Arduino Antenna Rotator Controller — AC & DC, one codebase

This is the rewrite of my original potentiometer-feedback rotator controller — the one a lot of you have already built. Same job as always: read the satellite position from your tracking software over USB and point a 2-axis antenna (azimuth + elevation) at it, using a plain Arduino Uno/Nano and a potentiometer per axis for feedback. No gyros, no compasses — I tried those, they don't hold up. A potentiometer, done properly, is boring and reliable, which is exactly what you want on a mast in the wind.

What's new this time: AC and DC motors now share one codebase and one pinout. You pick your motor type with a single line in the code before uploading — everything else, the tracking logic, the protocols, the display, the safety features, is identical underneath. If you've got a commercial AC rotator (relay-driven, like most of them), or you're building your own with a DC gear motor and an H-bridge, this is the same project either way.

How it talks to your computer

The controller listens continuously and figures out, line by line, whether your software is speaking EasyComm II or Yaesu GS-232B — no setting to configure, no mode to select. Send it either one and it replies in kind. This covers basically everything out there: PstRotator (my daily driver, and a genuinely excellent piece of software), Gpredict, SatPC32, HRD, WXtrack, MacDoppler. If your software offers a choice, EasyComm II is the more modern option — decimal precision, more information exchanged — but if it only speaks Yaesu, that works exactly as well.

Smooth movement, not on/off jerking

Every move — a serial command, a manual jog, homing, calibration, all of it — ramps up to speed and eases back down as it arrives, rather than slamming on and cutting off. On the DC build this comes from a genuinely simple trick: two power ramps, one based on how far you've traveled since you started moving, one based on how close you are to the target, and the motor gets whichever number is smaller at any given moment. That's the whole algorithm — no PID, nothing to tune blindly. Azimuth and elevation each get their own ramp distance and minimum power, since the two axes are rarely built the same mechanically.

On AC, obviously, you can't throttle a relay — but the tolerance logic is shared with DC, so the motor only bothers moving once the antenna has drifted far enough to matter, and it settles cleanly rather than hunting back and forth around the target.

Direction reversals are handled with real care. If a new target suddenly wants the antenna going the opposite way while it's driving hard, the controller doesn't just flip the relay or reverse the H-bridge on the spot — it eases the motor down to a stop first, waits a fixed dead time, then takes off the other way. Same protection on both motor types. Your gearbox and your relays will thank you.

Calibration — let the controller find its own limits

You never have to guess where your potentiometer's real electrical range sits. Hold the azimuth encoder button down at boot and the controller drives each axis out to both physical ends, watches the (filtered, smoothed) position reading go flat, and knows it's arrived — no limit-switch wiring required for this, though the hard electrical limit switches are still there for safety regardless. The measured endpoints get written to EEPROM and used from then on, survive a power cycle, and are more accurate than anything you could type in by hand.

A brand-new, never-calibrated board still runs — it just uses safe full-range defaults until you calibrate, and flashes "UNCALIBRATED" once at boot as a reminder, so nobody's left wondering why the numbers look a bit off before their first run.

Also configured once, at the top of the code, before you upload:

- Elevation range: 0–90° or 0–180°
- Azimuth range: 360°, or 450° if your rotator has the extra mechanical overlap — the pathfinding logic takes full advantage of that overlap to avoid unnecessary reversals
- Azimuth zero reference: North or South, for those of you in the southern hemisphere who asked for it
- Your parking position — wherever the antenna sits happiest with the least wind loading when you're done for the day

Configuring your build

These are plain lines near the top of the sketch — no menu, no button sequence, just edit and re-upload once, before you ever power the thing on:

```
#define MOTOR_TYPE        MOTOR_AC     // MOTOR_AC or MOTOR_DC
#define ELEVATION_MAX     90           // 90 or 180
#define AZIMUTH_MAX       360          // 360 or 450
#define STOP_REFERENCE    STOP_NORTH   // STOP_NORTH or STOP_SOUTH
```

• `MOTOR_TYPE` — pick the schematic that matches your build, further down this page.

• `ELEVATION_MAX` — 90 for a rotator that only tilts up to the horizon-to-overhead range, 180 if yours can look past vertical to the far horizon.

• `AZIMUTH_MAX` — 360 for an ordinary rotator, 450 if yours has that extra 90° of mechanical overlap (common on some commercial units) — the controller uses the overlap automatically to avoid unnecessary reversals.

• `STOP_REFERENCE` — which way your rotator's hard mechanical stop faces. North is the usual case; South is for builds where the stop was deliberately mounted the other way round.

Everything else — tuning constants, timing values, the parking position — lives in the same block, but these four are the ones that actually describe your hardware.

The display

Top row is always where the antenna actually is, live. Bottom row is the target, with an arrow showing which way it's currently driving: → / ← for azimuth, ↓ / ↑ for elevation, and a plain " =

" once it's arrived and sitting still. You can send a new target, or just turn an encoder, at any moment, mid-move or not — the antenna simply changes course, no waiting for it to finish the old command first.

Normal tracking: antenna at 123.4°/45.6°, heading to 123.0°/45.0°

A few small marks worth knowing: a "*" next to the azimuth value means you're jogging in the coarse 10°-per-click mode instead of the usual 1°. On a 450° rotator, a "!" shows up when the antenna (or its target) is sitting past the traditional 360° mark, out in that extra mechanical overlap — the number displayed is still a sensible compass bearing, the "!" just tells you it's being reached "the long way around" through the overlap zone rather than the short way.

450° overlap zone (!) with azimuth in x10 jog mode (*)

Both special marks at once: overlap-zone flag and coarse-jog flag

Quick reference for every symbol you'll see on the bottom row:

- → ← azimuth driving clockwise / counter-clockwise
- ↑ ↓ elevation driving up / down
- = axis has arrived, motor stopped
- T normal label in front of a target value
- * replaces T on azimuth while x10 jog mode is active
- ! target or position is in the 360°–450° overlap zone (450° rotors only)

Calibration screen

What you'll see on the LCD while holding the azimuth button at boot, and once it finishes. Each leg is labeled as it runs — AZ+, AZ-, EL+, EL- — so you always know which direction is currently being measured:

Calibrating azimuth clockwise (each leg is labeled: AZ+, AZ-, EL+, EL-)

Calibration complete

Manual control

Both rotary encoders are live all the time, computer connected or not. Turn one, and that axis's target updates immediately.

- Azimuth encoder: rotate to jog, press briefly to toggle between 1° and 10° per click — handy if you're an old-school HF operator swinging the beam a long way for a terrestrial contact, not just chasing a satellite one degree at a time.
- Elevation encoder: rotate to jog, press to send the antenna straight to your parking position — from anywhere, even mid-move.

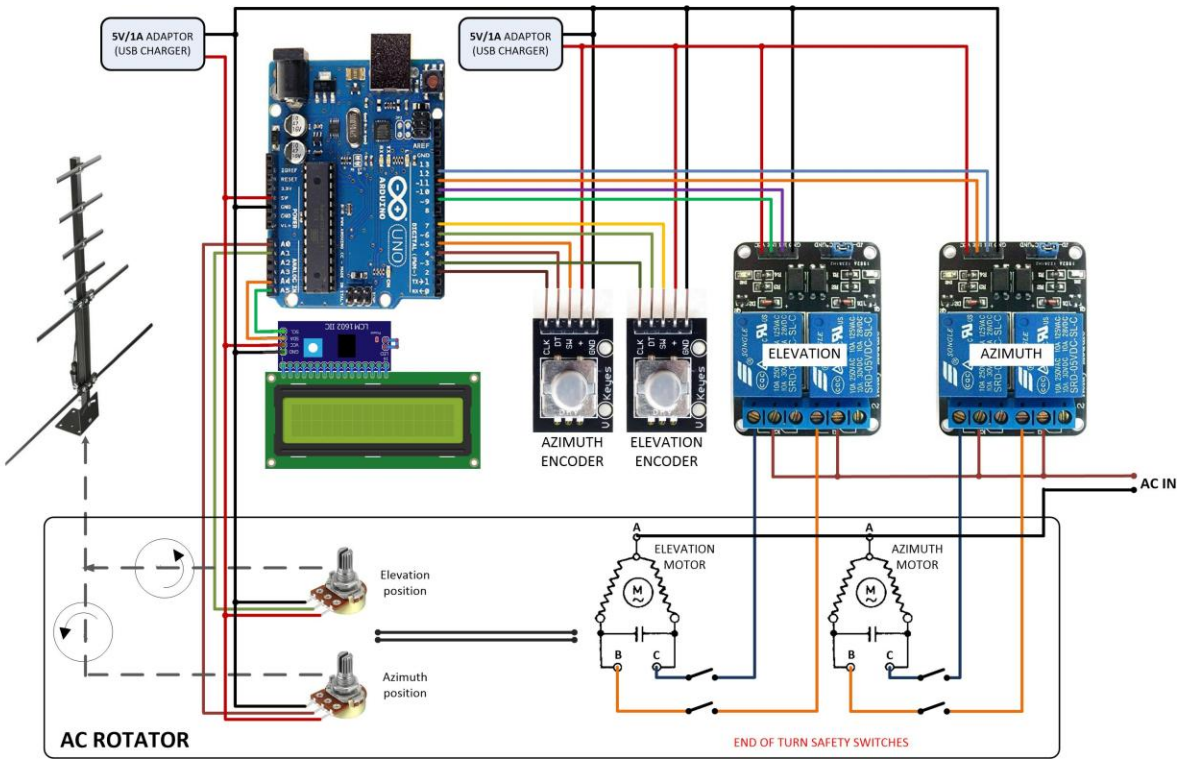
Safety

Physical end-of-turn limit switches are wired entirely in hardware — the Arduino has no say over them and can't override them. Each one only blocks motion further in that direction; reversing off a limit always works.

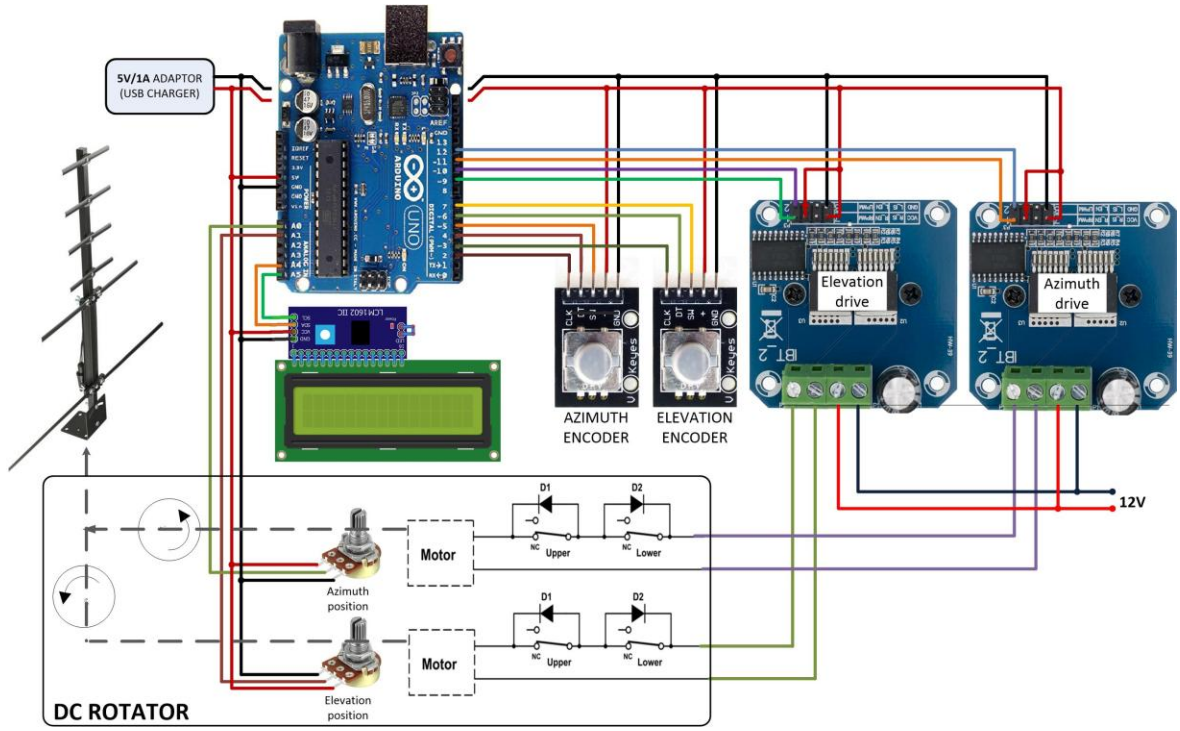
On top of that, the firmware watches for a commanded move that isn't producing any actual movement — a seized gearbox, a blown fuse, a wire that came loose — and if nothing's happening within a reasonable window, it cuts power and shows "AZIMUTH ERROR" or "ELEVATION ERROR" rather than quietly grinding away. Send a new command (or just turn the encoder) and it tries again automatically.

Two builds, one project

The schematics below show both variants side by side — relay-driven AC on one, H-bridge DC on the other. Same Arduino, same LCD, same encoders, same potentiometers for feedback; only the motor drive stage and one line of configuration differ.



AC version — relay-driven, dry-contact interface, works with most commercial rotators



DC version — H-bridge driven, full variable-speed control

Notes

1. Azimuth-only builds are fine — just skip the elevation hardware and put pin A1 to GND for elevation always zero.
2. Re-run calibration any time you change the elevation range, azimuth range, or north/south reference in the code — those settings change what the stored numbers mean.

Full requirements and design notes are documented alongside the code, if you want to see the reasoning behind any of the above.

Questions, bug reports, successes or "it works but here's what I'd change" — drop me a mail to yo3rak@gmail.com
Happy satellite hunting!

73, Viorel — YO3RAK