware and the same data shape, not a redesign.

2.3.7 How Manual and Auto Mode Fit Together

The machine has two levels of “manual”:

Hardware manual (HOA switch → Hand):

- Direct hardware control path, independent of the Teensy and RPi
- Teensy and RPi are not in the control loop
- No telemetry, no logging, no software safety
- Fallback path — the machine works even if all software is dead
- Useful for initial hardware testing, jog moves, maintenance

Software manual (HOA switch → Auto, no program loaded):

- Operator enters setpoints via touchscreen UI
- Teensy is active: force PID, safety interlocks, telemetry
- Everything is logged - this telemetry is the foundation for building the removal model
- Operator can adjust setpoints live while the machine runs
- **This is the primary mode for Phase 1 and 2**

Program execution (HOA switch → Auto, program loaded):

- polisher-control executes a controller-job program
- Setpoints come from the program file, not the operator
- Operator can still pause/resume/abort
- This is Phase 3 and beyond

The key insight: **software manual mode is not a lesser mode - it is the primary data acquisition mode** that makes program-driven operation possible. Every manual session with good telemetry makes future correction programs more accurate. Treat it as a first-class feature, not an afterthought.

2.4 Hardware Basis and Integration Ownership

This section defines the current machine-side hardware basis without over-prescribing every final component choice. The intent is to give Cédric a clear execution target while leaving room for detailed design, supplier comparison, and final selection work.

Use the status terms below consistently:

- **Fixed:** architecture or component direction already considered locked for this revision
- **Preferred, confirm in detailed design:** current best direction, but final selection remains part of integration work
- **Open, Cédric to finalize:** function is defined, but detailed hardware choice is intentionally left open

2.4.1 Sensors

2.4.1.1 Primary process force sensor

- **Current basis:** 6-axis load cell / F/T sensor, **Kunwei KWR75B-CAN** (BOM line: Fz 200 N) with integrated decoupling electronics.
- **Status:** Fixed (Cédric BOM 2026-05-18).
- **Required capability / notes:** Provides Fz as the controlled variable plus the remaining five channels for diagnostics. The selected KWR75B-CAN outputs decoupled engineering-unit force/moment data directly over CAN, with no external DAQ or signal-conditioner box. 200 N Fz full-scale gives margin over the 0-80 N operating range; the software safety limit (§11) and counterweight apparent-mass ceiling remain the binding upper constraints.
- **Interface expectation: Dedicated CAN link** into Teensy 4.1 through an isolated CAN transceiver (see §9.4).
- **Purpose:** Primary process feedback, with Fz as the controlled variable.

2.4.1.2 Table encoder

- **Current basis:** Absolute rotary encoder, **RLS Artos DHR 162 mm**, 20-bit, P/N DHR162DB20BAAS30DA02, with SAR162 ring P/N SAR162C143AF14WAP00 for the 143 mm table shaft.
- **Status:** Fixed (actual Renishaw/RLS order validated 2026-06-18; source sync 2026-07-06).
- **Required capability / notes:** High angular fidelity, deterministic absolute readout for modulation phase reference. Large-bore OEM ring-kit mount on the existing table shaft. The ordered DB output code is **BiSS-C bidirectional**, replacing the stale SC / SSI assumption.
- **Interface expectation: BiSS-C over RS422** into Teensy via an appropriate RS422 transceiver/interface. Cédric must confirm frame timing, CRC/status-bit handling, and exact wiring from the ordered RLS documentation before firmware lock.
- **Purpose:** Table angle, derived table RPM, phase reference for force modulation.

2.4.1.3 Arm encoder

- **Current basis:** Absolute rotary encoder on the swing-arm pivot, **RLS Orbis BR10**, 14-bit on-axis (stub-shaft magnet), P/N BR10DCD14B12DH00 + magnet BM120A190A1ABA00.
- **Status:** Fixed (actual Renishaw/RLS order validated 2026-06-18; source sync 2026-07-06).
- **Required capability / notes:** 230 μm radial uncertainty at the encoder bore, sufficient for logging, kinematic linearization, and post-hoc characterization of oscillation amplitude/center (see §3.3, §9.1). The ordered DC output code is **BiSS-C over RS422 at 5 V**, replacing the stale SC / SSI assumption. The earlier Artos DHR option-B path is closed.
- **Interface expectation:** **BiSS-C over RS422** into Teensy; confirm frame format, CRC/status handling, and transceiver/pinout before firmware lock.
- **Purpose:** Swing-arm oscillation position and amplitude, kinematic linearization reference (§9.5), derived amplitude/center statistics (§9.1).

2.4.1.4 Spindle speed feedback

- **Current basis:** **CubeMars AK60-6 V1.1 KV80 integrated actuator**, using the actuator's internal sensing/drive telemetry for commutation and actual speed feedback.
- **Status:** Fixed hardware direction (Antoine confirmation 2026-07-05; CEDRIC-PROP-20260609-001 accepted).
- **Required capability / notes:** Must provide stable actual spindle RPM for logging and fault checks. The AK60 closes the motor loop locally; the machine controller receives actual RPM, direction state when available, current/torque/thermal telemetry when available, and fault state. Firmware must treat ODrive S1 + M8325s as obsolete for this machine.
- **Interface expectation:** AK60 runtime command/telemetry via CAN is the current working basis. Final CAN topology, IDs, frame rates, timeouts, and fault handling must be frozen before firmware lock; keep the primary KWR75B-CAN force path isolated unless Cédric explicitly proves a shared-bus design is safe and deterministic.
- **Purpose:** Spindle commutation, speed monitoring, and tool rotation telemetry.

2.4.1.5 Force actuator feedback

- **Current basis:** Servomotor quadrature current (I_q) or equivalent torque/current monitor exported from the Z-axis servo drive (see §3.2).
- **Status:** Fixed architecture; monitor mapping to confirm during drive setup.
- **Required capability / notes:** Used as a linear proxy for belt tension / applied differential force. Complements the primary F/T sensor for observability and commissioning diagnostics. Not a replacement for the KWR75B-CAN sensor; the force PID still closes on F_z .
- **Interface expectation:** Preferred: 0-10 V analog into a Teensy ADC input. If the selected drive exposes current digitally instead, update the telemetry channel mapping before commissioning.
- **Purpose:** Force actuator observability, tuning support, and diagnostics during normal manual/job force control.

2.4.2 Actuators

2.4.2.1 Force actuator — Z axis

- **Current basis:** Counterweight-biased swing-arm head driven by the selected **AutomationDirect SureServo2 1 kW low-inertia servo motor with brake**, P/N SV2L-210B, direct-drive (1:1) in torque/current mode, routed through a **GT2/GT3 Twin Power double-sided belt** via two idler

pulleys. A passive counterweight cancels the static mass of the head (arm + spindle + tool) down to a maximum apparent mass of 80 N. **Control closes on Fz from the KWR75B-CAN sensor (§3.1); the counterweight cancels gravity and the servo modulates only the residual differential force.** See §3.8 for the full Z-axis chain.

- **Status:** Fixed component selection (Cédric BOM 2026-05-18); control mapping to confirm.
- **Required capability / notes:** Servo chosen for transparency (zero stiction), high bandwidth, and the 1:1 direct-drive anchoring that keeps vibration sources away from the tool. Idler bearings must be ZZ-shielded, light-oil lubricated; 2RS greased bearings are proscribed because they filter the servo's micro-corrections. The motor's brake is mandatory safety hardware (see §3.5), engages on E-stop or power loss, locks the shaft, and prevents tool drop.
- **Drive expectation:** **AutomationDirect SV2A-2150** servo drive, configured for torque/current control or the closest supported real-time force-actuator mode. Preferred command is analog torque/current setpoint from the Teensy with current/Iq feedback returned to telemetry.
- **Purpose:** Controls normal force on the tool pressing against the mirror. Maximum achievable crush force is intrinsically limited to the 80 N apparent-mass budget; this is a deliberate safety feature, not a software limit.

2.4.2.2 Table motor

- **Current basis:** Existing table drive architecture, currently understood as Baldor DC motor + reduction train + DC variator path.
- **Status:** Preferred, confirm in detailed design.
- **Required capability / notes:** Existing machine architecture is the starting point. Cédric should verify actual installed hardware, ratios, and usable control interface on the machine.
- **Drive expectation:** Speed command path from controller, isolated as required.
- **Purpose:** Table rotation.

2.4.2.3 Spindle motor

- **Current basis:** **CubeMars AK60-6 V1.1 KV80 integrated actuator** as the selected toolhead spindle system. This replaces the ODrive S1 + M8325s path and earlier Mosrac U7636 / separate-drive assumptions.
- **Status:** Fixed hardware direction (Antoine confirmation 2026-07-05; CEDRIC-PROP-20260609-001 accepted).
- **Required capability / notes:** From the controller's perspective: an AK60 spindle actuator that accepts speed/enable commands, accepts explicit clockwise/counter-clockwise direction selection, closes the motor loop locally, and exposes actual RPM plus fault state. Direction is represented separately from RPM; RPM remains a non-negative magnitude. Preserve sizing caveats: the AK60-6 KV80 path is acceptable for the documented normal polishing torque basis and transient aggressive cases, but sustained high-torque duty, gearbox ripple/backlash, wet/slurry protection, output interface, and thermal/current telemetry must be validated before commissioning sign-off.
- **Drive expectation:** AK60 integrated drive/controller. Runtime command/telemetry via CAN is the current working basis; if Cédric selects a different supported AK60 interface, update this spec and the machine capability profile before firmware lock.
- **Safety expectation:** AK60 has no safety-rated STO in the current plan. Firmware CAN stop/disable commands are normal-control actions only; the safety model must remove AK60 power through the safety-controlled post-contactor power path and bench-test coast-down/fault behavior.
- **Purpose:** Tool rotation.

2.4.2.4 Arm oscillation motor

- **Current basis:** AC induction motor driven through a VFD (mechanical selection owned by the mechanical spec; current basis on that side = Baldor EM3554T). The motor turns at a rotational speed; the oscillation of the swing arm is produced by the mechanical cam linkage, not by reversing or modulating the motor — this is how a classical swing-arm polisher works. In v1 the RPM is a single operator-set value for a session; in later iterations the same setpoint is expected to be varied over time (per-segment and eventually intra-segment) as part of the sim/program pipeline. The command path is the same in both cases.
- **Status:** Preferred, confirm in detailed design.
- **Required capability / notes:** From the controller's perspective, the arm drive is a VFD-fronted AC motor with a simple linear speed setpoint. No reversal, no oscillation waveform command — just RPM.
- **Drive expectation:** 0–10 V isolated speed reference from the Teensy DAC; run/stop and fault discretes through the HOA + safety chain.
- **Purpose:** Swing-arm oscillation drive (via cam).

2.4.2.5 Arm oscillation VFD

- **Current basis:** **AutomationDirect GS21-22P0** (current working basis).
- **Status:** Preferred, confirm in detailed design.
- **Required capability / notes:** VFD accepts an isolated 0–10 V analog setpoint from the Teensy and exposes run/stop + fault I/O into the HOA and safety chain. Frequency range 0–60 Hz.
- **Drive expectation:** 0–10 V isolated analog speed reference from the Teensy; run/stop and fault I/O into the HOA and safety chain.
- **Purpose:** Swing-arm motor drive.

2.4.2.6 CubeMars AK60-6 V1.1 Firmware-Facing Facts

The toolhead/spindle mechanical design is finalized outside this firmware document. The firmware/control scope is the electrical interface, configuration, command path, telemetry path, safety-stop assumptions, and fault handling for the selected AK60 integrated actuator path.

Firmware-facing AK60 assumptions to verify before commissioning:

- Selected actuator path: **CubeMars AK60-6 V1.1 KV80** integrated actuator.
- Low-speed polishing preference: KV80 remains preferred over KV140 for the torque-dominated 10–120 RPM use case.
- Approximate torque envelope to preserve in design notes: about **3 N·m rated / 9 N·m peak**. This is suitable for the documented normal polishing basis and acceptable for short aggressive transients, but not enough margin to treat the aggressive case as sustained duty without thermal/current validation.
- Current generated electrical basis: AK60 powered from a **24 VDC supply downstream of the safety-controlled post-contactor bus**. Confirm final supply rail, fuse/breaker sizing, cable/connector ratings, and voltage/current telemetry before commissioning.
- Runtime interface: CAN is the working basis. Freeze AK60 command IDs, telemetry IDs, frame rates, stale-frame timeout, startup/fault behavior, and whether AK60 shares a bus with KWR75B or uses a separate Teensy CAN channel.
- Direction: cw / ccw remains an explicit protocol enum separate from non-negative RPM. Cédric must bench-map the enum to the actual physical rotation convention.

- Environmental/mechanical checks: gearbox backlash/ripple, non-hollow packaging, wet/slurry protection, output interface, thermal path, and cable strain relief must be verified outside firmware before final commissioning.

Firmware obligations:

- Store the AK60 commissioning parameters used with the machine software release or machine-capability profile.
- Configure and log velocity limits, current/torque limits, thermal limits, acceleration/deceleration/ramp policy, spindle direction mapping, and fault thresholds.
- Expose AK60 fault state to the host and map faults into the normal `DRIVE_FAULT` / spindle fault handling path.
- Log at least actual spindle RPM; strongly recommended AK60 telemetry channels are listed in §9.1.
- Treat CAN stop/disable commands as normal-control actions, not safety-rated stop functions. E-stop safety must rely on the hardwired safety chain and power-removal path, then be bench-tested.

2.4.3 Cam Mechanism

- **Type:** Mechanical oscillating pivot — sets swing-arm sweep pattern (amplitude and center of oscillation).
- **Adjustment: Manual, mechanical** — the operator sets amplitude and center by hand on the machine before each run, reading the setting off a physical rule / sticker on the mechanism.
- **Instrumentation (v1): None on the cam itself.** No cam encoder, no amplitude/offset sensor, no actuator. The mechanical setting is not directly measured.
- **Operator workflow:** The operator reads the amplitude and center values off the mechanical scale and enters them into the UI as configuration (same pattern as `r_menante`, `L_menee`, `R_tool`). See §13.4 and the geometric gate in §13.4.1.
- **Post-hoc characterization:** The effective amplitude and center of the oscillation are derived by the firmware from the **arm encoder** statistics (peaks, troughs, and midpoints over the last N cycles) and exposed as telemetry channels `arm_amplitude_deg_derived` / `arm_center_deg_derived` (§9.1). This is characterization, not a control loop.
- **Controller role:** Accept the operator-entered amplitude/center as configuration, validate against machine capability, and compare the arm-encoder-derived values against them once the regime is established (§12). Never command the cam.
- **Future scope (post-v1):** A positioning device to mechanically set amplitude and center (and, plausibly, an actuator) is anticipated. That revision will reintroduce a real cam encoder and, eventually, active cam actuation. Tracked in §21.

2.4.4 Controller Hardware

2.4.4.1 Real-time controller

- **Current basis: Teensy 4.1.**
- **Status:** Fixed for this revision.
- **Role:** Inner loop, force PID, sensor reads, modulation, safety, telemetry emission.

2.4.4.2 Host computer

- **Current basis: Raspberry Pi 5 + touchscreen.**
- **Status:** Fixed for this revision.

- **Role:** State machine, job management, UI, telemetry logging, operator workflow.

2.4.4.3 Encoder interface hardware

- **Current basis:** RS422 / protocol interface hardware as required by final encoder selection. MAX3490-class transceiver remains a current working basis for the ARTOS path.
- **Status:** Preferred, confirm in detailed design.
- **Role:** Brings table encoder data into the real-time layer.

2.4.4.4 F/T conditioning hardware

- **Current basis:** Integrated in the Kunwei KWR75B-CAN sensor body.
- **Status:** Fixed (Cédric BOM 2026-05-18).
- **Role:** Performs 6-axis decoupling and emits engineering-unit force/moment data directly over CAN; no external DAQ or DSP conditioner box is required.

2.4.4.5 Isolation and analog interface hardware

- **Current basis:** KBSI-240D opto-isolator for the existing DC variator (table) path; additional isolation per final motor control selections.
- **Status:** Preferred, confirm in detailed design.
- **Role:** Safe command interface from controller into legacy power electronics.

2.4.4.6 Force actuator drive

- **Current basis:** **AutomationDirect SV2A-2150** drive for the SV2L-210B Z-axis servo (§3.2), configured for torque/current control or the closest supported real-time force-actuator mode.
- **Status:** Fixed component selection; control mapping to confirm.
- **Role:** Executes the Z-axis torque command path; current/Iq feedback supports actuator observability and commissioning diagnostics.

2.4.5 Safety Hardware

- **E-stop buttons:** NC (normally closed), series-connected.
- **Safety relay:** Pilz PNOZ X1 (Cat 3) or Banner XS26-2 (minimum for MVP).
- **E-stop circuit:** **Hardwired relay, independent of software** — cuts all VFD enable signals and de-energizes the Z servo (releases the NC brake coil → brake engages, locks shaft).
- **Z-axis failsafe brake:** **NC 24 VDC electromagnetic brake** on the Z servomotor shaft. Energized during normal operation (brake released). Power loss or E-stop de-energizes the coil, the brake engages instantly and locks the servo shaft — preventing any uncontrolled tool drop onto the mirror. Interacts with the state machine (§10/§11) as a response action, not a new state.
- **Counterweight failsafe:** Passive — the counterweight sets the maximum “apparent mass” of the head to 80 N. A belt rupture or servo failure cannot produce an impact greater than that, and the flexure-tripod axial compliance ($K_z \approx 8.67 \text{ N/mm}$) absorbs the residual energy. No software action required.
- **Tool removal / arm handling controls:** No powered Zero-G handle is part of v1. Tool removal is a mechanical procedure: software force command at zero, all axes stopped, swing arm/arc mechanically locked, cam arm nut removed to free arm rotation, arm moved aside, tool removed from the blank by hand. Any interlock switches for the lock / cam-arm-nut hardware should be wired into the safety chain or logged as discrete inputs once Cédric finalizes that hardware.
- **Fallback:** Hand mode remains operative even if RPi/Teensy crashes.

2.4.6 Operator Control Panel (HOA - Hand / Off / Auto)

- **Hand (Manual hardware):** Direct hardware control of table RPM, arm RPM, spindle RPM, and the force actuator, without routing through the Teensy. **Bypasses all digital control.** Works even with RPi/Teensy powered off. Input device is part of the controller-side hardware design.
- **Off:** System powered down.
- **Auto:** Teensy assumes control via DAC outputs, overriding the hardware-manual inputs with digital setpoints. This is where the software stack operates.

“**Important:** The HOA switch is a **hardware-level** mode selector. When in Hand mode, the software stack is not in the loop at all. The “Manual Mode” described in §12 of this spec is a **software manual mode within Auto** - the Teensy is active, telemetry is logging, safety interlocks are enforced, but the operator enters setpoints via the touchscreen UI instead of from a pre-planned job. Both the hardware manual workflow (Hand) and the software manual mode (Auto + no job) are valid operational modes.”

2.4.7 Power

Table 5 — Electrical supply rails required by the machine-side control hardware.

Supply	Rating	Feeds
Facility mains	120/240 VAC, 20A breaker	VFDs, DC variator, main power
24 VDC supply	5A	Teensy, sensors, stepper driver, safety relay, signal conditioners

2.4.8 Transmission Summary

Table motor (Baldor DC 1HP, P/N 35P574Z18361, existing)
 → Grove Gear reducer 10:1 (GR8150219-00, existing)
 → 8MGT3 synchronous belt (24T:72T = 3:1, Gates 1200-8MGT3-30)
 → Table (0-40 RPM, 89 N·m continuous, 110 N·m peak)

Z-axis / force actuator chain (counterweight-biased "inverted gravity")
 Head (arm + spindle + tool) ← belt ← counterweight (calibrated to
 head_mass - 80 N apparent)

|
 AutomationDirect SV2L-210B servo, 1 kW, direct-drive 1:1,
 anchored to machine base
 → NC 24 VDC electromagnetic brake on servo shaft (failsafe)
 → GT2/GT3 Twin Power double-sided belt
 → Idler 1 (coude du mât), ZZ light-oil
 → Idler 2 (bout du bras de contrepoids), ZZ light-oil
 → Head suspension
 → Flexure-tripod compliance ($K_z \approx 8.67 \text{ N/mm}$)
 → Tool pressing on mirror

Control: servo operates in torque mode. Counterweight cancels the static head weight; the servo modulates only the residual differential force. Force PID still closes on F_z from the Kunwei KWR75B-CAN sensor — the control scheme is unchanged; only the actuator output path differs from earlier stepper+ballscrew concepts.

Mechanical detail (bearings, shaft design, crank-rocker geometry, reducer mounting, counterweight sizing, idler placement) is out of scope for this firmware/control document — see the mechanical spec owned by Cédric.

2.5 System Boundary

2.5.1 What You Own (In Scope)

Table 6 — Firmware and host-controller responsibilities in Cédric’s implementation scope.

Domain	Responsibility
Teensy firmware	Real-time inner loop, sensor reads, actuator drive, force PID, angle-synchronized modulation, hardware safety
Host controller	Job intake, state machine, segment orchestration, Teensy comms, telemetry persistence, operator workflow, manual mode UI
Sensor integration	Encoder acquisition, F/T sensor reads, RPM measurement
Actuator control	Force actuator (linear, controls normal force on optic), spindle drive, table drive
Safety/interlocks	E-stop, limit detection, watchdog, fault handling, safe shut-down
Telemetry	Raw timestamped data capture, logging, export
Run reporting	Run summaries, event logs, alarm records, commanded-vs-actual
Machine capability descriptor	Static profile of what this hardware can safely do

2.5.2 What You Do NOT Own (Out of Scope)

These belong upstream. **Do not implement, approximate, or work around them.**

Table 7 — Upstream planning and analysis functions excluded from the machine firmware delivery.

Function	Owner
Dwell map generation	External planning tools (not this delivery)
Figuring strategy / pass planning	External planning tools (not this delivery)
Metrology interpretation	External planning tools (not this delivery)
Calibration logic (Preston coefficients)	External planning tools (not this delivery)
Job optimization or correction mode selection	External planning tools (not this delivery)
Validation of planning output against machine capabilities	External (pre-delivery step)
Translation from planning format to controller format	External (pre-delivery step)
Post-run normalization and analysis	External analysis tools
Translation loss tracking	External analysis tools

Key rule: If you catch yourself writing code that decides *what* polishing strategy to use rather than *how* to execute an approved one, stop. That belongs upstream.

2.6 Architecture Context

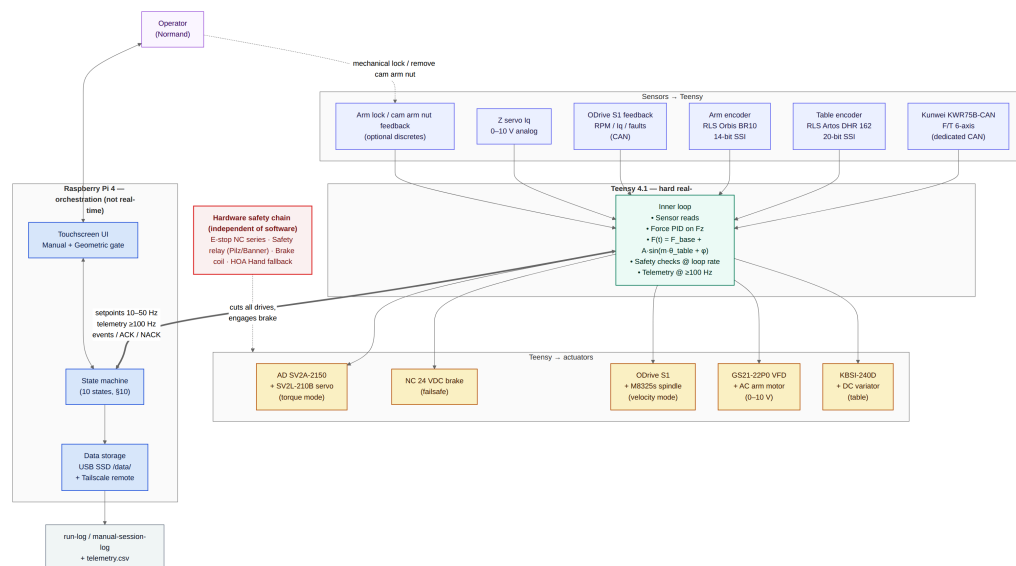


Figure 5 — Full system architecture. Sensors feed the Teensy real-time loop; Teensy drives actuators and exchanges setpoints/telemetry with the host; the host owns state machine, UI, and persistence. The hardware safety chain (E-stop, brake, HOA Hand fallback) runs independently of software and can cut all drives directly. See Appendix A.1 for the full-size landscape copy. Mermaid source: Fullum-Polisher-System-Architecture.mmd. (The older Fullum-Polisher-Control-Stack.* files are deprecated.)

2.6.1 Three Truths — Implementation Constraint

The planned / commanded / actual distinction introduced in §2.3 is a hard structural constraint on the implementation, not just explanatory framing:

- **Commanded** values (segment setpoints, controller-job fields) are flat scalars: force_n, table_rpm, spindle_rpm.
- **Actual** values (run-log, telemetry summaries) always use the _mean / _min / _max triplet suffix: force_n_mean, force_n_min, force_n_max.

Commanded and actual must never share a field name or collapse into a single number at any stage of the pipeline. See §2.3 for the conceptual definition and §8.3 for the run-log schema that enforces it.

2.7 Runtime Split

2.7.1 Host Controller (Python on RPi or PC)

Not real-time. Segment-level orchestration.

Table 8 — Host-controller responsibilities and non-real-time orchestration duties.

Responsibility	Detail
Load controller-job	Parse JSON, validate 4-gate sequence (§7.2)
State machine	10 states (including MANUAL), full transition table per §10.2
Segment orchestration	Send setpoint frames, manage segment timing
Teensy comms	Serial protocol, ACK/NACK, watchdog
Telemetry persistence	Buffer incoming telemetry, write to CSV
Run-log assembly	Build run-log.v1 with commanded vs actual
Operator controls	Pause, resume, abort, acknowledge gate
Event/alarm recording	Timestamped event log, alarm promotion

Update cadence to Teensy: 10-50 Hz (setpoint frames)

2.7.2 Teensy Firmware (C++ on Teensy 4.x)

Hard real-time. This is where physics happens.

Table 9 — Teensy firmware responsibilities executed in the hard-real-time loop.

Responsibility	Detail
Sensor reads	Encoders, F/T sensor, RPM - deterministic
Force PID	Closed-loop force control at loop rate
Force modulation	Angle-synchronized modulation (§9.3) - computed locally
Actuator outputs	Force actuator (linear), spindle, table
Safety enforcement	Limit checks, E-stop, watchdog, fast shutdown
Telemetry emission	Timestamped packets to host
Command intake	Parse setpoint frames, ACK/NACK

Inner loop target: 1 kHz class (stable, documented actual rate)

2.7.3 What the Teensy Computes Locally

These depend on real-time measured state and **must** run in firmware:

- Force PID loop
- $F(t) = F_{\text{base}} + A \times \sin(m \times \theta_{\text{table}}(t) + \varphi)$ - force modulation from live encoder
- Limit checking against safety_limits
- Watchdog timeout (host heartbeat)
- Fast safety shutdown
- Encoder acquisition → instantaneous angle and derived speed

2.7.4 What the Teensy Must NOT Decide

- Whether a polishing strategy is good
- What dwell map to use
- What correction mode to choose
- Whether metrology implies a different next pass
- Silent modification of segment intent beyond safety clipping/rejection

If the firmware clips or rejects a setpoint for safety, it must be explicit: NACK with reason code, logged as an event.

2.8 Input Contract: Controller-Job

2.8.1 What You Receive

You receive a controller-job.v1 JSON file. This is a ready-to-run program - it has already been validated against machine capabilities and translated to machine-safe values before it reaches the machine.

You never see or parse a job.v1 artifact. If someone hands you one, reject it.

2.8.2 Job Intake Validation (4-Gate Sequence)

When loading a controller-job, validate in this exact order:

1. `schema_version == "controller-job.v1"`. On failure: reject as wrong artifact type.
2. JSON Schema validation against the bundled `controller-job.v1` schema (Draft 2020-12, distributed with this spec revision from `polisher-sim/schemas/controller-job.schema.json`). On failure: reject as malformed payload.
3. `machine_id` matches this controller's ID. On failure: reject as wrong machine.
4. `controller_version` matches this controller's supported `polisher-control/*` version range. On failure: reject as version mismatch.

State transition: IDLE → JOB_LOADED only after all 4 gates pass.

Capability cross-check (Gate 4 addendum). Before accepting the job, the host must verify that every capability the job relies on (motion family, requested ranges, modulation harmonics, dither profile, segment duration) is listed in this controller's §15 machine-capability descriptor. If the job asks for anything the controller does not advertise, reject with a specific reason. This keeps older post-processors and newer controllers (or vice versa) from silently drifting apart as the suite evolves.

Unknown-field policy. If a controller-job or segment contains a field this controller does not recognize, the controller **rejects** the job rather than ignoring the field. Strict match is chosen over lenient parsing to prevent the post/sim pipeline from quietly relying on behavior the controller does not actually implement. New fields require a new schema version.

2.8.3 Controller-Job Top-Level Structure

“v1 framing reminder. The v1 controller never receives a real `controller-job.v1` in production — Normand runs the machine in manual mode. The shape below is the Phase 3 contract, exercised in v1 only through schema validation and round-trip tests. The job-intake path must still be implemented for v1 so that the moment a real job exists, no protocol or schema change is required.”

This example mirrors the current `controller-job.v1` schema in the active `polisher-sim` repo. It represents **translated truth**: a post-processed, machine-targeted execution package.

```
{
  "schema_version": "controller-job.v1",
  "controller_job_id": "cjob-2026-07-001",
  "job_id": "job-2026-07-001",
  "machine_id": "fullum-alpha",
  "machine_capabilities_ref": "machine-capabilities/fullum-alpha.v1.json",
  "controller_version": "polisher-control/0.1.0",
  "translated_at": "2026-07-15T09:00:00Z",
  "translated_by": "polisher-post/0.1.0",
  "translation_notes": "No clipping applied. Mechanical arm amplitude and center must be confirmed through the UI geometric gate before execution.",
  "segments": [ ... ],
  "translation_losses": []
}
```

Important: safety limits, telemetry-channel requirements, and supported feature ranges live in the machine-capability profile (§15), not in the controller-job payload. The controller-job can reference the profile used during translation, but the controller still cross-checks against its local profile before execution.

2.8.4 Segment Structure

Each segment is a self-contained, machine-executable command block:

```
{
  "segment_id": "seg-01",
  "source_pass_id": "pass-01",
  "sequence_index": 1,
  "duration_s": 1200,
  "commanded_force_n": 45.0,
  "commanded_table_rpm": 4.0,
  "commanded_spindle_rpm": 40.0,
  "commanded_spindle_direction": "cw",
  "commanded_cam_amplitude_deg": 31.3,
  "commanded_cam_offset_deg": 0.0,
  "force_modulation": {
    "harmonic": 2,
    "amplitude_n": 5.0,
    "phase_deg": 37.2
  },
  "dither_profile": "none",
  "notes": "Single-pass conservative figuring segment"
}
```

Field definitions:

- `segment_id`: string. Unique segment identifier.
- `source_pass_id`: string. Links back to planning pass for traceability.
- `sequence_index`: int. Execution order, 1-based per the current schema.
- `duration_s`: float, seconds. How long to run this segment.
- `commanded_force_n`: float, N. Base force setpoint.
- `commanded_table_rpm`: float, RPM. Table rotation speed.
- `commanded_spindle_rpm`: float, RPM. Spindle rotation speed as a non-negative magnitude.
- `commanded_spindle_direction`: enum, "cw" or "ccw". Toolhead spindle rotation direction, viewed from above the toolhead looking down toward the mirror/tool contact. The AK60 direction mapping is a commissioning item and must be configured so these enum values match physical observed rotation.
- `commanded_cam_amplitude_deg`: float, degrees. Machine-package representation of the mechanical oscillation amplitude. In v1 this is not actuator-commanded; it is confirmed by the operator through the geometric gate and logged as `configured_arm_amplitude_deg` in machine-side/manual records.
- `commanded_cam_offset_deg`: float, degrees. Machine-package representation of the mechanical oscillation center/offset. In v1 this is not actuator-commanded; it is confirmed by the operator through the geometric gate and logged as `configured_arm_center_deg` in machine-side/manual records.
- `force_modulation`: object/null. Angle-synchronized modulation parameters.
- `dither_profile`: string. "none" or a machine-approved profile such as "default".
- `notes`: string. Optional translation or operator context.

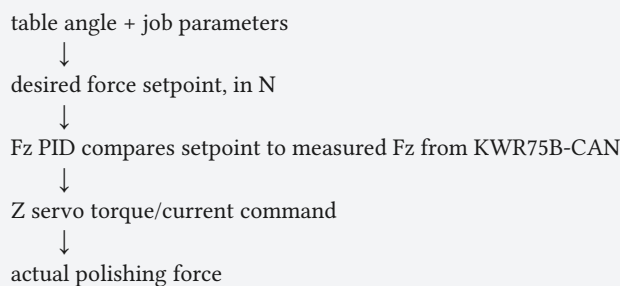
“Note on cam mechanism (v1): On this version of the Fullum polisher, the oscillation amplitude and center are set mechanically by the operator and are **not instrumented** - there is no cam encoder. The controller-job schema currently uses the legacy names `commanded_cam_amplitude_deg` and `commanded_cam_offset_deg` because those are the active suite contract fields. For this machine revision, interpret them as requested/expected mechanical settings, not as firmware-actuated cam commands. The operator confirms the actual mechanical setting through the mandatory geometric gate (§13.4.1), and the machine-side logs preserve the confirmed values as `configured_arm_amplitude_deg` /

configured_arm_center_deg. At runtime the controller compares the arm-encoder-derived amplitude and center (arm_amplitude_deg_derived / arm_center_deg_derived, §9.1) against these configured values once the regime is established and warns on divergence (§12). A future schema revision should rename these fields or split requested-vs-configured cam settings explicitly.”

2.8.5 Force Modulation Block

Force is still controlled by the Fz PID loop using the KWR75B-CAN force sensor. The modulation equation does **not** replace the PID and does **not** directly command the Z servo. It only computes the force setpoint that the PID should maintain at that instant.

The control stack is:



With no modulation, the setpoint is constant. For example, if the operator or job asks for 8 N, the PID continuously compares:

$$\text{error} = 8 \text{ N} - \text{measured_Fz}$$

If the sensor reads 6.5 N, the PID commands more Z-axis force. If the sensor reads 9 N, the PID backs off. That is the normal constant-force mode.

With modulation enabled, only the PID target changes slowly as the table rotates. Example: if a segment asks for 8 N base force with a 2 N modulation amplitude, the PID target moves smoothly between 6 N and 10 N while the table turns. The PID still closes the loop on measured Fz.

- F_base: base force from the segment, from commanded_force_n, in N.
- A: modulation amplitude, from force_modulation.amplitude_n, in N. This is the amount added above and below the base force.
- m: harmonic number, from force_modulation.harmonic. m = 1 means one force wave per table revolution; m = 2 means two waves per revolution; m = 3 means three waves per revolution.
- theta_table(t): live table angle from the encoder, converted to radians.
- phi: phase offset, from force_modulation.phase_deg, converted to radians. This rotates the force pattern around the mirror so the high-force zone occurs at the intended angular location.

The setpoint-generator implementation form is:

$$F_{\text{setpoint}}(t) = F_{\text{base}} + A \times \sin(m \times \theta_{\text{table}}(t) + \phi)$$

The host sends parameters only. The Teensy reads the table encoder continuously, evaluates this setpoint formula at 1 kHz, then feeds $F_{\text{setpoint}}(t)$ into the normal Fz PID loop. This keeps the desired force pattern locked to table angle even if the table RPM drifts slightly. The modulation itself is slow

(table RPM 0.5-5 → period 12-120 seconds), so the constraint is phase accuracy relative to table angle, not force-command bandwidth.

In job/program mode, this block is mainly used when the upstream planning / dwell-map tooling wants an angle-locked force pattern. In normal manual mode, modulation defaults off ($A = 0$), so the formula collapses to $F_{\text{setpoint}} = F_{\text{base}}$. If the operator sets 8 N manually with modulation off, the PID simply maintains 8 N from the force sensor.

2.9 Output Contract: Run Artifacts

2.9.1 Required Outputs

Every run must produce these artifacts, regardless of how it ended:

Table 10 — Required run artifacts emitted by the machine controller.

Artifact	Format	Description
Run-log	run-log.v1 JSON	Complete execution record for job-mode runs
Raw telemetry file	CSV, or equivalent raw capture format	Full-rate timestamped sensor data captured during execution
Event log	embedded in run-log	All state transitions, operator actions, alarms
Post-run processing status	JSON fields in manifest / log	Whether derived summaries, compressed telemetry, and retention classification were produced successfully

Raw telemetry is the audit source for the machine. It is not the preferred permanent analysis format. The host must preserve the raw full-rate capture first, then a host-side post-run routine derives smaller, analysis-ready artifacts from it (§17.6). If post-processing fails, the raw telemetry and run-log still make the run valid, but the failure must be recorded in the manifest and visible in the operator/engineering status view.

2.9.2 Run-Log Top-Level Structure

“**v1 framing reminder.** v1 emits manual-session-log.v1 (§13.6) in production. run-log.v1 shown below is the Phase 3 artifact, but its commanded / actual shape — flat commanded values, `_mean` / `_min` / `_max` triplets for actuals — is **the same shape** the manual-session-log uses for `actual_summary`. Building both against this convention is what makes Phase 1 telemetry directly comparable to future Phase 3 runs without reprocessing.”

```
{
  "schema_version": "run-log.v1",
  "run_id": "run-2026-07-001",
  "job_id": "job-2026-07-001",
  "controller_job_id": "cjob-2026-07-001",
  "controller_version": "polisher-control/0.1.0",
  "machine_id": "fullum-alpha",
  "started_at": "2026-07-15T09:30:00Z",
  "ended_at": "2026-07-15T09:55:12Z",
  "result_state": "completed",
  "segments": [ ... ],
  "commanded_summary": {
    "force_n": 45.0,
    "table_rpm": 4.0,
    "spindle_rpm": 40.0
  },
  "actual_summary": {
    "force_n_mean": 44.7,
    "force_n_min": 43.8,
    "force_n_max": 45.3,
    "table_rpm_mean": 3.98,
    "table_rpm_min": 3.95,
    "table_rpm_max": 4.02,
    "spindle_rpm_mean": 39.8,
    "spindle_rpm_min": 39.4,
    "spindle_rpm_max": 40.2
  },
  "alarms": [],
  "events": [ ... ],
  "telemetry_ref": "telemetry/run-2026-07-001.csv",
  "operator_notes": "",
  "hashes": {
    "telemetry_sha256": "d4e5f6...",
    "run_log_sha256": "a1b2c3..."
  }
}
```

2.9.3 Executed Segment Structure

Each segment in the run-log records both commanded and actual values:

```
{
  "segment_id": "seg-01",
  "source_pass_id": "pass-01",
  "sequence_index": 0,
  "requested_duration_s": 1200,
  "actual_duration_s": 1199.8,
  "commanded": {
    "force_n": 45.0,
    "table_rpm": 4.0,
    "spindle_rpm": 40.0
  },
  "configured_arm_oscillation": {
    "amplitude_deg": 31.3,
    "center_deg": 0.0,

```

```

"source": "operator_entered_v1_no_cam_encoder"
},
"actual": {
  "force_n_mean": 44.7,
  "force_n_min": 43.8,
  "force_n_max": 45.3,
  "table_rpm_mean": 3.98,
  "table_rpm_min": 3.95,
  "table_rpm_max": 4.02,
  "spindle_rpm_mean": 39.8,
  "spindle_rpm_min": 39.4,
  "spindle_rpm_max": 40.2,
  "arm_amplitude_deg_derived_mean": 31.2,
  "arm_center_deg_derived_mean": 0.1
},
"state": "completed",
"pause_windows": [],
"notes": ""
}

```

Convention: commanded block has flat values. actual block has `_mean`, `_min`, `_max` triplets. This makes confusion structurally impossible.

2.9.4 Result States

Table 11 — Canonical run result states recorded in run and manual-session logs.

State	Meaning
completed	All segments ran to completion, no issues
completed_with_pause	Completed but had operator pause(s)
aborted	Operator or host requested abort, controlled shutdown
faulted	Safety fault triggered, emergency stop

2.9.5 Event Record Format

```

{
  "timestamp": "2026-07-15T09:30:00.123Z",
  "event_code": "STATE_TRANSITION",
  "severity": "INFO",
  "from_state": "READY",
  "to_state": "RUNNING",
  "trigger": "start",
  "segment_id": "seg-01",
  "details": "operator confirmed, run started"
}

```

2.9.6 Alarm Promotion

Events with severity: "ALARM" are automatically promoted to the top-level alarms[] array in the run-log. Alarms include:

```
{
  "timestamp": "2026-07-15T09:42:15.456Z",
  "alarm_code": "FORCE_OVER_LIMIT",
  "severity": "critical",
  "measured_value": 85.2,
  "limit_value": 80.0,
  "action_taken": "EMERGENCY_STOP",
  "segment_id": "seg-01"
}
```

2.10 Telemetry Specification

“v1 framing reminder. The channel table below is **the full Phase 1–4 telemetry surface**, not a v1-only subset. Every Phase 3+ analysis tool — removal-model calibration, post-pass diff, multi-pass campaign tracking — keys off these exact channel names. Channels that look unused in v1 (e.g. force_setpoint_n in manual mode where setpoints come from the UI, or arm_amplitude_deg_derived which is post-hoc characterization) are still required because their absence breaks Phase 2 model calibration retroactively against Phase 1 data.”

2.10.1 Channel Table

2.10.1.1 Core Channels (Required - must log at ≥ 100 Hz)

- timestamp_us: uint64, μ s, Teensy clock. Monotonic machine time; all channels sync to this.
- table_angle_deg: float, deg, encoder. Current table rotation angle (0-360 continuous).
- arm_angle_deg: float, deg, encoder. Current swing-arm angle.
- fz_n: float, N, F/T sensor. Vertical normal force, primary process signal.
- mx: float, N·m, F/T sensor. Moment about X.
- my: float, N·m, F/T sensor. Moment about Y.
- mz: float, N·m, F/T sensor. Moment about Z.
- spindle_rpm_actual: float, RPM, tachometer/encoder. Measured spindle speed magnitude. Direction is logged separately when available.
- table_rpm_actual: float, RPM, encoder-derived. Measured table speed.
- arm_amplitude_deg_derived: float, deg, Teensy from arm encoder statistics. Half the peak-to-peak arm encoder range over the last N complete oscillation cycles. See derivation window below.
- arm_center_deg_derived: float, deg, Teensy from arm encoder statistics. Midpoint between the running peak and trough of the arm encoder over the last N complete oscillation cycles. See derivation window below.
- machine_state: enum, state machine. Current controller state.

Derivation window for arm_amplitude_deg_derived / arm_center_deg_derived:

- The firmware maintains a rolling window over the **last N = 5 complete oscillation cycles** of the arm encoder (cycle boundaries detected from zero-crossings of the arm angular velocity, i.e. at each peak / trough).
- On every new cycle boundary, amplitude is recomputed as $(\text{peak_max} - \text{trough_min}) / 2$ over the window and center as $(\text{peak_max} + \text{trough_min}) / 2$.
- Until 5 complete cycles have been observed since the last `SEGMENT_START` / `MANUAL_START`, both channels emit NaN (regime not yet established). After that, they are emitted at the telemetry rate, holding the last-computed value between cycle updates.
- Window size N is a firmware constant in v1; it may become a capability parameter later.

These channels are characterization, not control: the firmware does not act on them in the inner loop. The host consumes them in §12 to compare against the operator-entered `configured_arm_amplitude_deg` / `configured_arm_center_deg` and warn on divergence.

2.10.1.2 Extended Channels (Strongly Recommended)

- `fx_n`: float, N, F/T sensor. Lateral force X.
- `fy_n`: float, N, F/T sensor. Lateral force Y.
- `ft_status`: uint32 bitfield, F/T sensor. Raw KWR75B-CAN status word or mapped validity bitfield (validity, saturation, fault bits).
- `z_servo_iq_v`: float, V, Z servo drive analog out. Quadrature current (Iq) from the Z-axis AC servo drive, linear proxy for belt tension / applied differential force (§3.1, §3.2).
- `z_brake_engaged`: bool, safety chain. True when the Z-axis NC electromagnetic brake is engaged (de-energized coil). Expected false during normal RUNNING / MANUAL, true in IDLE / FAULTED / E-stop.
- `spindle_drive_state`: enum/string, AK60 integrated actuator. AK60 state or mapped controller state for the spindle drive.
- `spindle_drive_error`: uint32/string, bitfield/code, AK60 integrated actuator. Raw or mapped AK60 error / fault code.
- `spindle_bus_voltage_v`: float, V, AK60 integrated actuator. DC bus / supply voltage reported by the actuator if exposed.
- `spindle_iq_a`: float, A, AK60 integrated actuator. Measured torque/current channel, if exposed over the selected interface.
- `spindle_motor_temp_c`: float, degC, AK60 integrated actuator. Motor/drive temperature telemetry, if exposed over the selected interface.
- `arm_angle_linearized_deg`: float, deg, Teensy from encoder + geometry. Crank-rocker linearized arm angle (§9.5); NaN if geometry not validated.
- `force_setpoint_n`: float, N, setpoint generator. Current target force after modulation; this is the input to the Fz PID, not the PID actuator output.
- `table_rpm_setpoint`: float, RPM, host command. Commanded table speed.
- `spindle_rpm_setpoint`: float, RPM, host command. Commanded spindle speed magnitude.
- `spindle_direction_setpoint`: enum, cw / ccw, host command. Requested toolhead spindle direction.
- `spindle_direction_actual`: enum, cw / ccw, AK60/firmware-derived. Actual observed or actuator-reported spindle direction after direction mapping.
- `force_actuator_cmd`: float, V or mA, DAC output. Raw actuator command signal.
- `estop_active`: bool, hardware. E-stop circuit state.
- `interlock_state`: uint16 bitfield, multiple. Packed interlock status bits.
- `mode`: enum, controller. manual / active.

2.10.2 Telemetry Principles

1. **Single clock source.** All channels timestamped from one monotonic Teensy clock. No mixing of host time and firmware time in the same telemetry stream.
2. **Raw values.** Log raw measured values, not only filtered/averaged. Filtering happens upstream.
3. **Commanded AND actual.** Both setpoint channels and measured channels must be present for force, table RPM, spindle RPM.
4. **Validity flags.** If a sensor read fails or saturates, mark it - do not silently substitute a default. Use NaN, a sentinel value, or a parallel validity column.
5. **No gaps without explanation.** If samples are missed, the gap must be detectable from timestamps.

2.10.3 Telemetry CSV Format

Header fields, in order:

```
timestamp_us
table_angle_deg, arm_angle_deg
fx_n, fy_n, fz_n, mx, my, mz, ft_status
spindle_rpm_actual, table_rpm_actual
arm_amplitude_deg_derived, arm_center_deg_derived
machine_state, force_setpoint_n
```

Example rows:

```
1234567000,182.5,127.3,0.8,-0.4,44.9,0.12,-0.08,0.02,0,39.8,4.01,NaN,NaN,RUNNING,45.0
1234568000,182.8,127.3,0.7,-0.5,44.7,0.11,-0.09,0.02,0,39.9,4.00,31.2,0.1,RUNNING,45.0
```

One row per sample. Header row uses exact channel names above. No units in data rows.

The CSV file is the first-write raw capture and compatibility format. It is intentionally simple so that commissioning tools can inspect it with pandas, numpy, or spreadsheet software. Long-term history should not depend on preserving every normal full-rate CSV row forever. Permanent engineering history is produced by the host-side post-run routine: compressed raw archive when retained, down-sampled trace, per-segment statistics, dwell/work maps, anomaly windows, and hashes (§17.6).

2.10.4 KWR75B-CAN Transport — Dedicated CAN Link

Status (§3 taxonomy): sensor choice is **Fixed** for this revision. Exact CAN frame identifiers, byte order, scaling, and status-bit definitions are **Preferred, confirm in detailed design** against the KWR75B-CAN package and manual, tracked as an open item in §21.4.

The primary F/T sensor is the Kunwei KWR75B-CAN. Its electronics are integrated inside the sensor body, so there is no external DAQ or DSP conditioner. The sensor publishes already-decoupled force/moment samples over a **dedicated CAN link** into the Teensy 4.1.

Topology

Table 12 — Dedicated KWR75B-CAN sensor-link topology and protocol assumptions.

Item	Detail
Link	Dedicated CAN pair from KWR75B-CAN to a Teensy 4.1 CAN interface through an isolated CAN transceiver
Bus	One sensor on the bus for v1 unless Cédric explicitly adds more CAN devices
Bit rate	Vendor default or 1 Mbit/s working basis, to be confirmed from the Kunwei package
Protocol	Vendor CAN frame map carrying decoupled Fx, Fy, Fz, Mx, My, Mz and status
Network dependency	None. This link does not share the RPi LAN or Tailscale link

Frame format (canonical internal representation after CAN decode)

```
struct FT_LoadSample {
    float Fx, Fy, Fz; // N
    float Mx, My, Mz; // N·m
    uint32_t status; // mapped KWR75B-CAN validity/fault flags
};
```

The KWR75B-CAN performs the 6-axis decoupling internally. The Teensy receives already-decoupled, engineering-unit values and feeds Fz directly into the force PID. Status bits must be inspected on every frame; sensor faults or saturation raise FT_SENSOR_INVALID per §11.1.

Receive loop (reference skeleton)

```
#include <FlexCAN_T4.h>

FlexCAN_T4<CAN1, RX_SIZE_256, TX_SIZE_16> ftCan;
FT_LoadSample currentLoad;

void setup() {
    ftCan.begin();
    ftCan.setBaudRate(1000000); // confirm against Kunwei settings
}

void loop() {
    CAN_message_t msg;
    while (ftCan.read(msg)) {
        decodeKwr75bFrame(msg, currentLoad); // vendor frame map
        // Feed currentLoad.Fz into the force PID
        applyForceControl(currentLoad.Fz);
    }
}
```

Requirements

- Document the actual measured CAN sample arrival rate (target: matches or exceeds the force-loop requirement; log the measured value at commissioning).

- Treat stale frames as a fault: if no valid F/T frame arrives within a configurable window, raise FT_SENSOR_INVALID. Default timeout is 5 ms until the vendor sample rate is confirmed.
- Log the raw or mapped F/T status word as ft_status so post-run analysis can distinguish sensor-side faults from control-side anomalies.
- The exact CAN frame map, byte order, scaling, and status bits are sensor-vendor-defined and must be cross-checked against the KWR75B-CAN manual before integration.

2.10.5 Arm Kinematic Linearization (Firmware Obligation)

The swing-arm oscillation has a non-linear angular velocity profile relative to the drive motor (the mechanism geometry itself is defined in the mechanical spec). Because Preston-equation removal depends on actual tool velocity across the optic, the controller must linearize arm position before any dwell or modulation computation that uses arm angle.

Inputs the firmware consumes (names only — definitions live with the mechanical spec):

Table 13 — Mechanical geometry inputs consumed by the firmware linearization model.

Symbol	Role for the firmware
r_menante	Drives the mapping from motor angle to arm angle
L_menee	Drives the leverage / angular-zero offset of the arm
R_tool	Tool radial offset, used by the dwell / velocity model

These values are collected and validated by the host UI (§13.4.1) and passed to the Teensy at job/manual start. They are **configuration**, not telemetry.

Firmware obligations:

- Apply the linearization at inner-loop rate using the live arm encoder reading and the configured geometry.
- Expose both arm_angle_deg (raw encoder) and arm_angle_linearized_deg (post-linearization) in telemetry.
- Refuse SEGMENT_START / MANUAL_START with GEOMETRY_NOT_VALIDATED if the configured values are missing or flagged stale.

The host UI owns collection and validation; the Teensy owns runtime use. Nothing in this document defines the mechanism itself.

2.11 State Machine

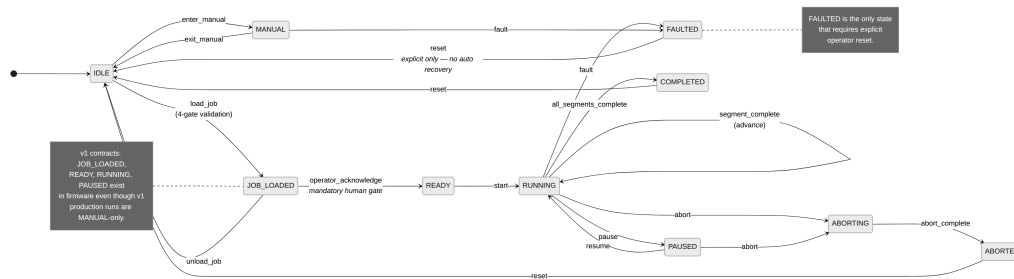


Figure 6 — — Controller state machine. 11 states; FAULTED only exits via explicit operator reset. Note that JOB_LOADED, READY, RUNNING, PAUSED, and segment-related transitions exist in v1 firmware even though production v1 runs are MANUAL-only. Preserving these states is what allows Phase 3 program execution to drop in without rework. See Appendix A.2 for the full-size landscape copy. Mermaid source: Fullum-Polisher-State-Machine.mmd.

2.11.1 States

- IDLE: No job loaded, machine safe, ready for job intake or manual mode.
- JOB_LOADED: Job validated and loaded, awaiting operator acknowledge.
- READY: Operator acknowledged, machine armed, awaiting start.
- RUNNING: Executing segments, force active, telemetry flowing.
- PAUSED: Operator-requested pause, actuators safe, clock stopped for segment.
- ABORTING: Controlled shutdown in progress, retracting Z, stopping spindle.
- COMPLETED: All segments finished successfully.
- ABORTED: Abort sequence completed.
- FAULTED: Safety fault, machine stopped, requires explicit reset.
- MANUAL: Manual operation - operator controls setpoints directly, no job loaded.

2.11.2 Transition Table

- IDLE + load_job $\rightarrow$ JOB_LOADED: after 4-gate validation passes.
- IDLE + enter_manual $\rightarrow$ MANUAL: no job needed; operator enters setpoints directly.
- JOB_LOADED + operator_acknowledge $\rightarrow$ READY: **mandatory human gate**; no auto-bypass.
- JOB_LOADED + unload_job $\rightarrow$ IDLE: cancel before starting.
- READY + start $\rightarrow$ RUNNING: actuators engage, telemetry starts.
- RUNNING + pause $\rightarrow$ PAUSED: ramp force to zero, record pause_window start.
- RUNNING + segment_complete $\rightarrow$ RUNNING: advance to next segment.
- RUNNING + all_segments_complete $\rightarrow$ COMPLETED: normal finish.
- RUNNING + abort $\rightarrow$ ABORTING: controlled shutdown sequence.
- RUNNING + fault $\rightarrow$ FAULTED: safety fault, immediate safe state.
- PAUSED + resume $\rightarrow$ RUNNING: record pause_window end, resume segment.
- PAUSED + abort $\rightarrow$ ABORTING: abort from paused state.
- MANUAL + exit_manual $\rightarrow$ IDLE: ramp down, stop, return to idle.
- MANUAL + fault $\rightarrow$ FAULTED: safety fault during manual operation.
- ABORTING + abort_complete $\rightarrow$ ABORTED: shutdown sequence finished.
- COMPLETED + reset $\rightarrow$ IDLE: ready for next job.
- ABORTED + reset $\rightarrow$ IDLE: ready for next job.
- FAULTED + reset $\rightarrow$ IDLE: **only exit from FAULTED**; explicit operator action required.

2.11.3 State Machine Rules

1. **Illegal transitions are rejected and logged.** Raise `IllegalTransitionError` with `from_state`, `attempted_trigger`, and `timestamp`. Never silently ignore.
 2. **Every transition emits an event** with `timestamp`, `from_state`, `to_state`, `trigger`, and optional notes.
 3. **Operator acknowledge is mandatory.** The `JOB_LOADED` → `READY` transition requires a physical or explicit operator input. **Never auto-acknowledge.** This is the human gate that confirms the physical setup (lap seated, optic centered, guards closed).
 4. **FAULTED can only exit via explicit reset.** No automatic recovery from faults.
 5. **Result state at end of run is unambiguous.** The run-log `result_state` maps directly from the terminal machine state.
-

2.12 Safety, Interlocks, and Fault Handling

2.12.1 Interlocks and Fault Responses

The following interlocks are mandatory for v1. Each one must emit a timestamped event and be visible in the run-log or manual-session-log as applicable. The table was intentionally converted to a list so safety behavior is unambiguous in PDF export.

Hard-stop faults - no automatic resume:

- **E-stop active** (`ESTOP_ACTIVATED`): hardware circuit opens. Immediate all-stop, force actuator command to zero, Z brake engages. Reset required.
- **Force over limit** (`FORCE_OVER_LIMIT`): $fz_n \setminus > \text{ safety_limits.force_max_n }$. Emergency stop, force actuator command to zero. Reset required.
- **Encoder invalid** (`ENCODER_LOST`): no encoder pulse for >50 ms during commanded motion. Emergency stop. Reset required.
- **Drive fault** (`DRIVE_FAULT`): drive error signal active. Emergency stop. Reset required.
- **F/T sensor invalid** (`FT_SENSOR_INVALID`): sensor fault flag, reading outside physical range, or invalid KWR75B-CAN status. Emergency stop. Reset required.
- **F/T frame stale** (`FT_SENSOR_INVALID`): no valid KWR75B-CAN frame within the configurable window, default 5 ms until vendor rate is confirmed, during `RUNNING` or `MANUAL`. Emergency stop, force actuator command to zero. Reset required.

Recoverable pauses / warnings:

- **Force under limit** (`FORCE_UNDER_LIMIT`): $fz_n \setminus < \text{ safety_limits.force_min_n }$ for >2 s during `RUNNING`. Warning, then controlled pause. Resume allowed after operator check.
- **Spindle RPM deviation** (`SPINDLE_RPM_DEVIATION`): $|\text{actual} - \text{commanded}| \setminus > 20\%$ for >3 s. Warning, then controlled pause. Resume allowed after check.
- **Table RPM deviation** (`TABLE_RPM_DEVIATION`): $|\text{actual} - \text{commanded}| \setminus > 20\%$ for >3 s. Warning, then controlled pause. Resume allowed after check.
- **Host comms timeout** (`HOST_COMMS_TIMEOUT`): no valid host frame for >500 ms. Controlled pause first; abort after 5 s if comms do not recover. Resume allowed only if comms recover inside the 5 s window.

Refused transitions / normal operating blocks:

- **Geometry not validated** (GEOMETRY_NOT_VALIDATED): no active geometry record, or stale flag set on `r_menante`, `L_menee`, `R_tool`, configured arm amplitude, or configured arm center. Refuse transition into RUNNING or MANUAL. Resume allowed after UI gate re-passed.
- **Arm lock / cam arm nut not in operating configuration** (ARM_HANDLING_INTERLOCK): if lock / cam-arm-nut feedback is installed, refuse software-controlled motion unless the arm is in the correct operating configuration. The mechanical tool-removal workflow remains an operator procedure outside powered motion.

2.12.2 Safety Enforcement Split

Table 14 — Safety enforcement layers and their expected response times.

Layer	Responsibility	Response Time
Teensy firmware	E-stop, force limits, encoder loss, drive faults, watchdog	<1 ms (within loop iteration)
Host controller	Comms timeout, RPM deviation, segment duration, state machine integrity	<100 ms
Hardware	E-stop circuit (hardwired, independent of software)	Immediate

Principle: Hardware safety is independent of software. Software safety is layered: firmware catches fast faults, host catches slow faults. Neither alone is sufficient.

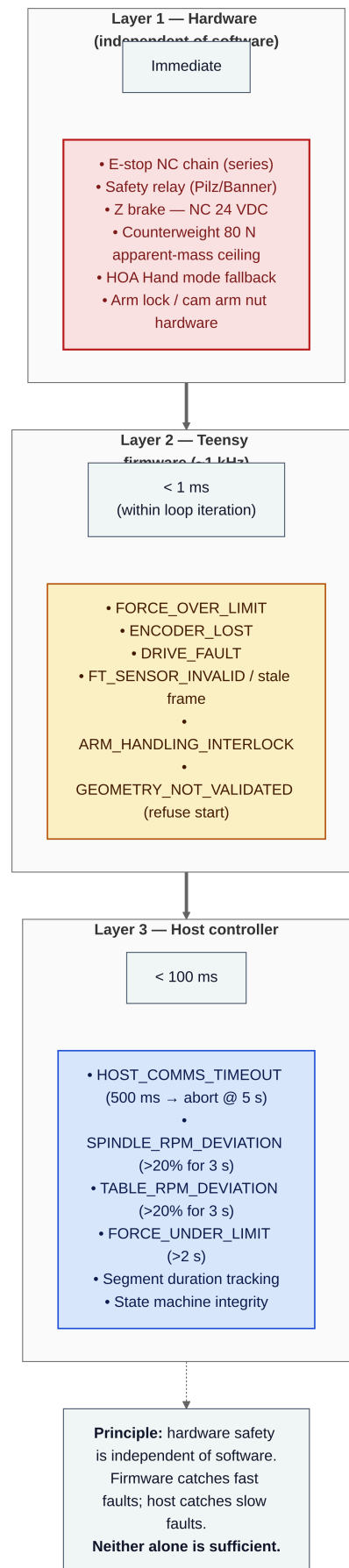


Figure 7 — — Three-layer safety hierarchy. See Appendix A.3 for a readable landscape copy. Mermaid source: Fullum-Polisher-Safety-Hierarchy.mmd.

2.12.3 Watchdog / Heartbeat

- Host sends heartbeat frame to Teensy at ≥ 10 Hz
- Teensy expects heartbeat within 500 ms window
- On timeout: Teensy enters controlled pause, raises HOST_COMMS_TIMEOUT
- If comms not restored within 5 seconds: Teensy initiates abort sequence autonomously
- Host monitors Teensy telemetry stream - if no telemetry for 500 ms, host declares firmware fault

2.12.4 Startup Safety

1. On power-up, machine is in IDLE - all actuators off, force actuator at zero
 2. No motion possible without load_job $\rightarrow$ operator_acknowledge $\rightarrow$ start sequence
 3. Any transition to RUNNING requires all interlocks clear
 4. E-stop must be explicitly released before any arming
-

2.13 Segment Execution Lifecycle

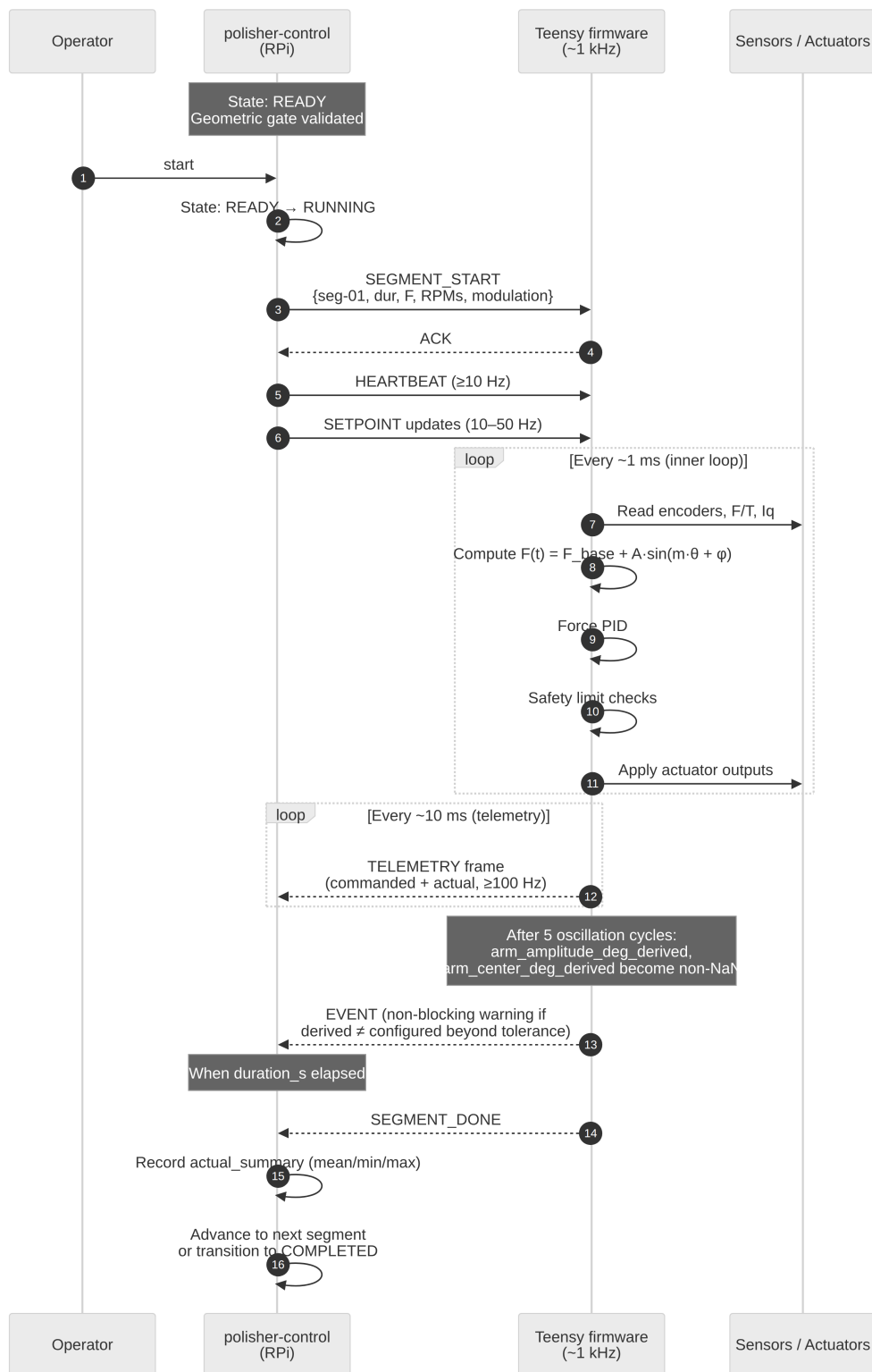


Figure 8 — Segment execution sequence. Full-page layout in the PDF. The host orchestrates at segment granularity; the Teensy runs the inner loop at 1 kHz. Mermaid source: Fullum-Polisher-Segment-Lifecycle.mmd.

This is the step-by-step sequence for executing one segment:

1. Host reads next segment from controller-job
2. Host validates segment setpoints against safety_limits
3. Host verifies oscillation kinematics once regime is established:
 - Wait until the Teensy has published non-NaN values for `arm\_amplitude\_deg\_derived` and `arm\_center\_deg\_derived` (i.e. 5 complete oscillation cycles since SEGMENT_START, per §9.1).
 - Compare derived values against the operator-entered `configured\_arm\_amplitude\_deg` / `configured\_arm\_center\_deg`.
 - Warn operator (non-blocking event) if $|\text{derived} - \text{configured}|$ exceeds a configurable tolerance on either amplitude or center.
 - This check characterizes the mechanical setting against what the operator entered; it does NOT gate segment start, since the derived values are only available after the regime is established.
4. Host sends SEGMENT_START frame to Teensy:
 - segment_id
 - duration_s
 - commanded_force_n
 - commanded_table_rpm
 - commanded_spindle_rpm
 - force_modulation params (if any)
5. Teensy ACKs SEGMENT_START
6. Host begins sending SETPOINT frames at 10-50 Hz
7. Teensy runs inner loop at ~1 kHz:
 - Read sensors (table encoder, arm encoder, F/T, RPM)
 - Compute modulated force: $F(t) = F_{\text{base}} + A \times \sin(m \times \theta + \phi)$
 - Execute force PID
 - Apply actuator outputs
 - Check safety limits
 - Emit TELEMETRY frames at ≥ 100 Hz
8. Host captures telemetry, checks for alarms
9. When duration_s elapsed:
 - Teensy sends SEGMENT_DONE frame
 - Host records segment actual summary (_mean, _min, _max)
 - Host advances to next segment or completes run

2.13.1 Pause During Segment

1. Operator or host sends PAUSE command
2. Host transitions: RUNNING $\rightarrow$ PAUSED
3. Host sends PAUSE frame to Teensy
4. Teensy: ramp down force to zero, maintain spindle/table at safe idle
5. Teensy: continue telemetry (so pause duration is recorded)
6. Record pause_window: { "paused_at": "<ISO timestamp>" }
7. On RESUME:
 - Host transitions: PAUSED $\rightarrow$ RUNNING
 - Host sends RESUME frame
 - Teensy: ramp force back to setpoint
 - Record: { "paused_at": "...", "resumed_at": "..." }
 - Segment timer resumes (pause time excluded from polishing time)

2.13.2 Abort Sequence

1. ABORT command received (operator, host, or auto-triggered)
2. Host transitions: RUNNING/PAUSED → ABORTING
3. Host sends ABORT frame to Teensy
4. Teensy:
 - Ramp force actuator to zero
 - Stop spindle
 - Stop table
 - Send ABORT_COMPLETE frame
5. Host transitions: ABORTING → ABORTED
6. Run-log emitted with result_state: "aborted"
7. Partial segment data preserved in run-log (actual_duration_s < requested_duration_s)

2.14 Manual Operation Mode

2.14.1 Purpose

Manual mode is for **Normand and operators** to run the polisher without a pre-planned job. This is the primary mode during:

- initial machine commissioning and tuning,
- lap conditioning and break-in,
- exploratory polishing (learning the machine behavior),
- quick touch-up runs where a full planning cycle is unnecessary,
- hardware testing and sensor verification.

Manual mode lets the operator set force, RPMs, and other parameters directly through the UI and adjust them live while the machine runs.

2.14.2 How It Works

1. Machine is in IDLE (no job loaded)
2. Operator selects "Manual Mode" on the UI
3. Host transitions: IDLE → MANUAL
4. Operator enters setpoints via UI:
 - Force (N)
 - Table RPM
 - Spindle RPM
 - Spindle rotation direction (`cw` / `ccw`)
 - Optional: force modulation parameters (harmonic, amplitude, phase)

Note: oscillation amplitude and center are set mechanically by the operator on the machine and then entered into the UI as configuration (no cam encoder in v1) - see §13.4 and §13.4.1.
5. Operator presses "Start" - actuators engage, telemetry begins
6. Operator can adjust setpoints LIVE - changes take effect immediately
7. Operator presses "Stop" - controlled ramp-down
8. Host transitions: MANUAL → IDLE
9. Manual-session log emitted

2.14.3 Manual Mode vs Job Mode

Table 15 — Operational differences between planned job execution and touchscreen manual mode.

Aspect	Job Mode (RUNNING)	Manual Mode (MANUAL)
Source of setpoints	controller-job.v1 package	Operator UI in real time
Duration	Defined per segment	Open-ended - runs until operator stops
Setpoint changes	Only between segments	Live - operator adjusts any time
Force modulation	Parameters from job package, typically from upstream planning / dwell-map tooling	Defaults off; operator can optionally enable/disable/adjust live
Telemetry logging	Always	Always
Safety/interlocks	Full enforcement	Full enforcement - identical to job mode
Run-log output	run-log.v1 with segments	manual-session-log (see §13.6)
Planning traceability	job_id → cjob_id → run_id	None - no upstream job reference

2.14.4 What the Operator Controls

The manual mode UI must expose these parameters with **live adjustment**:

Manual mode's baseline behavior is constant-force PID. With modulation off, the force target is simply the operator-entered force value. The modulation controls below are optional live controls; leaving modulation off is the normal simple path.

2.14.4.1 Actuated Parameters (operator commands)

Table 16 — Manual-mode setpoints and defaults exposed on the operator touchscreen.

Parameter	Unit	Range (from machine capabilities)	Default
Force	N	0 – 80	0 (no contact)
Table RPM	RPM	0 – 5.0	0 (stopped)
Spindle RPM	RPM	0 – 80	0 (stopped)
Spindle direction	enum	cw / ccw	cw
Force modulation	on/off	-	off
Mod. harmonic	-	1, 2, 3	2
Mod. amplitude	N	0 – 20	0
Mod. phase	deg	0 – 360	0

2.14.4.2 Operator-Entered Oscillation Configuration (no sensor in v1)

- `configured_arm_amplitude_deg`: degrees, range 0-45 clamped against machine capability. Amplitude the operator set mechanically on the cam, **read off the physical scale / sticker** and entered here. Subject to the geometric gate (§13.4.1).
- `configured_arm_center_deg`: degrees, range -30 to +30 clamped against machine capability. Center of oscillation the operator set mechanically on the cam, **read off the physical scale / sticker** and entered here. Subject to the geometric gate (§13.4.1).

These are configuration, not live setpoints — changing them mid-run is not meaningful because the cam is mechanical. The UI flags them stale and re-runs the gate whenever the operator reports a mechanical adjustment.

2.14.4.3 Read-Only Displays (from encoders — not actuated)

- Table angle: degrees, from the table encoder. Live table rotation angle.
- Arm angle: degrees, from the arm encoder. Live swing-arm angle.
- `arm_amplitude_deg_derived`: degrees, from Teensy arm-encoder statistics (§9.1). Derived oscillation amplitude once the regime is established. Compared against `configured_arm_amplitude_deg` to warn on divergence. NaN until 5 complete cycles have elapsed.
- `arm_center_deg_derived`: degrees, from Teensy arm-encoder statistics (§9.1). Derived oscillation center once the regime is established. Compared against `configured_arm_center_deg` to warn on divergence. NaN until 5 complete cycles have elapsed.

The UI should **surface** the derived values next to the configured ones so the operator can see, a few cycles in, whether the mechanical setting matches what they entered.

All values are clamped to machine capability limits before reaching the Teensy. If the operator enters a value outside the safe range, the UI rejects it or clamps it and shows a warning. The firmware also enforces limits as a second layer.

2.14.4.4 Mandatory Geometric Parameter Gate

Five mechanical parameters are **set by hand on the machine** and cannot be inferred from any sensor in v1:

- The three kinematic linkage parameters from §9.5: `r_menante`, `L_menee`, `R_tool`.
- The two oscillation parameters from §3.3 / §13.4: `configured_arm_amplitude_deg`, `configured_arm_center_deg`.

Together these define the kinematic relationship between motor angle, arm angle, oscillation envelope, and tool position, and they feed directly into the Preston-equation integrity of every run (manual or job). In v1 there is no cam instrumentation, so the amplitude and center values depend entirely on the operator reading the physical scale correctly — which is why they sit behind the same gate as the linkage parameters.

The UI must enforce the following, uniformly across all five parameters:

1. **Blocking validation pop-up.** Before any transition into RUNNING (job mode) or MANUAL (software manual mode), the UI presents a modal dialog showing the five currently stored values and requires the operator to confirm each one. The dialog cannot be dismissed without explicit confirmation; there is no “skip” path.
2. **Re-validation on mechanical change.** Any time the operator (or service personnel) reports a mechanical adjustment — including any change to the cam amplitude or center — the stored values are flagged stale and the gate must be re-passed before the machine can run.

3. **Persistence with timestamp and operator.** Each validated set of parameters is persisted with { r_menante, L_menee, R_tool, configured_arm_amplitude_deg, configured_arm_center_deg, validated_at, operator } and included in every run-log and manual-session-log.
4. **Firmware-side refusal.** If the host attempts to send SEGMENT_START or MANUAL_START without an active, non-stale geometry record covering all five parameters, the Teensy NACKs with GEOMETRY_NOT_VALIDATED and the host surfaces the gate again. This is a defense-in-depth second layer beyond the UI pop-up.

Rationale: a mechanical change that is not reported to the UI silently invalidates all post-run metrological correlation. Making this a hard gate protects the Phase-1 telemetry dataset that everything else depends on.

2.14.5 Manual Mode State Rules

1. **Cannot enter manual mode with a job loaded.** enter_manual is only valid from IDLE. If a job is loaded, unload it first.
2. **Cannot load a job while in manual mode.** load_job is only valid from IDLE.
3. **Safety is identical.** All interlocks, E-stop, force limits, watchdog - everything in §11 applies equally in manual mode. A fault transitions MANUAL → FAULTED.
4. **Telemetry always runs.** Same channels, same rates as job mode. Manual sessions produce usable telemetry for upstream analysis.
5. **Setpoint changes are logged.** Every operator setpoint change, including spindle direction changes, emits an event with the old and new values, timestamped.
6. **No automatic duration.** The machine runs until the operator stops it. There is no timer.
7. **Ramp behavior.** When the operator changes force or RPM, the firmware ramps smoothly to the new value (not step changes). Ramp rate should be configurable but have safe defaults.

2.14.6 Manual Session Log

Even manual runs produce a log so that telemetry is traceable and can be optionally fed back into the suite for calibration or analysis.

```
{
  "schema_version": "manual-session-log.v1",
  "session_id": "manual-2026-07-15-001",
  "machine_id": "fullum-alpha",
  "controller_version": "polisher-control/0.1.0",
  "operator": "Normand",
  "started_at": "2026-07-15T10:15:00Z",
  "ended_at": "2026-07-15T10:42:30Z",
  "result_state": "completed",
  "arm_oscillation_setting": {
    "configured_amplitude_deg": 8.0,
    "configured_center_deg": 0.0,
    "derived_amplitude_deg_mean": 8.1,
    "derived_center_deg_mean": 0.2,
    "note": "Configured values entered by operator from the physical scale on the cam (no cam encoder in v1). Derived values computed by firmware from arm encoder statistics once regime established (§9.1)."
  },
  "setpoint_history": [
    {
      "timestamp": "2026-07-15T10:15:00Z",
```

```

    "force_n": 30.0,
    "table_rpm": 3.0,
    "spindle_rpm": 40.0,
    "spindle_direction": "cw",
    "force_modulation": null
  },
  {
    "timestamp": "2026-07-15T10:22:15Z",
    "force_n": 40.0,
    "table_rpm": 3.0,
    "spindle_rpm": 40.0,
    "spindle_direction": "cw",
    "force_modulation": {
      "harmonic": 2,
      "amplitude_n": 5.0,
      "phase_deg": 0.0
    }
  }
],
"actual_summary": {
  "force_n_mean": 35.2,
  "force_n_min": 29.5,
  "force_n_max": 40.8,
  "table_rpm_mean": 2.99,
  "spindle_rpm_mean": 39.8,
  "total_polishing_time_s": 1650
},
"alarms": [],
"events": [ ... ],
"telemetry_ref": "telemetry/manual-2026-07-15-001.csv",
"operator_notes": "Lap break-in, started at 30N, increased to 40N after 7min",
"hashes": {
  "telemetry_sha256": "abc123..."
}
}

```

Key differences from run-log.v1:

- session_id instead of run_id
- No job_id or controller_job_id (no upstream job)
- setpoint_history[] instead of segments[] - records every operator change
- operator field - who was at the controls
- total_polishing_time_s - total time with force engaged

2.14.7 Manual Session Telemetry

Same CSV format as job mode (§9.3). Same channels. Same timestamps. The only difference is that force_setpoint_n may change mid-stream as the operator adjusts, and machine_state will show MANUAL instead of RUNNING.

This means external analysis tools can optionally ingest manual session telemetry - same parser, just a different log schema wrapper.

2.14.8 Host ↔ Teensy in Manual Mode

In manual mode, the host sends SETPOINT frames (not SEGMENT_START) directly:

Table 17 — Host-to-Teensy command messages used during software manual mode.

Message	When	Content
MANUAL_START	Operator presses Start	Initial setpoints (force, RPMs, spindle direction, modulation) + configured oscillation amplitude/center (from the geometric gate, §13.4.1)
SETPOINT	Operator adjusts any value	Updated setpoint fields
MANUAL_STOP	Operator presses Stop	Controlled ramp-down

The Teensy responds identically to job mode:

- ACK / NACK on every command
- TELEMETRY at ≥ 100 Hz
- EVENT on alarms/faults

The Teensy does not need to know whether it is in manual or job mode. From the firmware’s perspective, it receives setpoints and executes them. The distinction is a host-side concept only. This keeps the firmware simple.

2.14.9 Tool-Weight Compensation and Tool Removal Workflow

Powered Zero-G manipulation is **cancelled for the MVP / v1 delivery** and deferred to the secondary-priority future-feature bucket. It should not be implemented unless the core manual-control, telemetry, safety, and commissioning scope is already stable and Antoine explicitly re-opens it later.

The KWR75B-CAN sensor is mounted on the tool side and measures the force path through the tool/blank interface. When the tool is in the air, that load path is open, so the contact-force reading trends toward zero apart from bias and fixture effects. That makes it the wrong signal for supporting or controlling the head during free-space manipulation. Z-servo current/Iq remains useful for actuator diagnostics, but it is not a robust substitute for a real hand-guided support loop. v1 therefore does not include a ZERO_G state, dead-man Zero-G handle, sensorless admittance mode, or Zero-G session log. Treat any future powered manipulation mode as a separate design study, not as a hidden v1 requirement.

2.14.9.1 Force compensation during normal contact

Tool/head weight compensation belongs in the normal force-control chain, not in a manipulation mode:

1. The machine stores a commissioned tool/head offset, `fz_tool_weight_offset_n`, for the installed tool configuration.
2. The controller keeps raw KWR75B force available for diagnostics as `fz_raw_n`.
3. The force PID operates on compensated contact force:

```
fz_contact_n = fz_raw_n - fz_tool_weight_offset_n
```

1. `force_setpoint_n` is compared against `fz_contact_n`, so the operator/program commands polishing contact force, not tool dead weight.
2. If the tool is not in contact with the blank, `fz_contact_n` is not a valid support/manipulation signal. The firmware must not infer free-space support force from it.

The sign convention and exact offset value are commissioning items. Once locked, they must be written into the machine capability/configuration record and repeated in every manual-session/run log.

2.14.9.2 Safe tool-removal sequence

For tool removal, lap inspection, or moving the head away from the blank, use the mechanical workflow:

1. Stop the active manual/job operation.
2. Command force to zero and stop table, spindle, and arm drive.
3. Confirm the compensated force is near zero and the machine is in IDLE or stopped MANUAL.
4. Engage and verify the swing arm/arc mechanical lock.
5. Remove the cam arm nut so the arm rotation is freed.
6. Move the arm aside by hand using the mechanical handling path.
7. Remove the tool from the blank by hand.
8. Reinstall/seat the tool, re-engage the clutch/release mechanism, unlock the arm only when ready, and re-run the geometric gate before software-controlled operation.

Firmware should expose arm-lock and cam-arm-nut permissives as discrete inputs if those switches exist in the final hardware. Until then, this is a documented operator procedure, not a powered control state.

2.14.9.3 Firmware requirements

- No ZERO_G state in the v1 state machine.
- No ZERO_G_ARM, ZERO_G_DISARM, ZERO_G_ENTERED, or ZERO_G_EXITED protocol messages.
- No sensorless admittance loop or Iq-zero hand-guided mode.
- Add/support a persistent `fz_tool_weight_offset_n` configuration value for the installed tool/head configuration.
- Log both raw and compensated force when practical: `fz_raw_n` and `fz_contact_n`. If only one canonical force channel is emitted, `fz_n` shall mean compensated contact force and the offset must be logged in the session metadata.
- Treat arm-lock / cam-arm-nut feedback, if installed, as safety/permissive inputs. Software-controlled motion is refused unless the arm is in the correct operating configuration.

2.15 Host ↔ Teensy Protocol

“**v1 framing reminder.** v1 only exercises the manual subset of this message set (MANUAL_START, SETPOINT, MANUAL_STOP, plus heartbeat/ACK/NACK/telemetry/event). SEGMENT_START, PAUSE, RESUME, ABORT, and SEGMENT_DONE must still be defined, parsed, and ACK-able in v1 — even if the host never sends them in production — so that Phase 3 program execution lights up without a protocol revision.”

2.15.1 Philosophy

This is a **setpoint/telemetry protocol**, not G-code.

- The host says: “execute this approved segment, maintain these setpoints, apply this modulation”
- The Teensy says: “here is current measured reality, here are alarms/events, here is whether I accepted”
- **No hidden strategy mutation.** The Teensy never silently changes setpoints.

Upstream-compatibility constraints on the protocol. Whatever concrete wire format is used, it must:

1. Carry every field defined in the controller-job contract (§7) and telemetry contract (§9) without lossy translation or renaming.
2. Support the full message set required by the state machine (§10), segment lifecycle (§12), manual-mode flow (§13), and geometric gate (§13.4.1).
3. Keep safety-relevant messages (ABORT, NACK, fault reports, heartbeat) bounded-latency and robust to single-frame loss.

These are the only constraints this document places on the protocol; they exist to keep the wire format aligned with the post-processor / sim / program pipeline upstream.

2.15.2 Message Classes

Host to Teensy:

- HEARTBEAT: Watchdog keepalive.
- SEGMENT_START: Begin new segment with parameters.
- SETPOINT: Update setpoints mid-segment, if needed.
- PAUSE: Request controlled pause.
- RESUME: Resume from pause.
- ABORT: Request controlled abort.
- MANUAL_START: Enter manual mode with initial setpoints.
- MANUAL_STOP: Exit manual mode, controlled ramp-down.
- ESTOP: Software E-stop, complementing the hardware E-stop path.

Teensy to Host:

- ACK: Command accepted.
- NACK: Command rejected, with reason code.
- TELEMETRY: Sensor data packet.
- EVENT: State change or alarm.
- SEGMENT_DONE: Segment duration complete.
- ABORT_COMPLETE: Abort sequence finished.

2.15.3 Protocol Requirements

Table 18 — Serial protocol requirements for deterministic host-to-Teensy communication.

Requirement	Detail
Versioning	Every frame includes protocol_version byte
Framing	Length-prefixed or delimited - no reliance on timing
Integrity	CRC-16 or CRC-32 on every frame
ACK/NACK	Every command from host gets explicit ACK or NACK
NACK reasons	Machine-readable fault/reason code, not text
Deterministic parsing	Fixed field order or tagged fields - no ambiguity
No text parsing	UI display strings are separate from protocol data

2.15.4 Example Frame Concepts

Host → Teensy SEGMENT_START:

```
[HEADER: version=1, type=SEGMENT_START, length=N]
segment_id: "seg-01"
duration_s: 1200.0
force_n: 45.0
table_rpm: 4.0
spindle_rpm: 40.0
spindle_direction: "cw"
mod_harmonic: 2
mod_amplitude_n: 5.0
mod_phase_deg: 37.2
[CRC]
```

Note: oscillation amplitude/center are **not** per-segment command fields. In v1 they are operator-entered configuration carried in the geometric gate record (§13.4.1) and passed once at SEGMENT_START / MANUAL_START. The Teensy reports derived amplitude/center back in telemetry (§9.1) for post-hoc characterization and divergence warnings.

Teensy → Host TELEMETRY:

```
[HEADER: version=1, type=TELEMETRY, length=N]
timestamp_us: 1234567000
table_angle_deg: 182.5
arm_angle_deg: 127.3
fz_n: 44.9
mx: 0.12
my: -0.08
mz: 0.02
spindle_rpm: 39.8
spindle_direction_actual: "cw"
table_rpm: 4.01
arm_amplitude_deg_derived: 31.2
```

```

arm_center_deg_derived: 0.1
state: RUNNING
force_setpoint_n: 45.0
[CRC]

```

Whether this is framed JSON-lines, compact binary (MessagePack, COBS-framed structs), or a custom format is an implementation choice. **The requirement is that it is documented, versioned, and robust.**

A follow-on **Host ↔ Teensy Protocol Spec v1** should define the exact wire format.

2.16 Machine Capability Descriptor

Your controller must expose a static capability profile so upstream software knows what can be safely requested. This example mirrors the current machine-capabilities.v1 schema in the active polisher-sim repo.

```

{
  "schema_version": "machine-capabilities.v1",
  "machine_id": "fullum-alpha",
  "machine_family": "swing-arm",
  "machine_name": "Fullum swing-arm polisher",
  "controller_version": "polisher-control/0.1.0",
  "last_verified": "2026-07-15T00:00:00Z",
  "supported_motion_families": ["swing-arm-rosette"],
  "force_range_n": [5, 80],
  "table_rpm_range": [0.5, 5.0],
  "spindle_rpm_range": [10, 80],
  "supported_spindle_directions": ["cw", "ccw"],
  "cam_amplitude_range_deg": [0, 45],
  "cam_offset_range_deg": [-30, 30],
  "force_modulation": {
    "supported": true,
    "max_harmonics": 3,
    "max_amplitude_n": 20.0,
    "notes": "Angle-synchronized force modulation computed from live table encoder angle."
  },
  "supported_dither_profiles": ["none", "default"],
  "segment_duration_limits": {
    "min_s": 10,
    "max_s": 7200
  },
  "pause_resume_support": true,
  "telemetry_channels": [
    { "name": "timestamp_us", "unit": "us", "sample_rate_hz": 100 },
    { "name": "table_angle_deg", "unit": "deg", "sample_rate_hz": 100 },
    { "name": "arm_angle_deg", "unit": "deg", "sample_rate_hz": 100 },
    { "name": "fz_n", "unit": "N", "sample_rate_hz": 100 },
    { "name": "mx", "unit": "N_mm", "sample_rate_hz": 100 },
    { "name": "my", "unit": "N_mm", "sample_rate_hz": 100 },
    { "name": "mz", "unit": "N_mm", "sample_rate_hz": 100 },
    { "name": "spindle_rpm_actual", "unit": "RPM", "sample_rate_hz": 100 },
    { "name": "spindle_direction_setpoint", "unit": "enum(cw|ccw)", "sample_rate_hz": 10 },
    { "name": "spindle_direction_actual", "unit": "enum(cw|ccw)", "sample_rate_hz": 10 },
  ]
}

```

```

    { "name": "spindle_drive_state", "unit": "enum", "sample_rate_hz": 100 },
    { "name": "spindle_drive_error", "unit": "bitfield", "sample_rate_hz": 100 },
    { "name": "spindle_bus_voltage_v", "unit": "V", "sample_rate_hz": 100 },
    { "name": "spindle_iq_a", "unit": "A", "sample_rate_hz": 100 },
    { "name": "spindle_motor_temp_c", "unit": "degC", "sample_rate_hz": 10 },
    { "name": "table_rpm_actual", "unit": "RPM", "sample_rate_hz": 100 },
    { "name": "arm_amplitude_deg_derived", "unit": "deg", "sample_rate_hz": 100 },
    { "name": "arm_center_deg_derived", "unit": "deg", "sample_rate_hz": 100 },
    { "name": "z_servo_iq_v", "unit": "V", "sample_rate_hz": 100 },
    { "name": "z_brake_engaged", "unit": "bool", "sample_rate_hz": 100 },
    { "name": "machine_state", "unit": "enum", "sample_rate_hz": 100 }
  ],
  "force_actuator": {
    "architecture": "counterweight_biased_torque_servo_v1",
    "servo_power_w": 1000,
    "servo_motor": "AutomationDirect SV2L-210B",
    "servo_drive": "AutomationDirect SV2A-2150",
    "coupling": "direct_drive_1_to_1",
    "transmission": "gt2_gt3_twin_power_belt_two_idlers",
    "control_mode": "torque",
    "max_apparent_mass_n": 80,
    "failsafe_brake": "nc_24vdc_electromagnetic",
    "auxiliary_feedback": "servo_iq_0_10v_analog",
    "notes": "Counterweight cancels head static weight; controller modulates residual differential only. Force PID closes
on Fz from the Kunwei KWR75B-CAN 6-axis sensor."
  },
  "spindle_drive": {
    "system": "CubeMars AK60-6 V1.1 KV80 integrated actuator",
    "status": "selected_final",
    "actuator": "CubeMars AK60-6 V1.1 KV80",
    "controller": "integrated AK-series drive/controller",
    "encoder": "integrated actuator sensing / vendor telemetry",
    "control_mode_v1": "velocity",
    "preferred_runtime_interface": "CAN, topology/IDs/rates/timeouts to freeze before firmware lock",
    "supply_voltage_v_working_basis": 24,
    "safety_stop_basis": "remove actuator power through safety-controlled post-contactor bus; CAN stop is not safety-
rated",
    "rated_torque_nm_approx": 3,
    "peak_torque_nm_approx": 9,
    "required_firmware_items": [
      "saved_ak60_commissioning_parameters",
      "velocity_command_scaling",
      "spindle_direction_mapping",
      "actual_rpm_telemetry",
      "drive_fault_mapping",
      "current_torque_limit_configuration",
      "thermal_limit_configuration",
      "safety_power_removal_bench_test"
    ]
  },
  "safety_limits": {
    "max_force_n": 80,
    "max_table_rpm": 5.0,
    "max_spindle_rpm": 80,
    "notes": "Hard safety limits enforced regardless of job request. Counterweight sets a passive mechanical ceiling at
80 N apparent mass that no software path can exceed."
  },
  "tool_handling": {

```

```

"powered_zero_g_supported": false,
"force_compensation": "fz_contact_n = fz_raw_n - fz_tool_weight_offset_n",
"removal_method": "mechanical_lock_remove_cam_arm_nut_move_arm_aside_remove_tool_by_hand",
"notes": "Tool-side force sensing is only valid through the tool/blank load path. In-air handling is a mechanical
procedure, not a powered firmware state."
},
"known_constraints": [
  "Single-pass controller execution only in v1",
  "No dwell-time modulation in v1",
  "Cam amplitude and offset are mechanical settings, not software-actuated in v1",
  "Operator must confirm geometric parameters before RUNNING or MANUAL",
  "Tool removal is handled by the mechanical lock / cam-arm-nut removal / manual-removal workflow, not by a
powered Zero-G firmware state"
],
"unknowns": [
  "Z-axis drive control mapping pending confirmation on the SV2A-2150",
  "Measured force loop bandwidth pending commissioning",
  "AK60 command/telemetry mapping, CAN topology, safety power-removal behavior, and thermal/current limits
still to be documented and bench-tested"
]
}

```

This describes what is supported, not what might work experimentally. Conservative values. Update when hardware is proven.

2.17 Timing Summary

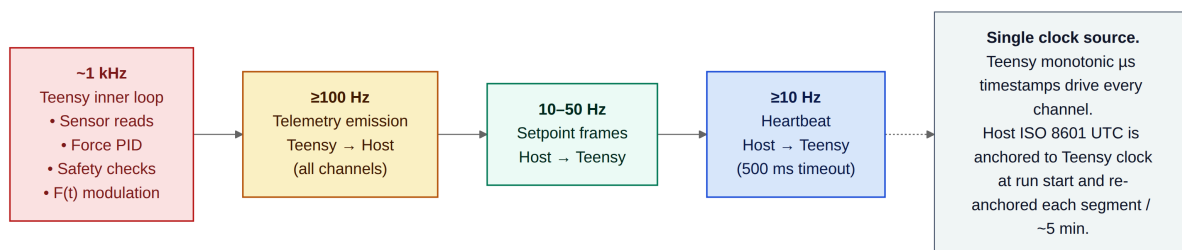


Figure 9 — Loop and stream rates. The single Teensy monotonic clock anchors every channel; host UTC is bound to that clock at run start and re-anchored each segment (or every 5 min during open-ended manual sessions). Mermaid source: Fullum-Polisher-Timing-Rates.mmd.

Table 19 — Timing targets for control loops, command streams, telemetry, and events.

Loop / Stream	Target Rate	Tolerance	Owner
Teensy inner loop	1 kHz	Must be stable $\pm 5\%$, document actual	Firmware
Force PID execution	Same as inner loop	-	Firmware
Sensor reads	Same as inner loop	-	Firmware
Safety limit checks	Same as inner loop	-	Firmware
Telemetry emission (Teensy $\rightarrow$ Host)	≥ 100 Hz	Document actual rate	Firmware
Setpoint frames (Host $\rightarrow$ Teensy)	10-50 Hz	-	Host
Heartbeat (Host $\rightarrow$ Teensy)	≥ 10 Hz	500 ms timeout	Host
Telemetry logging to CSV	≥ 100 Hz	Match emission rate	Host
State machine event emission	On every transition	<10 ms latency	Host

Timestamp requirements:

- Teensy uses one monotonic microsecond clock for all telemetry
- Host records ISO 8601 UTC for events and run-log timestamps
- Telemetry CSV uses Teensy timestamps (not host timestamps)
- On run start, record the mapping: `teensy_timestamp_us` $\leftrightarrow$ `host_utc` for correlation
- For long runs, re-anchor the mapping periodically (at least once per segment boundary, or every 5 minutes for open-ended manual sessions) so that downstream offline analysis can correct for Teensy-clock drift across long captures. Each anchor point is appended to the run-log / manual-session-log as an event.

2.18 Data Storage, Export, and Remote Access

The RPi must store all run data locally and support remote access for engineering analysis. The telemetry data produced by this machine is consumed by external planning tools - it needs to be exportable and accessible without requiring Normand to manually copy files.

2.18.1 Local Storage

All telemetry and run data must be written to a **USB SSD connected to the RPi**, not to the SD card.

Table 20 — Local storage requirements for reliable telemetry and run-artifact retention.

Requirement	Detail
Storage medium	USB 3.0 SSD, minimum 256 GB (1 TB recommended)
Mount point	/data/ (or equivalent, consistent path)
Write target	All run-logs, manual-session-logs, telemetry CSVs, event logs
SD card role	OS, application code, and configuration only - no run data
Power loss	Use fsync after writing run artifacts. Telemetry CSV should flush periodically (every 5-10 seconds at minimum).

Data volume estimate: At $100 \text{ Hz} \times 20 \text{ channels} \times 4 \text{ bytes} \approx 8 \text{ KB/s} \rightarrow 29 \text{ MB}$ per hour of polishing. A 1 TB drive holds years of continuous use.

2.18.2 File Structure

Run data must follow a predictable directory layout so sync tools and analysis scripts can find files reliably:

```

/data/
├── runs/
│   ├── run-2026-07-001/
│   │   ├── run-log.v1.json
│   │   ├── telemetry.csv
│   │   ├── telemetry-derived.parquet
│   │   ├── segment-stats.json
│   │   ├── dwell-map.npz
│   │   ├── work-map.npz
│   │   ├── anomaly-windows/
│   │   └── manifest.json
│   ├── run-2026-07-002/
│   └── ...
├── manual/
│   ├── manual-2026-07-15-001/
│   │   ├── manual-session-log.v1.json
│   │   ├── telemetry.csv
│   │   ├── telemetry-derived.parquet
│   │   ├── segment-stats.json
│   │   ├── dwell-map.npz
│   │   ├── work-map.npz
│   │   └── manifest.json
│   └── ...
├── capabilities/
│   └── machine-capabilities.v1.json
└── status.json      ← machine state summary (see §17.4)

```

Naming convention: Directory names match the run_id or session_id from the log files. No spaces, no special characters.

2.18.3 Remote Access (Tailscale)

The RPi must be configured to join a **Tailscale mesh VPN** so that Antoine can access telemetry data and monitor the machine remotely.

Table 21 — Remote-access requirements for commissioning, diagnostics, and telemetry retrieval.

Requirement	Detail
VPN	Tailscale installed and authorized on Atomaste network
Access	SSH (port 22) over Tailscale - no public IP, no port forwarding needed
Address	Stable Tailscale IP (100.x.x.x), hostname registered
Network dependency	None during operation. Machine runs, logs, and saves data even with no network. Tailscale is for remote access only.
Normand's network	Requires basic internet at the shop (WiFi or Ethernet). Tailscale handles NAT traversal.

What remote access enables:

- SSH into the RPi to inspect run data, debug issues, check logs
- rsync or scp to pull telemetry files for analysis
- Monitor machine state during commissioning without being on-site
- Push software updates to polisher-control when needed

2.18.4 Machine Status Endpoint

The RPi should maintain a small `status.json` file (updated every few seconds) that summarizes current machine state:

```
{
  "machine_id": "fullum-alpha",
  "controller_version": "polisher-control/0.1.0",
  "state": "MANUAL",
  "uptime_s": 3600,
  "last_run_id": "run-2026-07-001",
  "last_session_id": "manual-2026-07-15-003",
  "disk_usage_gb": 12.4,
  "disk_free_gb": 243.6,
  "timestamp": "2026-07-15T14:30:00Z"
}
```

This is a simple file on disk - no web server required. Antoine can `cat` it over SSH. A lightweight web dashboard can be added later if needed.

2.18.5 Auto-Sync (Post-Run)

After each run or manual session completes, the RPi should attempt to sync the new data to a remote server:

Table 22 — Post-run synchronization requirements and operator fallback path.

Requirement	Detail
Trigger	Run or manual session completes (any result state)
Method	rsync over Tailscale (SSH transport)
Target	Configurable remote path (set during initial setup)
Failure behavior	Log the failure, retry on next sync cycle. Never block the operator.
Manual fallback	Normand can always export to USB stick from the touchscreen UI

The sync is a background convenience - the primary data path is always local storage. If network is down, data accumulates locally and syncs when connectivity returns.

2.18.6 Telemetry Retention and Post-Processing Contract

The machine should acquire and save telemetry at the full control/diagnostic rate, but the permanent engineering archive should be derived from each run. This keeps the future ML and digital-twin dataset useful without turning six months of operation into a CSV-storage problem.

Responsibility split

- **Teensy firmware:** emit synchronized full-rate telemetry with stable channel names and a monotonic timestamp.
- **Host controller:** persist the raw telemetry stream locally, write run/manual logs and hashes, and launch a non-blocking post-run routine after each run or manual session.
- **Post-run routine:** produce compressed and derived artifacts for analysis. This routine belongs on the host or engineering workstation, not inside the Teensy firmware loop.
- **Retention maintenance:** prune or move old raw full-rate captures according to policy. This may run daily, weekly, or monthly, but it must not be the first time a run is processed.

Per-run post-processing trigger

The post-run routine should run immediately after every completed, aborted, or faulted run/manual session. It must never block the operator from starting the next safe operation. If processing fails, the host preserves the raw files, logs the failure, and retries later.

Derived artifacts to generate

- telemetry-derived.parquet, or equivalent columnar compressed file, containing a permanent 10-20 Hz analysis trace.
- segment-stats.json, containing mean/min/max/std/p95 values and force-error integrals for each segment or manual-session interval.
- dwell-map.npz, containing position/time accumulation in the machine or mirror analysis frame.
- work-map.npz, containing Preston-weighted accumulation using force, velocity, and dwell time.
- anomaly-windows/, containing preserved full-rate windows around faults, force excursions, sensor dropouts, pauses, operator interventions, or other flagged events.
- Manifest entries and SHA-256 hashes for every raw and derived artifact.

Retention policy baseline

Table 23 — Baseline telemetry retention policy for long-duration polisher operation.

Data tier	Recommended retention	Purpose
Live control buffer	Current run only	Control diagnostics and immediate fault recovery
Raw full-rate telemetry	30-90 days during commissioning, then 7-30 days in stable production	Debugging, sensor validation, and replay of recent issues
Event-triggered full-rate windows	Keep indefinitely unless manually reviewed and discarded	ML/anomaly training and root-cause analysis
Downsampled 10-20 Hz trace	Keep indefinitely	Long-term digital twin and run-history analysis
Segment summaries, dwell maps, work maps, metrology references, logs, hashes	Keep indefinitely	Calibration, removal-model updates, audit trail

The permanent trace should default to **20 Hz** while force-control dynamics and real machine behavior are still being characterized. The rate can be reduced to **10 Hz** later if analysis confirms that position/work-map accumulation and calibration quality are not degraded.

v1 scope note

For first hardware proving, raw full-rate capture, run/manual logs, hashes, and stable filenames are mandatory. The derived artifacts may start as a simple post-run script, but the contract above should be respected from the beginning so that Phase 1 data can feed polisher-sim, polisher-post, and future ML/digital-twin development without a schema migration.

2.18.7 What This Does NOT Include

- No cloud services or third-party platforms
- No real-time telemetry streaming requirement (nice-to-have for commissioning, not required)
- No dependency on network for machine operation - the machine works fully offline
- No remote control of the machine (safety concern - operator must be physically present)

2.19 First-Proving Scope (v1)

This section is the **in-scope / out-of-scope boundary** for the first hardware integration. For the detailed pass/fail conditions each item must satisfy, see §19 (Acceptance Criteria) — §18 says *what* is built, §19 says *how it's proven done*.

2.19.1 In Scope for v1

Functional capabilities the v1 delivery must contain. Each line maps to one or more acceptance clauses in §19.

Table 24 — v1 firmware and controller capabilities mapped to acceptance criteria.

Capability	Maps to §19
Accept a single controller-job.v1 package, 4-gate intake validation	§19.1
Sequential segment execution, force PID, table/spindle RPM control	§19.2
Table + arm encoder acquisition, F/T sensor (at minimum Fz)	§19.2, §19.11
Force modulation from live encoder angle	§19.2
Timestamped telemetry at ≥ 100 Hz, CSV export	§19.3
10-state machine with MANUAL, mandatory operator acknowledge	§19.4
Pause / resume / abort, E-stop, host $\leftrightarrow$ Teensy watchdog	§19.5
Event + alarm recording, run-log.v1 with commanded vs actual	§19.6
Machine capability descriptor	§19.1
Manual mode — operator direct-control with live setpoint adjustment and manual-session-log	§19.7
USB SSD storage with /data/runs/{id}/, /data/manual/{id}/ layout	§19.8
Tailscale access, status.json, USB-stick export from touchscreen	§19.8
Geometric gate for r_menante, L_menee, R_tool, configured arm amplitude, configured arm center	§19.10
KWR75B-CAN transport with status-word logging and stale-frame watchdog	§19.11
Tool-weight compensation and safe removal workflow (§13.9) — compensated force channel, offset logging, mechanical lock / cam-arm-nut removal / manual-removal procedure	§19.12

2.19.2 Explicitly Optional for v1

- **Powered Zero-G / admittance manipulation mode:** cancelled for MVP; secondary-priority future-feature bucket only, and only if core manual operation, telemetry, safety, and commissioning are complete with time remaining.
- Advanced dwell-profile execution (angle-varying dwell time)
- Arbitrary angle lookup tables
- Multi-pass orchestration inside the controller
- Metrology import or interpretation
- Calibration logic
- Autonomous optimization
- Remote control of the machine (safety: operator must be present)
- Auto-sync (rsync post-run - nice-to-have, not required for v1)
- Multi-machine coordination

2.20 Acceptance Criteria

The machine-side implementation is considered **aligned with the Polisher Software Suite** when all of the following are demonstrably true:

2.20.1 Input Contract

- [] Accepts controller-job.v1 JSON and rejects job.v1 or malformed input
- [] 4-gate validation sequence implemented exactly as specified (§7.2)
- [] Rejects jobs targeted at a different machine_id
- [] Rejects jobs with incompatible controller_version

2.20.2 Execution

- [] Segments execute sequentially in sequence_index order
- [] Force PID maintains commanded force within $\pm 10\%$ steady-state
- [] Force modulation computed from live table encoder angle (not time-based approximation)
- [] Table and spindle RPM controlled to commanded setpoints
- [] Toolhead spindle direction follows commanded cw / ccw selection and is logged explicitly
- [] Segment duration tracked accurately (pause time excluded)

2.20.3 Telemetry

- [] All core channels (§9.1) logged at ≥ 100 Hz
- [] Single monotonic timestamp source for all channels
- [] Commanded and actual values both present in telemetry
- [] Sensor faults detectable (NaN, validity flag, or sentinel)
- [] Telemetry CSV parseable by standard tools (pandas, numpy)
- [] Raw full-rate telemetry is preserved before any compression, downsampling, or retention action
- [] Post-run processing failure is detectable from the manifest or status output

2.20.4 State Machine

- [] All 10 states (including MANUAL) and transitions implemented per §10.2
- [] Illegal transitions raise errors and are logged (never silently ignored)
- [] Operator acknowledge gate is mandatory (cannot be bypassed)
- [] Every transition emits a timestamped event

2.20.5 Safety

- [] E-stop triggers immediate all-stop independent of software state
- [] All interlocks in §11.1 implemented with documented thresholds
- [] Watchdog timeout causes controlled pause, then abort
- [] FAULTED state can only exit via explicit operator reset
- [] Firmware safety checks run at inner loop rate (1 kHz)

2.20.6 Output Contract

- [] Run-log.v1 emitted for every run (including faulted/aborted)
- [] commanded and actual blocks are structurally distinct, with flat commanded values and mean/min/max suffixes for actual values
- [] result_state is unambiguous and matches terminal machine state
- [] Events include all state transitions and operator interventions
- [] Alarms promoted to top-level alarms[] array
- [] Hashes present for telemetry and run-log integrity

2.20.7 Manual Mode

- [] MANUAL state reachable from IDLE via enter_manual trigger
- [] Operator can set and adjust force, table RPM, spindle RPM, spindle direction, and modulation live; oscillation amplitude/center are entered once via the geometric gate (§13.4.1) and not live-editable
- [] Configured oscillation amplitude/center values are persisted with the geometry record and compared against the firmware-derived values once regime is established, with non-blocking divergence warnings
- [] All setpoint changes clamped to machine capability limits
- [] Every setpoint change logged as a timestamped event
- [] Safety interlocks identical to job mode (§11)
- [] Telemetry logged at same rate and format as job mode (§9)
- [] Manual-session-log emitted on exit with setpoint_history and actual_summary
- [] Cannot enter manual mode with a job loaded (and vice versa)
- [] Fault during manual mode transitions to FAULTED (same as job mode)

2.20.8 Data Storage and Remote Access

- [] All run data written to USB SSD, not SD card
- [] File structure follows §17.2 layout (/data/runs/{id}/, /data/manual/{id}/)
- [] status.json maintained with current machine state
- [] Tailscale installed and reachable from Atomaste network
- [] SSH accessible over Tailscale
- [] Operator can export run data to USB stick from touchscreen UI
- [] Post-run routine runs automatically after each run/manual session or logs a retryable failure
- [] Derived telemetry archive, segment statistics, dwell/work maps, and anomaly windows follow §17.6 naming and retention rules once implemented

2.20.9 Boundary Discipline

- [] Controller does not generate, modify, or interpret polishing strategy
- [] Controller does not import or act on metrology data
- [] Controller does not optimize pass parameters
- [] Any safety clipping or rejection is explicit (NACK + event, never silent)

2.20.10 Kinematics and Geometric Gate

- [] UI enforces a blocking modal for `r_menante`, `L_menee`, `R_tool`, `configured_arm_amplitude_deg`, `configured_arm_center_deg` before RUNNING or MANUAL
- [] Stored geometry (all five parameters) includes timestamp and operator, persisted into every log
- [] Firmware NACKs `SEGMENT_START` / `MANUAL_START` when geometry is stale or any of the five parameters is missing (`GEOMETRY_NOT_VALIDATED`)
- [] Firmware linearizes arm angle from encoder + `r_menante` + `L_menee` before dwell/modulation math
- [] Raw arm encoder angle and linearized arm angle are both present in telemetry
- [] Firmware emits `arm_amplitude_deg_derived` and `arm_center_deg_derived` in telemetry, NaN until 5 complete oscillation cycles have elapsed since `SEGMENT_START` / `MANUAL_START`
- [] Host raises a non-blocking warning event when derived amplitude/center diverge from configured values beyond the configured tolerance

2.20.11 KWR75B-CAN Transport

- [] Teensy 4.1 CAN interface configured with isolated CAN transceiver, dedicated KWR75B-CAN link
- [] CAN receiver decodes the vendor frame map into the canonical `FT_LoadSample` struct (§9.4)
- [] F/T status word logged as `ft_status`
- [] Stale-frame watchdog raises `FT_SENSOR_INVALID` on configurable timeout
- [] Measured CAN sample arrival rate documented and meets the force-loop requirement

2.20.12 Tool-Weight Compensation and Safe Removal Workflow

- [] `fz_tool_weight_offset_n` stored in the machine configuration for the installed tool/head configuration
- [] Force PID uses compensated contact force (`fz_contact_n = fz_raw_n - fz_tool_weight_offset_n`), not raw tool-side force, for `force_setpoint_n` comparison
- [] Session/run metadata records the active force offset and sign convention
- [] Telemetry logs `fz_raw_n` and `fz_contact_n` when practical; if only `fz_n` is emitted, `fz_n` is explicitly documented as compensated contact force
- [] Firmware contains no powered Zero-G/admittance state or dead-man Zero-G command path
- [] Operator workflow documents the mechanical removal sequence: stop, zero force, lock swing arm/arc, remove cam arm nut to free arm rotation, move arm aside, remove tool by hand
- [] If arm-lock / cam-arm-nut feedback switches are installed, software-controlled motion is refused unless those signals indicate the correct operating configuration

2.21 Recommended Follow-On Documents

After this spec, the next deliverables should be:

Table 25 — Recommended follow-on control documents after the v1 firmware specification.

Document	Purpose	Priority
Host ↔ Teensy Protocol Spec v1	Exact wire format, message IDs, framing, CRC, byte layout	High - needed for firmware development
Telemetry Channel Spec v1	Exact field names, units, ranges, validity encoding, sample rates	High - needed for DAQ integration
Event/Alarm Code Table v1	Canonical event codes, severity levels, descriptions, recommended actions	Medium - needed for operator UI
Controller-Job Schema v1 (formal JSON Schema)	Machine-readable schema for validation (Draft 2020-12)	Medium - reference already exists in suite
Machine Capability Profile v1 (for Fullum-alpha)	Actual measured limits from hardware commissioning	Medium - update after hardware testing
Operator Workflow Guide	Step-by-step for loading jobs, acknowledging, monitoring, handling faults	Low - needed before first operator training

2.22 Open Questions for Cédric (Control & Firmware Side Only)

Cédric’s `Calculus_Notes` currently cover **Zone 3 (Swing Arm)** and **Zone 5 (Controls & UI)**. Mechanical design of the other zones is out of scope for this document, but the following **control/firmware-facing interfaces** still need Cédric’s input before v1 can be locked. Pure mechanical questions (bearings, counterweight balance, shaft concentricity, actuator mounting) belong in the mechanical spec and are intentionally not listed here.

Every item flagged “**Preferred, confirm in detailed design**” or “**Open, Cédric to finalize**” in §3 appears here or in §3 itself. These are the decisions needed before the electrical/firmware BOM can be frozen. Once closed, update the §3 status columns accordingly and strike the matching item below.

2.22.1 Zone 1 — Table (Control Interface)

- Confirm the KBSI-240D opto-isolated interface as the DC-variator command path, or name the replacement.
- Confirm BiSS-C frame details for the ordered table and arm encoders: timing, CRC/status bits, RS422 transceiver/pinout, shielding, and startup/error handling.
- Confirm whether the table drive or its command path changes electrically as part of the overhaul.

2.22.2 Zone 2 — Spindle / Toolhead Drive (Control Interface)

- Define the AK60 runtime command and telemetry path to the controller. CAN is the working basis, but Cédric must freeze CAN topology, IDs, frame rates, stale-frame timeout, startup behavior, and error isolation before firmware lock.
- Define command scaling for spindle velocity setpoint, clockwise/counter-clockwise direction mapping, enable/disable, fault reset, ramp-to-zero / live-reversal policy, and safe stop behavior.
- Define the AK60 telemetry subset mapped into the host logs: actual RPM, actuator state, actuator error, bus/supply voltage, torque/current if available, and motor/drive temperature if available.
- Store/export the final AK60 commissioning parameter set with the machine software release.
- Define the spindle enable/fault I/O and the hardwired AK60 power-removal path through the HOA + safety chain; prove with a bench E-stop/power-loss test.

2.22.3 Zone 4 — Z-Axis / Force Actuator (Control Interface)

The Z-axis architecture is now locked as a **counterweight-biased torque-mode AC servo** using the AutomationDirect SV2L-210B / SV2A-2150 selection from Cédric's BOM (see §3.2, §3.5, §3.8). The remaining items are:

- **SV2A-2150 control mapping:** confirm the exact torque/current command mode, scaling, enable/fault wiring, and current/Iq monitor path. Preferred feedback remains 0-10 V into the Teensy ADC; if not available, update `z_servo_iq_v` naming and telemetry mapping before commissioning.
- **Z limit-switch / hard-stop wiring** into the safety chain — soft limits in firmware are not sufficient by themselves.
- **NC 24 VDC brake coil** driver path and diagnostic feedback (brake-engaged sense) so the `z_brake_engaged` telemetry channel (§9.1) reflects actual state, not commanded state.

2.22.4 Cross-Cutting Items

- **Safety relay model.** §3.5 offers Pilz PNOZ X1 or Banner XS26-2 — pick one so the wiring drawings can start.
- **KWR75B-CAN frame map confirmation** against the Kunwei package/manual (CAN IDs, byte order, scaling, status bits, update rate).

2.22.5 Future Scope (post-v1)

- **Cam positioning device.** A mechanical device to set the oscillation amplitude and center of the swing arm (today: set by hand off a physical scale, no instrumentation). When this is introduced, it will reintroduce a real cam encoder — and, plausibly at that point or shortly after, cam actuation. At that time: re-add a cam encoder row to §3.1; re-expand §3.3 beyond “not instrumented”; add `cam_amplitude_deg` / `cam_offset_deg` telemetry channels back to §9.1; update `cam_type` in §15 away from “mechanical_no_instrumentation_v1”; and decide whether the operator-entered `configured_arm_*` fields become sensor-read values, stay as cross-check configuration, or are deprecated entirely.
- **Controller-varied arm oscillation speed.** In v1 the swing-arm motor runs at a constant operator-set RPM (§3.2). In the full sim/program pipeline — where aberration correction and controlled dwell shaping are driving the machine — this setpoint will be varied over time by the controller on a per-segment basis (and eventually intra-segment). The v1 command path is already VFD-ready, so this is a firmware/capability expansion, not a hardware redesign: add a `commanded_arm_rpm` field (or an

equivalent time-profile) to the segment structure, expose the range in §15, and add it to the setpoint/telemetry pair.

2.23 Related Documents

- 00-Project-Index
- System-Map
- Polisher-Control-System
- Command-and-Firmware-Guide
- Polisher-Post-System
- Polisher-Sim-System
- Job-Schema-v1
- Run-Log-Schema-v1
- Machine-Capabilities-Schema-v1
- Authoritative schema bundle: polisher-sim/schemas/ (maintained by Antoine, distributed with each spec revision)

2.24 Appendix A: Large-Format Control Figures

These appendix copies are provided for readability. The main-body figures preserve flow and context; the appendix figures are formatted for landscape review.

2.24.1 Appendix A.1: Figure 4 Landscape Copy

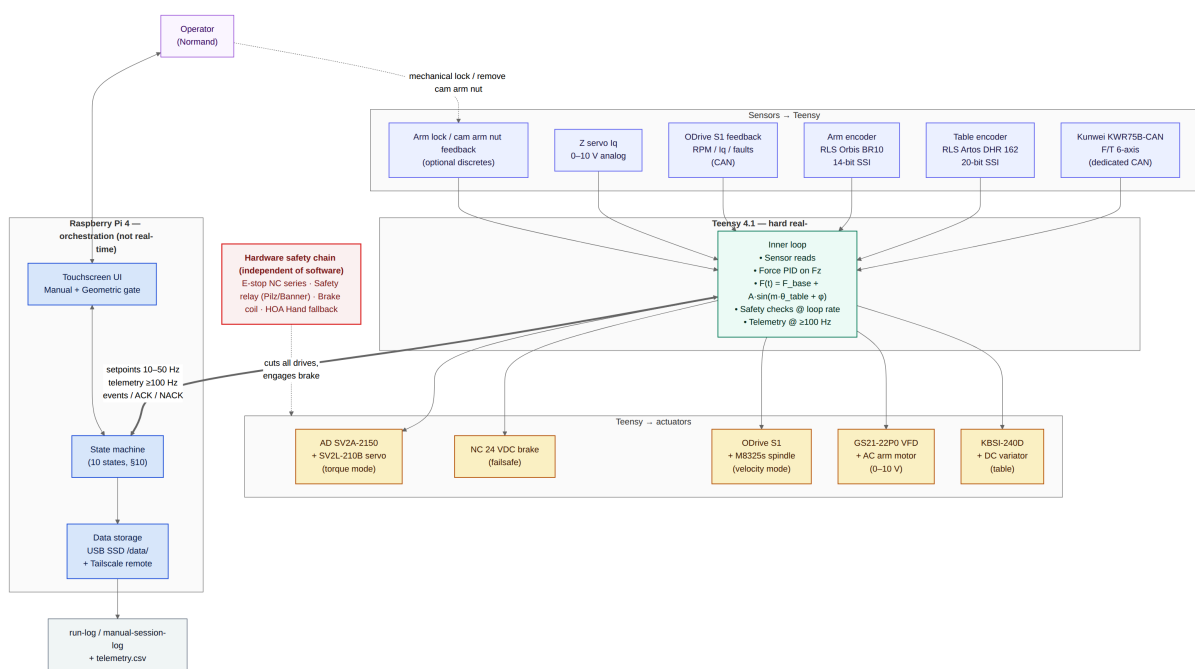


Figure 10 — Large-format landscape copy of Figure 4, full system architecture.

2.24.2 Appendix A.2: Figure 5 Landscape Copy

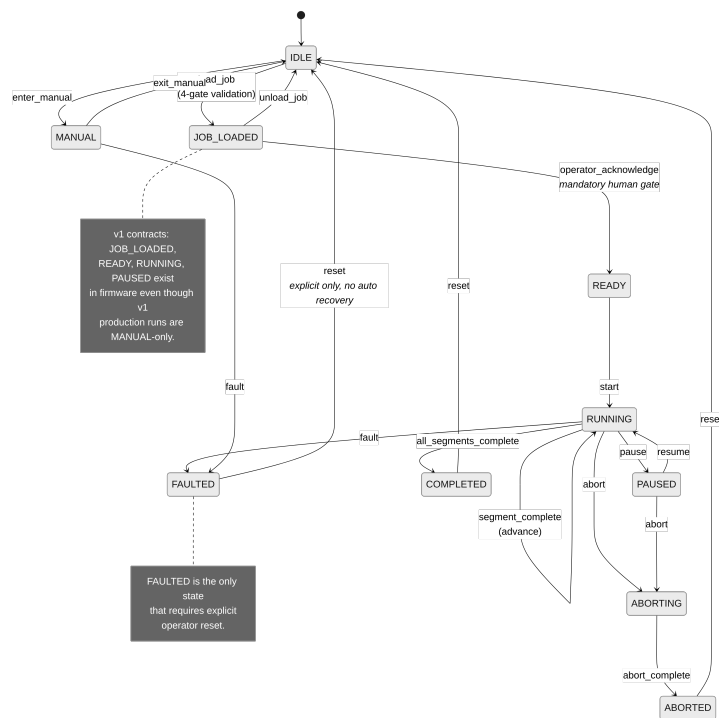


Figure 11 — Readable large-format copy of Figure 5, controller state machine. This copy uses the same state content as *Fullum-Polisher-State-Machine.mmd*, arranged vertically so transition labels remain readable on the landscape PDF page.

2.24.3 Appendix A.3: Figure 6 Landscape Copy

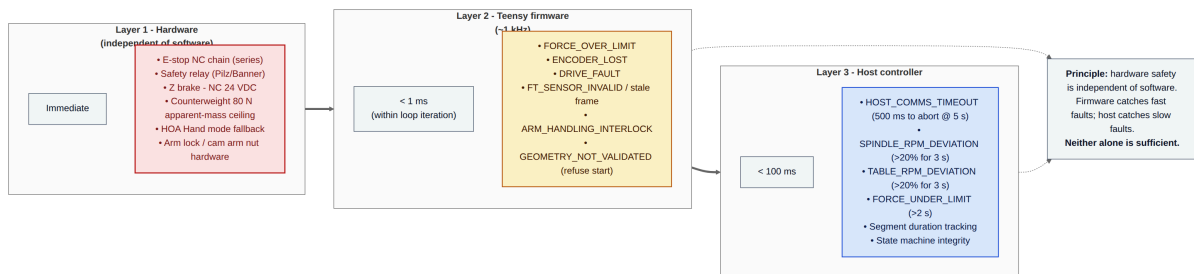


Figure 12 — Large-format landscape copy of Figure 6, same safety hierarchy content arranged horizontally for readability.