Antenna Rotator Controller Build

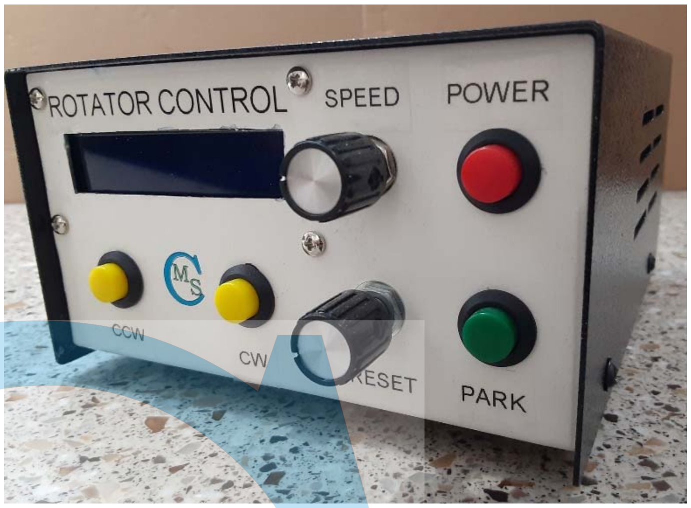


Figure 1 - Rotator controller

It is kind of funny how things work out. I bought a missile-launcher type of mast for the specific purpose of supporting my new hex-beam 6-band folded-dipole antenna. The antenna covers 20-meters, 17-meters, 15-meters, 12-meters, 10-meters, and 6-meters, with the driven elements of each band in a "W" shape, and the reflectors arranged in an open half-hexagon opposite the driven elements. Each of the band elements occupies a specific slot in the concentric hexagons, with the longest (20-meter) naturally to the outside and the shortest (6-meter) to the center. This antenna is highly directional, which brings us to the topic of this article.

For best results, a directional antenna requires a means of turning or "aiming" the antenna in the desired direction, that being the direction of maximum forward gain. Following the very good advice of fellow club members **Frank Romeo N3PUU** and **Al Arrison KB2AYU**, I designed a two-level rotator motor support system, with the rotator motor mounted to the lower-level platform, while the upper-level plate holds a hefty thrust bearing intended to support the antenna mast against lateral forces such as heavy wind gusts.

The rotator motor itself is a Yaesu G-800-series unit, which operates on voltages up to 24VDC, with a 5VDC sensing circuit that outputs a voltage from 0VDC to 4.5VDC, proportional to the rotator's angular position between 0° and 450°, a full one and one half turns of the mast.

Once the rotator was designed and built, the next logical need was for a unit to control the rotator motor from within the shack. In my situation, the hex-beam is located about a hundred and twenty-five feet from the shack, a not unreasonable distance. What was needed was a controller that could output the required motor voltage of up to 24VDC, with the ability to reverse the polarity of these outputs in order to reverse the motor and therefore the rotator direction. In addition, the unit would need to be able to power and read the control circuit, interpreting the 0-4.5VDC position voltage signal correctly. Furthermore, there was a need to be able to control the rotational speed, ramping it up and slowing it down as required as the limits of the desired travel approach. My desire was for an azimuth-only rotator controller (Figure 1), as I have no need or capability for any elevation adjustment.

Once again, I turned to **Frank Romeo N3PUU** for advice. Frank had built just such a controller for the GCARC Clubhouse HF station. I wanted to duplicate his build for my own station, as it was well-suited to the task at hand. Frank was extremely helpful to me, sharing freely of his design, including providing part numbers where available for the components used in his build.

Frank's design was based on the design published by **Anthony Good K3NG**, a Radio Amateur from Jim Thorpe, Pennsylvania. The K3NG design included far more capability than I needed, so it was a matter of paring it down to just those features that I wanted to include, and then also to make some minor modifications to the programming of the microcontroller (µC) to tailor the operation to my needs.

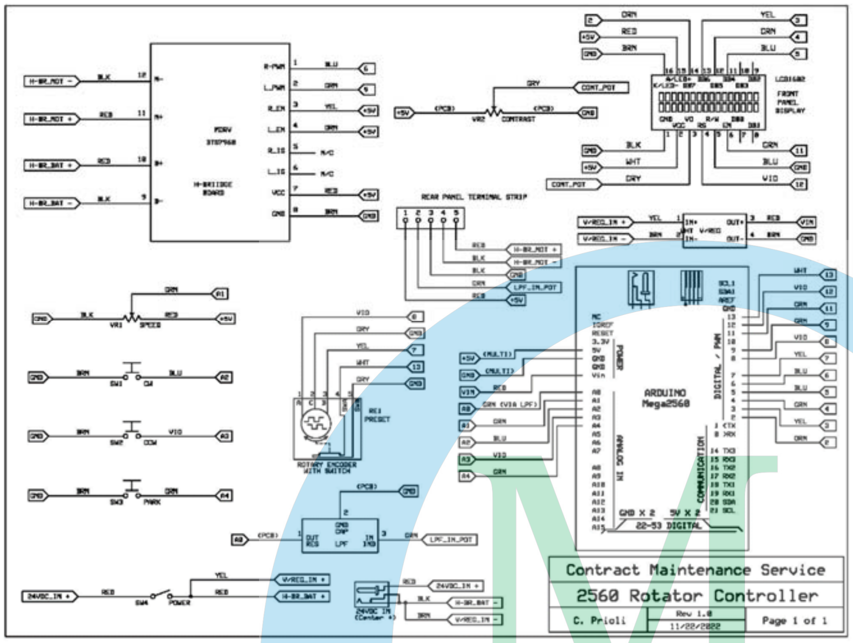


Figure 2 - Rotator controller schematic diagram

The rotator controller, as shown in the schematic at Figure 2, is based on an Arduino Mega 2560 controlling a dual H-bridge motor driver board. A simple 5VDC-output power supply board is incorporated as a part of the unit, and the front panel uses a sixteen-character per row, two-row backlit LCD panel for communication of the unit status. On the front panel, there are controls for parking the rotator, turning the rotator in the clockwise direction, turning it in the counter-clockwise direction, controlling the rotator motor speed, and setting the rotator to a preset angular position. Of course, there is also a power switch mounted there. The rear panel has the 24VDC power inlet jack, the USB type "B" jack for the Arduino board, and the barrier-type terminal strip carrying the leads to and from the rotator motor.

The Arduino Mega 2560 (Figure 3) is used with a blank 2560 shield board (Figure 4), onto which certain specific components have been added. First off, pin headers must be installed along the right and left edges of the board, inserting them into the analog and digital ports from the underside of the board, thus enabling the board to be stacked onto the Arduino Mega 2560. Specifically, a total of fifteen pins (one eight-pin strip and one seven-pin strip) are installed to the right-hand edge, from port 15 up the right edge of the board to port 13. Then, three eight-pin strips are installed to the left-hand edge of the board, including the stretch from port A8 to A15, from A0 to A7, and from VIN up to and including the unmarked port just past the IOREF port. The remainder of the modifications to the shield board are made on top of the board.

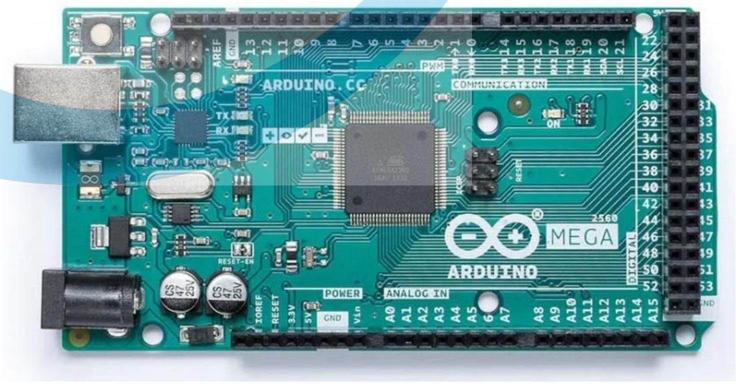


Figure 3 - Arduino Mega 2560

A pin-header row of twelve pins must be installed in a line across the shield board, in alignment with the second GND port on the left edge of the board, directly opposite the p

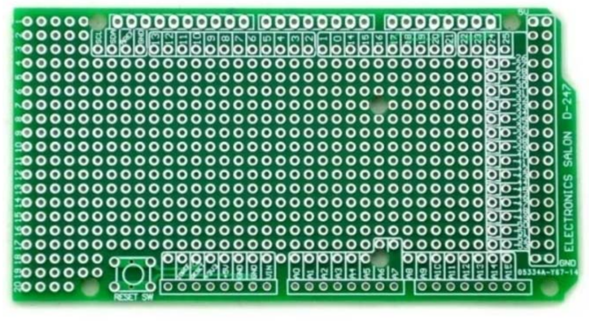


Figure 4 - Mega 2560 shield board

in 6 port on the right edge. Underneath the shield board, all of these pins must be soldered together. The first hole is already tied to the GND port. Next, a similar row of seven pins must be installed in alignment with the +5V port on the left edge of the shield board. These pins too must all be soldered together; again, the first hole is already connected to the +5V port. In a similar manner, single pins are installed in alignment with the VIN, the A0, the A1, the A2, the A3, and the A4 ports along the left edge of the shield board. Along the right edge of the shield board, single pins are installed in alignment with the ports numbered 2 through 13. A lone single pin is installed in the fifth hole inboard from the AREF port hole on the shield board's right edge.

Next up is to install a low-pass filter consisting of a 4.7nF 3kV ceramic disc capacitor, a 250mW 5% 1kΩ carbon film resistor, and a 150µH conformally-coated wirewound inductor. The placement of this filter is not critical, so long as it ties in to the proper port of the Arduino and is properly connected to the external circuit. The way that I installed the LPF is as follows:

- the resistor is formed to a lead spacing of 0.4" and placed with one lead in the first hole next to the port A0 pin, occupying holes 1 through 5 in the A0 row;
- the capacitor is inserted with one lead into hole 6 in the A0 row perpendicular to the A0 row, so that its opposite lead will fall into hole 6 of the first GND row of the shield board;
- the inductor is formed to a lead spacing of 0.4" and placed with one pin in hole 6 of the A1 row, occupying holes 6 through 10 of that row;
- a single header pin is installed into hole 10 of the A0 row;
- the various adjacent component leads are soldered together, thus connecting one end of the resistor to the A0 pin, connecting the RLC joint in the center of the filter, and connecting the pin in hole 10 of the A0 row to the second end of the inductor; and
- the ground end of the capacitor must be soldered to the row of ground pins.

The shield board must also have a provision for adjusting the contrast of the LCD panel. This is accomplished by installing a single-turn trimmer potentiometer into the area between the +5VDC pin row and the lone header pin installed in the AREF row. Install the trimpot so that its single pin side is towards, aligned with, and two rows down from the lone pin in the AREF row, placing that pin of the trimpot into the port 13 row. That pin of the trimpot must be soldered to the single header pin two rows above it. Of the two remaining pins, the one to the left must be tied into the +5V pin row, while the trimpot pin on the right gets tied directly to the GND pin row. The Arduino shield board is now fully prepared, and it is time to start preparing the BTS-7960 43A dual H-bridge (Figure 5) board.

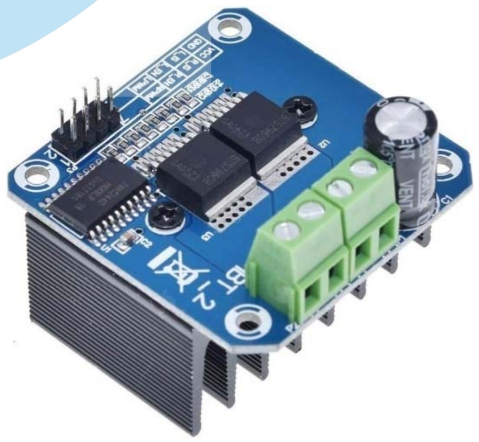


Figure 5 - BTS-7960 dual H-bridge board

At least one and maybe two modifications are required to be made to the H-bridge board. The first mod, necessary in all cases, is to desolder and remove the four-pin terminal block from the upper edge of the board, where the 24VDC current in and motor current out points are located. The easiest way to remove this connector is to first cover the pin area with rosin flux, and then to solder a three-quarter-inch long segment of bare solid 10AWG wire across all four pins of the connector, using plenty of solder to secure the wire to the pins. Then, working with the foil side of the PCB upward and the connector facing downward, simply heat and melt the solder that was flooded onto the pins, keeping the iron moving to melt the entire solder pool. Once the solder is adequately melted, the connector will simply fall out of the board. Clean up the board using solder wick to remove the remaining solder and to open the component pin holes, finishing the job with some 99% IPA to remove the remaining flux.

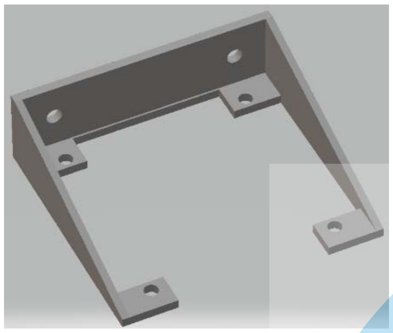


Figure 6 - H-bridge board bracket

The second modification can be done in some different ways. This mod involves the eight header pins at the lower right corner where the communication from the Arduino meets the H-bridge board. The problem, as I saw it, was that these header pins came straight out from the board, which puts them into an interference situation with the H-bridge board bracket (Figure 6) as well as with the edge of the Arduino and its shield boards. I built two copies of this controller (more about that later), and I handled this problem in two different ways on the two builds. In the first build, I removed the straight header pins and replaced them with a 45° header set, to which the requisite wires would simply plug onto just as they would have onto the original straight header. In the second build, I did away with the pin header there altogether, opting instead to simply solder the wire leads directly into the pin header holes in the PCB. This turned out to be the better alternative in the long run. The wires were still plug-ins at their opposite ends, so ease of disconnection for testing, repair or replacement was still maintained, but the connection at the H-bridge board was now more compact as well as being more reliable. The H-bridge board itself is mounted vertically to a custom 3-D printed bracket, which in turn is mounted to the floor of the enclosure, placing the board-mounted heat sink towards the outside of the enclosure and the motor current connections at the top of the board. The mounting position of the H-bridge board is well-considered, as the heat sink channels are thus arranged in a vertical orientation, providing for better cooling with a chimney effect, and the high-current motor leads are at the top of the board, where any heat radiated from them will be unable to do any harm to other components above them.

Because of the fact that the H-bridge requires a +24VDC supply, the unit is powered by an off-the-shelf 117VAC input to 24VDC/3A output power supply. I designed and built a small +5VDC supply that draws off the incoming +24VDC, and I used that to power the control circuitry of the H-bridge board as well as the Arduino. Because I already needed the +5VDC supply as above, there was no sense in *not* also powering the Arduino from that same +5VDC supply, which is capable of delivering a full ampere of current. However, doing so required a little

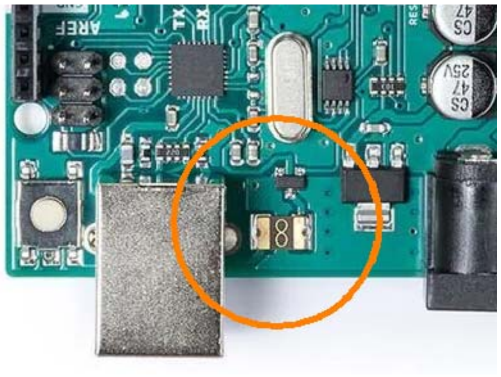


Figure 7 - Arduino fuse to be removed

bit of surgery to the Arduino, as we do not want two competing power supplies to the Arduino, nor do we want the +5VDC from the onboard supply feeding back to the host PC when the rotator controller is connected to a PC for CAT control of the radio system. To that end, the fuse on the Arduino must be removed.

The fuse on the Arduino Mega 2560 (Figure 7) is a self-resetting type and is located directly adjacent to the USB connector shell. Judicious application of heat via a soldering iron will allow removal of the fuse from the board. Do this carefully so as not to damage the board pads or substrate. Note however that once the fuse has been removed, it will no longer be possible to power the Arduino through the USB cable in the usual manner.

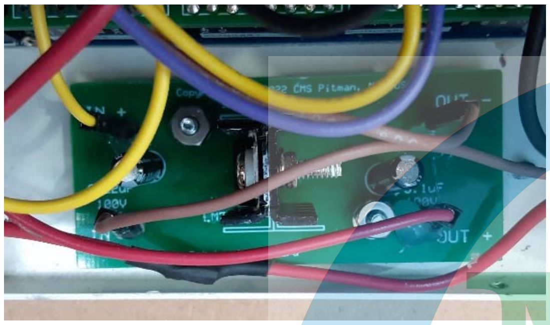


Figure 8 - +5VDC power supply board

Let's move on to the +5VDC supply. This power supply is an incredibly simple contrivance, consisting of three components and a few header pins on a small printed circuit board. The board (Figure 8) carries a pair of electrolytic capacitors – a 0.22µF input capacitor and a 0.1µF output capacitor – as well as a common LM7805 three-pin voltage regulator IC. The board has four single header pins, one each for input voltage, output voltage, and the input and output ground leads. The incoming power to the voltage regulator comes directly from the +24VDC supply via the power switch; the output is routed to the VIN port of the Arduino via that pin on the shield board. The input ground is the negative side of the incoming 24VDC supply, and the output ground is routed to the Arduino GND port on the shield board.

The POWER switch on the front panel (Figure 9) is a latching pushbutton switch of the SPST type. This switch and the other pushbutton switches are all of a single family (DS-228) of switches having commonality in size and physical appearance. Of course, I selected different colors for different functions, to help make switch recognition a bit easier. While all of these pushbutton switches are SPST normally-open types, only the power switch is a latching switch.

The CW and CCW controls are normally-open momentary pushbutton switches, each of which grounds the port to which the switch is connected when the switch is closed (*i.e.*, when the button is pressed). So long as either of these two switches is held closed, the rotator will turn in the direction indicated by the switch that is pressed. The directional view of the rotator is as if it were being looked at from above. The PARK switch is a similar switch, which also grounds its connected port. However, with this switch, pressing it and grounding the PARK port of the Arduino triggers a planned motion which rotates the motor to a specific pre-programmed rotational angle.

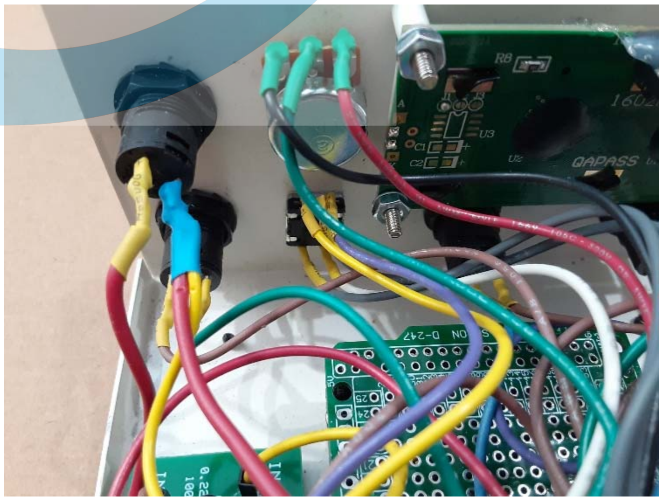


Figure 9 - Rear view of front panel controls

Often, an antenna owner will want to park an antenna so that it offers the least amount of wind load in the direction of the prevailing local wind. Unfortunately, this practice can also result in premature wear to the rotational sensing potentiometer and the teeth of the internal gears due to the constant vibration of the antenna in that one fixed position. In any event, the PARK switch function is to return the rotator to its pre-programmed parking position and stop it there.

The next control up for discussion is the PRESET control. This device is a rotary encoder with an incorporated SPST switch, which is activated by pressing in on the control knob. The encoder is a PEC11-type having thirty detents per revolution and producing fifteen pulses per revolution. Because of the nature of rotary encoders, the direction in which the control has been rotated is readable by the software. The basic K3NG firmware release has this encoder's output producing a turning motion of one-half of a degree per pulse. This is defined in the ***rotator_settings.h*** file at Line 178, which reads *"#define AZ_POSITION_ROTARY_ENCODER_DEG_PER_PULSE 0.5"* which can be modified, as I did, to a more useful value. I set this value to 2.0 degrees per pulse.

Here is what all of that is about. When setting a preset position for the antenna, the procedure is to dial in the azimuth desired by rotating the encoder control knob. While a setting of 0.5°/pulse gives high resolution, it also means that it will take much more spinning of the knob to reach a given value. Suppose, for example, that the antenna is currently aimed to an azimuthal direction of 180°, or due south. If you want to change that to 0° (due north), it will take 360 pulses at the standard degrees per pulse value setting. With the encoder producing 15 pulses per revolution, it will take 360/15 or twenty-four full turns of the encoder knob to dial in that preset position. By changing the value to 2°/pulse, we now have the ability to traverse the 180° of antenna rotation in only six turns of the encoder. That is because the new setting gives us 30° of antenna rotation for each full rotation of the encoder knob or shaft. With a desired change of 180° in antenna position, and 30° of antenna swing per turn of the encoder knob, we get six turns instead of twenty-four turns to achieve a 180° antenna direction change. That makes perfect sense, seeing as how the rate is four times as fast with the new setting. There is one small drawback to this setting, in that the finest antenna position resolution is now two degrees instead of one-half of a degree. If that is objectional, a fair compromise would be to set the *degrees per pulse* setting to *1.0*, which would double the original rate but still give a resolution of one degree.

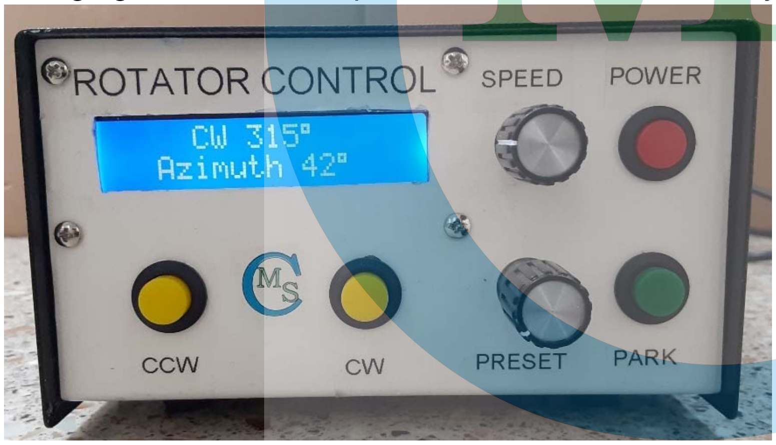


Figure 10 - Rotator controller display operational

Once the desired azimuthal angle is dialed into the controller display window, simply press the encoder knob inward on the panel to initiate the movement of the antenna. The antenna will begin rotating to its new position, will slow down when approaching the set position, and will stop when it gets there.

The SPEED control on the front panel is a 100kΩ single-turn linear potentiometer used to set the maximum rotational speed of the antenna while the motor is turning. Rotating this control imposes a proportional voltage on the Arduino port A1, which is read by the μC software and used to control the speed in accordance with that proportional voltage input.

The front panel display (Figure 10) will show three basic pieces of information, the current azimuthal direction in degrees and cardinal compass directions, the target azimuthal angle in degrees, and the direction of motion, either clockwise (CW) or counter-clockwise (CCW). Once antenna rotation is complete, the display will revert to a simple ***"Azimuth xxx°"*** display, possibly with a cardinal compass direction above it. The display unit is a white-on-blue backlit LCD panel having a 16x2 character by row format. While other LCD panel colors are available, I find the white-on-blue to be the easiest on the eyes, which is why I chose that particular unit. The LCD panel contrast is adjustable via a trimmer potentiometer mounted to the Arduino shield board. In this case, the trimpot is a Kyocera 601040 100kΩ miniature top-adjust potentiometer. It is connected with one end at +5VDC, the opposite end to ground, and the wiper is tied to the V_O pin of the LCD panel, pin 3.

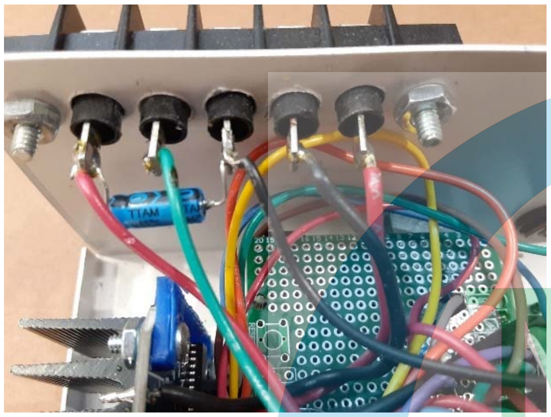


Figure 11 - Rear panel terminal strip

Power entry at the rear of the controller enclosure is made via a 5.5mm x 2.1mm coaxial DC power jack with its center pin being the positive side of the circuit. The positive lead is routed to the front panel POWER switch, and then from there to the H-bridge board's BAT+ pin, and also to the +5VDC supply (voltage regulator) board's power input pin. The negative lead is connected to the H-bridge board's BAT- pin and also to the voltage regulator board's ground input pin.

The voltage regulator's +5VDC output is routed to the VIN port on the Arduino shield, and the ground side of the voltage regulator board is connected to the GND port on the Arduino shield board.

Provision is made on the enclosure rear panel for a USB connection to the Arduino Mega 2560 board, necessary for CAT control of the antenna rotator. The enclosure rear panel also carries a Molex 38721-6705 five-terminal barrier-type terminal strip (Figure 11) with turrets on the inside of the controller. This terminal strip is the communications point between the controller and the rotator motor. Two of the terminals are used by the motor power leads, while the remaining three are tied to the rotator unit's position-sensing potentiometer. A 4.7μF 50V axial aluminum electrolytic capacitor is installed between the +5VDC terminal (Terminal 1) and the GND terminal (Terminal 3) on the terminal strip inside the enclosure.

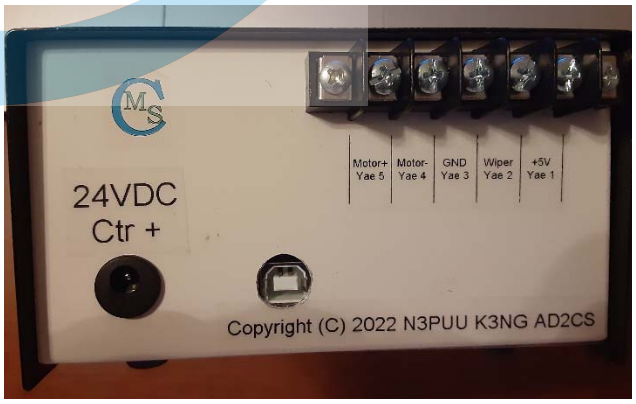


Figure 12 - Rear panel exterior view

The enclosure is a two-piece black steel over white aluminum sheet metal assembly, part number 208911 from Jameco Electronics. It includes ventilation openings in both sides of the upper half of the enclosure. Most of the labeling of this project was handled through the use of black-on-clear printed labels produced by my Dymo Rhino 5200 label printer. The exceptions are the CMS logos on the rear and front panels (Figures 12 & 13), which are water-slide decals printed on my color laser printer.

The Arduino Mega 2560 and its accompanying shield board are secured to the floor of the enclosure via 4-40 hardware and tubular spacers, as is the voltage regulator PCB. The custom 3D-printed H-bridge board bracket is also mounted to the floor of the enclosure, being placed directly on the floor and being secured by 4-40 machine screws with hex nuts and lock washers. The LCD panel is secured to the front panel with 4-40 hardware and tubular spacers.

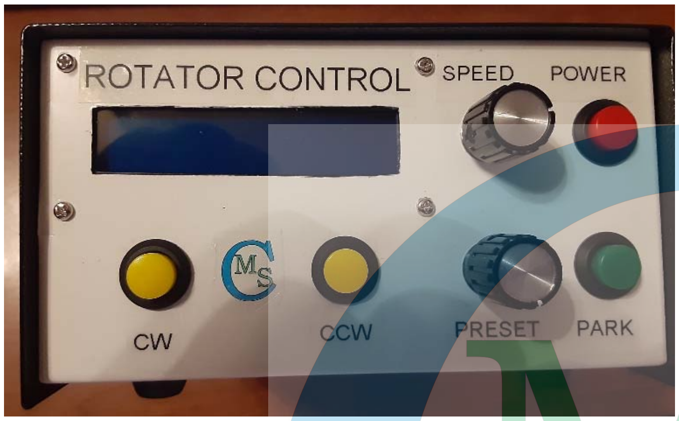


Figure 13 - Front panel exterior view

Most of the interior wiring (Figure 14) is made up of 22AWG stranded hook-up wire with crimped-on terminals to fit the header pins. While this could easily have been done using off-the-shelf Berg or DuPont wires with female connectors, the quality of many of those wires is sketchy at best. As I have the terminals, housings, wire, and crimper all on hand, it was a simple and natural decision to "roll my own" wires. This had the added advantage that I could make each wire in the color and length that I wanted, and I could assemble many of the wires that connected to adjacent pins into single multi-pin plugs. That alone made it much easier and neater to wire things up. However, for added security, I also chose to hot-glue (Figure 15) many of these wire connectors into place once they were all connected properly and the unit was tested.

I mentioned earlier that I have built two copies of this controller, and so I did. The first one that I had built was placed on loan to the GCARC Clubhouse for temporary use in the VHF room. I soon decided to make that loan a permanent donation, which is why I ended up building a second copy of the unit. While they were originally carbon-copies of each other, I have since made some modifications to the controller at the Clubhouse.

One day, during a re-calibration of the controller, the unit went blank, and then started displaying gibberish and impossible values on the LCD panel. I initially thought that what was needed was a simple reload of the operating software to the microcontroller. I hooked the unit up to my laptop USB port and copied the software in successfully, but that did not solve the problem. The unit still booted up to gibberish, and then began showing azimuth angles in the four- and five-digit range. I next brought the unit to my shop, opened it up, and found that the only thing wrong was that the

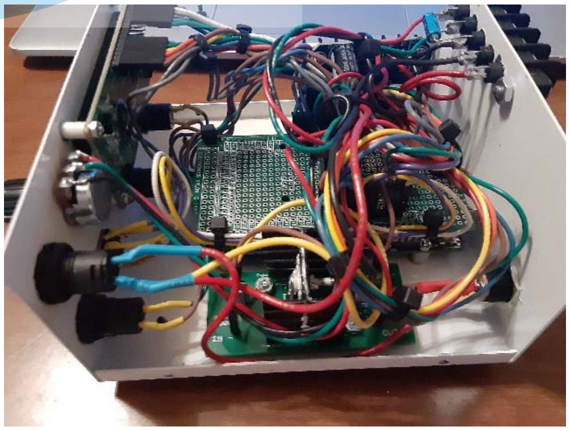


Figure 14 - Interior wiring view

firmware/software in the µC had been corrupted at the bootloader level. I stripped out the Arduino shield so that I could get to the ICSP programming header, burned a new bootloader to the board, and then re-installed the software. This time, it was successful, and the unit operated as it should, and I did a successful calibration of the unit. I returned it to the Clubhouse, putting it back into service.

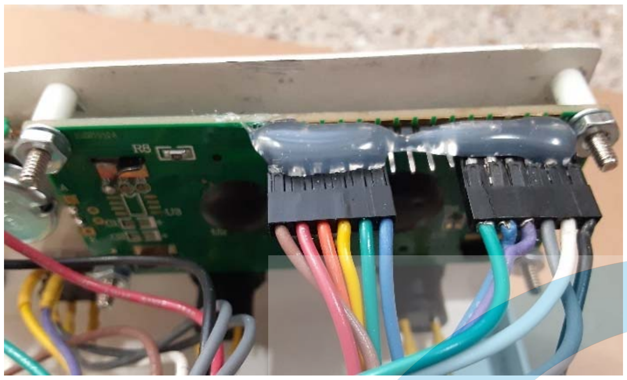


Figure 15 - Hot-glued wiring connectors

A week later, the same problem occurred, again during another attempted re-calibration. Again, I brought it home and stripped it down to the Arduino so that I could once again burn a new bootloader to the µC. Again, the repair took and the unit worked properly, *and I was able to calibrate it with no problem.* Naturally, I brought it back to the Clubhouse and placed it back into service.

Once again, the same problem occurred. This time, I decided to do something just a little bit differently. I completely disassembled the unit, removing the Arduino Mega 2560 from the unit. I then de-soldered and removed the ICSP header, replacing it with a six-position pin socket of the same configuration. Next, I added a set of long header pins to the shield board in alignment with the pin socket on the Arduino board, in such a manner that the pins would mate into the new pin socket when the shield was installed back onto the Arduino, and that the upper ends of these six pins were available to be used as a pass-through ICSP header on the shield. I re-assembled everything, and then tested the job by burning a new bootloader through the new ICSP header pins, and then afterwards installing the software through the USB in the normal manner. My reasoning was that while I did not know why the calibration process was corrupting the boot loader, at least I could re-burn the bootloader without having to disassemble the boards to get access to the ICSP header.

I returned the unit to service, and this time I assisted in the calibration process. For whatever reason, this time the calibration went off without any problems. That unit is still operating there at the clubhouse as it was designed to do.

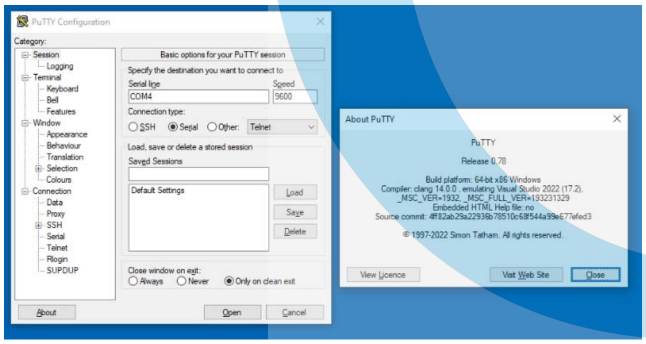
The reason that I mention all of this is that it points out a valid modification that any builder might want to make to the Arduino Mega 2560 and the shield board. It is probably a good idea to make the ICSP header accessible by simply removing the controller cover, and without the need to disassemble the unit any further.

The software for this controller is readily available online – just Google *"K3NG rotator controller"* to find it. Modify the ***rotator_pins.h*** file as necessary in accordance with the file notes, and also modify the ***rotator_settings.h*** file of you want to change the encoder resolution. The *Arduino IDE* software is required for compiling and uploading the programming to the Arduino Mega 2560. This software is available online at no charge.

Take your time with the wiring to ensure proper connections of the various Arduino ports with their associated inputs and/or outputs. Also, take your time with making the necessary openings in the enclosure panels to get the best possible appearance. The large opening for the LCD panel was made by drilling a series of adjacent holes, cutting away the waste between the holes, and then filing the opening to size with the aid of a piece of square metal-lathe bit stock clamped along the openings, one at a time, in turn around the opening. The file will cut the aluminum, but will glide right over the tool steel bit stock, leaving a nice clean edge and a squared-off opening. This tip, too, was thanks to **Frank Romeo N3PUU**. He is a wealth of information!

All of the other openings cut in the enclosure panels were simple round holes, easily drilled. However, there is another tip worth mentioning, which I use regularly when I am drilling holes of more than about an eighth of an inch in diameter into sheet metal on the drill press. Take a square of old cotton cloth, *e.g.,* a scrap from an old tee shirt, about two inches on each side. Center-punch the hole center lightly. Then, fold the cloth scrap in half twice, producing a four-ply cloth stack about an inch square. Place the folded cloth scrap over the hole location, and hold the cloth in place while you start the drill motor. Bring the twist drill bit down onto the cloth and then, using light pressure, drill through both the cloth and the sheet metal. The result will be a nice round and clean hole in the sheet metal, without any grabbing of the metal by the twist drill and therefore no spinning of the metal in the drill press. I use this trick quite often – it works every time. While step drill bits will also make round holes without the grabbing and spinning, the hole diameter selection from a step bit is nowhere near as wide as the offerings of my entire half-inch drill index. Another advantage of this tip is the fact that there will usually be little or no burr around the hole on the back side of the metal sheet. When using this tip, cut and fold a new scrap of cloth for each hole that you drill.

Any way you look at it, this project was one that was well worth taking on. I enjoyed the build process both times. I have not yet added the ICSP header to the copy of the controller in my shack, but if I ever need to open up the controller to reload the software, I will make the modification at that point in time.



Specify the destination you want to connect to
Serial line: COM4
Speed: 9600
Connection type: SSH [✓ Serial] Other: Telnet
Load, save or delete a stored session
Saved Sessions
Default Settings
Close window on exit: Always Never [✓ Only on clean exit]

PuTTY
Release 0.78
Build platform: 64-bit x86 Windows
Compiler: clang 14.0.0 , emulating Visual Studio 2022 (17.2), _MSC_VER=1932, _MSC_FULL_VER=193231329
Embedded HTML Help file: no
Source commit: 4ff82ab29a22936b78510c68f544a99e677efed3
© 1997-2022 Simon Tatham. All rights reserved.

*Figure 16 - **PuTTY** user interface*

Calibration of the unit is necessary to synchronize the display to the actual azimuthal angle of the antenna. This is easily done through the use of the *PuTTY* software package. *PuTTY* will launch a terminal window that is in communication with the controller. We start by issuing a *"C"* command to the controller, which returns the current azimuthal angle. The controller then prompts the user to manually rotate the antenna fully clockwise, after which the *"F"* command is sent to the controller. That command causes the controller to store the fully clockwise position on non-volatile memory, after which the user issues the *"O"* command. The controller now prompts the user to manually rotate the antenna fully counter-clockwise. With the antenna fully counter-clockwise, the user presses the *"ENTER"* key to store the offset into non-volatile memory and complete the calibration process. In order to properly synchronize the calibrated controller to the antenna, the assembly should be so mounted that when the controller is at its zero-degree (0°) position, the antenna is pointing due north. In this circumstance, the zero-degree and fully clockwise position should then coincide at due north. The controller provides for an additional half-turn of azimuthal overlap in its control capability, as does the Yaesu G-800 rotator motor. The main *PuTTY* user interface and its *"About PuTTY"* window are shown in Figure 16.

PuTTY is readily available online as freeware under the MIT open-source license, and is downloadable both as executables and as source code. Executables are available for 32-bit versions and both Arm and x86 64-bit versions of Windows MSI installers, as well as being offered as a Unix source archive. As of the time that this is being written, the current version of *PuTTY* is version 0.78, which was released on 29 October 2022. The current version of *PuTTY* can be downloaded at https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html at any time.

A commonly-used PC interface for controlling the antenna rotation is a nice piece of software called *PST Rotator.* Because I do not have an elevation component to this rotator, I use the *PST Rotator AZ* version of the software (Figure 17). This software is extremely useful and flexible, and it also includes an interface for fine-tuning the antenna offsets. Several presets can be defined and then the antenna can be swung to those points at a single click of the mouse. While not free, the cost is very reasonable and well worth the few dollars charged. I recommend this software to anyone using automated antenna rotation systems. It plays quite nicely with most radio CAT systems.

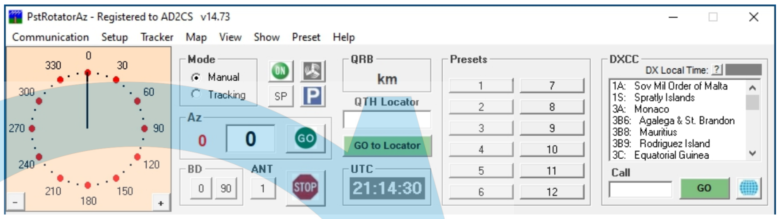


*Figure 17 - **PST Rotator** user interface*

I am happy with the results of this project, in both of its completed devices. Even if the basic design is not my own, and even if I copied the construction details of the unit that Frank had built, enough of my own work went into this project that I do not feel ashamed in writing it up. The idea is to encourage others who may need such a controller to home-brew one as I did. Feel free to copy my work… it is freely shared as was the work of those who went before me. It should also be noted that in my labeling of the unit, where I usually indicate that the device is a product of my shop, this time I also gave props to both N3PUU and K3NG, as is only right.

Feel free to reach out to me at chris@ad2cs.com with any questions or for help with any aspect of the build of this rotator controller.