Controlling a Hybrid XYZ CNC Setup

X axis: voice coil actuator with magnetic linear displacement sensor | Y & Z axes: stepper motors

Why the X Axis Needs Different Treatment

A voice coil (linear motor) has no detents, no lead screw, and no mechanical self-locking. Unpowered, it doesn't stay put, and it doesn't move in discrete "steps" — it's a force-output device ($F = k \cdot I$). That means the X axis must run as a closed-loop servo axis using the linear encoder for position feedback, while Y and Z can remain open-loop step/dir axes.

Architecture for the X Axis

Three pieces are required:

- **Current-mode amplifier / servo drive** for the voice coil — not a stepper driver. Options include Advanced Motion Controls (AMC), Copley Controls, Elmo, or Applied Motion linear servo drives. These accept a command signal (analog ±10V, step/dir, or a digital fieldbus like EtherCAT/CANopen) and close an internal current loop.
- **Feedback wiring** from the magnetic linear sensor into either the drive itself or the motion controller — check whether the sensor outputs quadrature (A/B/Z), SSI/BiSS (absolute), or analog sine/cosine. This determines what interface the drive/controller needs.
- **A PID (or PID + feedforward) position loop** closing around that feedback. This loop can live in one of two places:
  - **In the servo drive** — the drive does its own PID and accepts step/dir or analog velocity commands from the main CNC controller. Simplest integration; the main controller just "sees" a third step/dir-like axis.
  - **In the motion controller** — the controller reads encoder counts directly and computes the PID output; the drive just amplifies current. More control/tuning flexibility, needed for true coordinated multi-axis interpolation accounting for the servo's dynamics.

Controller / Software Choice

Given the mix of a true servo axis with two stepper axes, the main options are:

- **LinuxCNC** — the most common choice for exactly this hybrid case. With a Mesa FPGA card (7i92, 7i96, etc.) it natively supports PID-closed servo axes (reading encoder feedback, driving step/dir or PWM+analog output to the voice coil drive) alongside pure step/dir stepper axes, all coordinated together. Very likely the best practical path unless already committed to industrial hardware.
- **Industrial motion controllers** — Galil, Delta Tau/Omron PMAC, ACS — handle mixed axis types natively and have mature auto-tuning tools, at higher cost.
- **Mach3/4, grbl, Klipper** — these are step/dir-centric and don't natively close a servo loop from a linear encoder. Usable if the voice coil drive does its own internal PID and just accepts a step/dir command, but coordinated-motion awareness of the servo's actual dynamics is lost.

Practical Setup Steps

1. Identify sensor output type

Check your magnetic linear encoder's datasheet: incremental quadrature (A/B/Z), absolute SSI/BiSS, or analog sin/cos. This decides what input your servo drive or controller card needs — get it wrong and nothing else works.

2. Choose where the PID loop lives

Decide whether the voice coil's servo drive closes its own position/current loop (drive accepts step/dir or analog command from your main controller) or whether your motion controller closes the loop directly (controller reads encoder counts, outputs current command to a simpler amplifier). LinuxCNC + Mesa card is the common route for the second option.

3. Wire the voice coil to a current-mode servo amplifier

Never drive a voice coil with a stepper driver or straight PWM to a motor terminal — it needs a proper linear/servo current amplifier (e.g. AMC, Copley, Elmo) sized for its force/current rating, with the encoder feedback wired into it or into your controller's encoder input.

4. Wire Y and Z as normal step/dir stepper axes

These stay open-loop through standard stepper drivers (e.g. TB6600, DM542, or your controller's onboard drivers) — no change needed from a typical CNC build.

5. Home and reference the X axis

Since the voice coil has no inherent zero position, you need a home switch or an index pulse from the encoder to establish a repeatable reference on power-up, since position is lost the moment power is cut.

6. Tune the X-axis PID loop

Start with proportional-only gain until you get stable positioning without oscillation, then add derivative for damping and a touch of integral for steady-state accuracy. Voice coils have very low mechanical damping/inertia compared to leadscrew-driven axes, so they tend to be twitchier to tune than a typical servo motor + ballscrew setup — velocity and acceleration feedforward terms help a lot here.

Safety Note

If the voice coil loses power or the servo loop faults, the axis has zero holding force — it will drift or fall under gravity or spring load. If X is vertical or under any preload, a mechanical brake, a spring-return-to-safe-position, or at minimum a fast fault-detection routine that halts Y/Z motion the instant X's servo drive reports a fault is strongly recommended.

comparison of the controller solutions discussed, followed by a proposed architecture that addresses each one's gaps.

Solution	Hardware Structure	Capabilities	Limitations
ACS Motion Control / PI G-901, G-910	Single integrated controller+amp chassis; each axis port individually configurable for servo, DC, voice coil, or stepper motor; encoder feeds directly into the same unit	True coordinated multi-axis interpolation across mixed motor types; native mixed-motor support (no external glue logic); mature auto-tuning/diagnostics; industrial reliability	High per-axis cost (often four figures+); proprietary firmware/software, vendor lock-in; steep configuration learning curve; often overkill for a 3-axis build
Galil DMC / Delta Tau (Omron) PMAC	Multi-axis controller card/box; servo axes PID-closed internally off encoder feedback, stepper axes via onboard step/dir generators	Deterministic coordinated motion across mixed axis types; extensive motion programming language; decades of industrial track record	Expensive; dated/complex programming environments; licensing costs add up; heavier support burden
Dedicated voice-coil drive (e.g. Moticont) + separate generic stepper controller	Two independent controllers: voice-coil drive closes its own PID internally off the linear encoder; a separate ordinary board	Low cost, modular, easy to source; drive is purpose-tuned for voice coils out of the box; simple wiring	**No coordination between the two controllers** — the G-code planner driving Y/Z has zero visibility into X's actual following error, so contoured moves involving X can desync

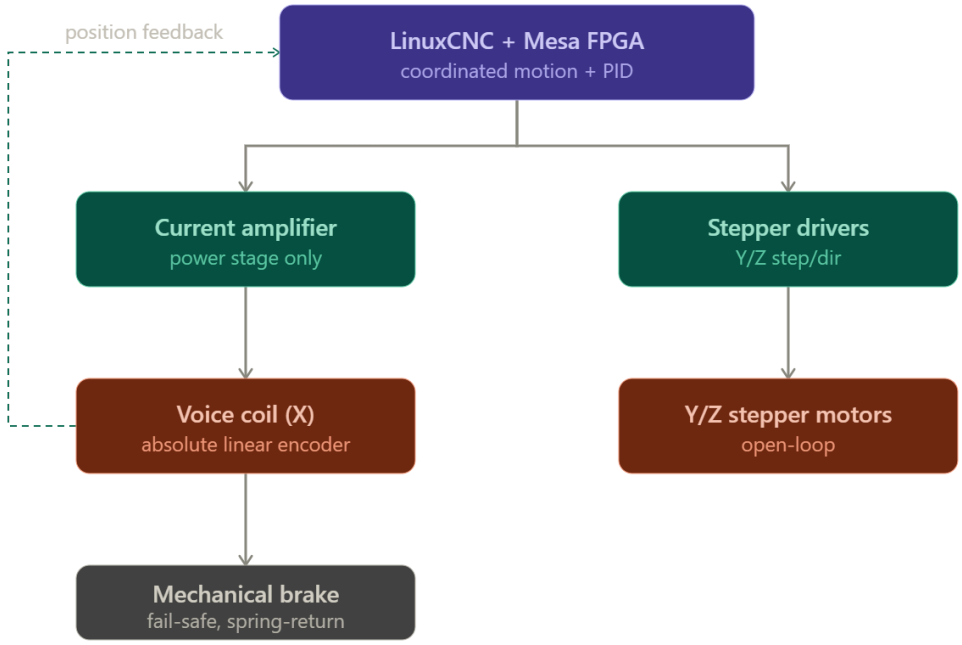
	(Mach-class) drives Y/Z open-loop		under load; two separate config/tuning environments
LinuxCNC + Mesa FPGA I/O card (7i92/7i96, etc.)	Single PC running LinuxCNC; Mesa card does encoder counting + step/dir generation; HAL closes X's PID loop off encoder feedback, output feeds an external power amp; Y/Z run as normal step/dir joints	Genuinely coordinated motion including the servo axis (X's error is visible to the trajectory planner); open, inexpensive, highly customizable; large community	Requires real-time Linux/HAL expertise; PID tuning is manual, no auto-tune; still needs an external current amp (Mesa isn't a power stage); community support only, no vendor SLA
Mach3/Mach4, grbl, Klipper + external self-closing voice-coil drive	Step/dir-centric hobby controller; voice-coil axis driven "blind" as a step/dir input into a drive that closes its own internal loop	Cheapest, most familiar toolchain; minimal Y/Z setup	Main controller has **zero visibility** into X's real position/following error — can't detect or react to lag or faults on that axis; closed-loop benefit is largely wasted at the system level

Proposed architecture

No single option above is strictly dominant — the industrial controllers (ACS/PI, Galil, PMAC) solve coordination but cost the most and lock you into proprietary tooling; the drive-pair and Mach/grbl approaches are cheap but blind to X's real behavior; LinuxCNC+Mesa gets coordination cheaply but leaves the power stage and tuning maturity gaps. A hybrid closes essentially all of these:

- **Brain**: LinuxCNC + a Mesa FPGA card (7i92/7i96 or similar) as the single coordinating controller — closes X's PID loop in HAL using the linear encoder feedback, generates Y/Z step/dir, and interpolates all three together. This directly fixes the "blind axis" problem of the drive-pair and Mach/grbl approaches, at open-source cost rather than industrial-controller cost.
- **Muscle**: A dedicated linear/voice-coil current amplifier (AMC, Copley, Elmo-class) used purely as a *current amplifier* — i.e., configured so it does **not** close its own PID loop, just amplifies the current command Mesa/LinuxCNC computes. This gives you industrial-grade power delivery, thermal protection, and current limiting without paying for a full industrial motion-controller license.
- **Sensing**: Swap to (or confirm) an absolute (SSI/BiSS) linear encoder rather than incremental-with-index. This removes the homing-routine dependency and restores a known position immediately after any power cycle — mitigating the "position lost on power-loss" issue common to every option above.
- **Safety**: Add a normally-engaged mechanical brake or spring-return-to-safe mechanism on X, since a voice coil has zero holding force when unpowered or faulted — this is a gap in *every* solution in the table, not just the DIY ones, and needs to be solved mechanically regardless of controller choice.
- **Tuning gap**: LinuxCNC's halscope plus a feedforward term calculated from the voice coil's known force constant and moving mass gets you most of the way to what vendor auto-tune gives you, closing that gap without buying into a proprietary ecosystem.

This lands you with genuinely coordinated 3-axis motion, industrial-grade power handling on X, and full visibility into the servo axis's real behavior — at a fraction of the ACS/PI/Galil/PMAC price, and without the "two controllers that don't talk to each other" problem of the drive-pair or Mach/grbl routes.



Purple: control · Teal: drive electronics · Coral: motors · Gray: safety

The dashed teal line shows the feedback path that makes this different from the drive-pair or Mach/grbl approaches: LinuxCNC/Mesa sees X's actual encoder position on every control cycle, so it can coordinate X's motion with Y/Z in real time rather than treating X as a black box. The amplifier and stepper drivers are deliberately kept "dumb" (pure power stages), keeping all the intelligence — and all the tuning — in one open, inspectable place.