

Rotator control unit with Arduino Nano

Ing. Ján Uhrin, OM2JU



- Powered by 13.8V from the power supply to the transceiver
- Controls DC motor of rotator; In my case rotator has 24V DC motor, therefore using boost/converter to generate 24V from 13.8V
- Displays current azimuth and other configuration data on 2x16 character display
- Easy operation and configuration using rotary encoder
- System menu and saving of important parameters to EEPROM
- Alternative control via serial port
- In Idle mode powered only by USB from PC/Notebook. With user outputs A, B relays can be controlled, e.g. to switch on 13.8V power supply or control the antenna switch

Arduino code development in Visual Studio Code

I have used Microsoft Visual Studio Code to implement the code for Arduino Nano:

- Download and install Arduino IDE (<https://www.arduino.cc/en/main/software>) - this is needed for background compilation of the code
- Install the latest version of Visual Studio Code (<https://code.visualstudio.com/>)
- Run Visual Studio Code and add extension Arduino for Visual Studio Code (author Microsoft)
- Configure the type of Arduino board, the COM port through which we load the program, etc. It is possible either through the status bar at the bottom right or by editing the *arduino.json* configuration file

Projects should be placed in an existing directory of the Current user / *Documents / Arduino*, ideally generate a simple example from the original Arduino IDE and modify this one. Furthermore, if the project uses existing libraries, these must be downloaded in the original Arduino IDE, for example the TimerOne library, which will be mentioned later. If there are problems with some libraries, it is necessary to check / add the correct paths in the *c_cpp_properties.json* configuration file.

```
Rotator_Counter.ino  c_cpp_properties.json  arduino.json
Rotator_Counter.ino  setup()
365 //
366 digitalWrite(USER_OUT_1, 0);
367 digitalWrite(USER_OUT_2, 0);
368 digitalWrite(DISPLAY_BG, 1); // We want to have display background ON
369 Rotator_Motor_Control(S_MOTOR_OFF);
370 //
371 // Initial state
372 rotator_state = S_SHOW;
373 //
374 // LCD setup and welcome message
375 lcd.begin(16,2); // setting LCD as 16x2 type
376 disp_line(0, VERSION_INFO);
377 disp_line(1, VERSION_INFO2);
378 delay(1500);
379 //
380 // Read config data stored in EEPROM
381 eeDataB0 = EEPROM.read(128); // Ebyte(128) = Data is valid or not ('R' = valid)
382 if (eeDataB0 == 'R') { // found valid rotator init data
383     pulse_cnt_max = EE_rd_32b(0);
384     RotaryEnc_Rotator_angle_deg_min.cntVal = EE_rd_32b(4);
385     RotaryEnc_Rotator_angle_deg_max.cntVal = EE_rd_32b(8);
386     RotaryEnc_Rotator_motor_delay.cntVal = EE_rd_32b(12);
387     calc_rotator_data();
388     rotator_data_valid++;
```

Figure - Visual Studio Code example, my favorite dark display theme , eye-friendly

Interrupts

I have long lived in the belief that Arduino libraries are so simplified that they do not allow use of interrupts. Interrupts are very important for modern microprocessor systems, only by means of interrupts the dynamics and reasonable response of the system to external events can be achieved. In the rotator control, we will use interrupts in two places:

1. Counting pulses from incremental encoder on the pin D2 rotator

Pin D2 is configured so that a change in the input from 0 to 1 (rising edge) causes interrupt. The interrupt handler routine only increases or decreases the pulse-counter based on the rotation direction which we know.

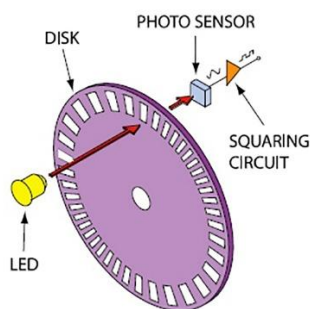


Figure – Principle of incremental encoder

2. Serving of the quadrature rotary encoder which is used to navigate the menu / increment-decrement values / also has the confirm pushbutton

On the output of the encoder are two quadrature signals phase shifted by 90 degrees. Their relative phase is either +90 or -90 degrees depending on the direction of rotation. As in the case of 1 we could use interrupt on rising edge of the A output, while in the service routine checking the value of B output to determine direction. However, with mechanical contacts there is substantial noise on outputs, this would lead to problems. Solution is to use lowpass RC filter on A and B outputs, or to use software solution described below.

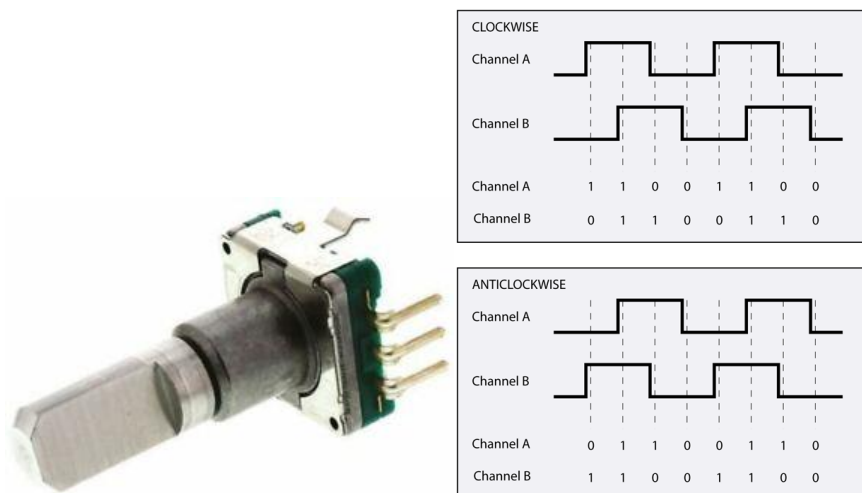


Figure – Quadrature encoder construction and output signals A and B

A possible solution is to sample the output signal fast enough to be able to detect the leading edge of one output and at the same time slow enough to filter out the noise. For this purpose, a Timer interrupt is required to execute the interrupt routine with a defined period. I found an elegant solution in library TimerOne, you need to download

this in the original Arduino IDE. I set the regular interrupt from the timer to 1.5 milliseconds, in the service routine I check value of output A, and if I find pattern 0001 (a rising edge with filtered noise) I check the status of the other output to determine the direction of rotation. Moreover, this solution allowed me to realize a dynamic increase by the addition / subtraction rate depending on the rotation speed. This is practical if we need to quickly change the azimuth by a larger value, we simply rotate the knob faster.

Components used to sense rotation of motor

I have used single channel incremental encoder with photointerrupter e.g. TCST1230 from Vishay. My goal was to use only single wire to get the pulse information from rotator to hamshack. As the DC motor of rotator is powered by either negative or positive voltages depending on direction, I had to rectify the DC motor voltage using diode bridge. Series resistor value for diode in photointerrupter needs to be properly calculated.

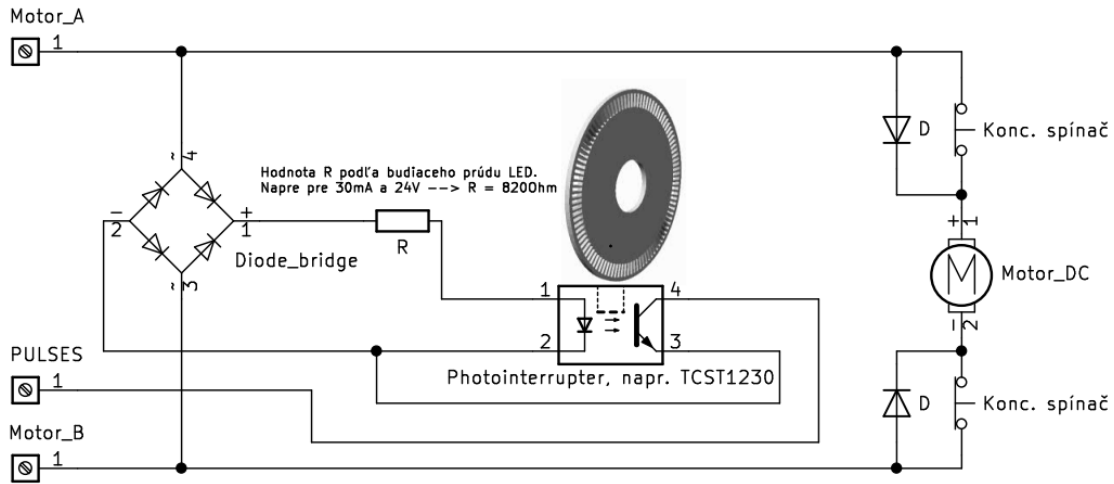


Figure – Pulse counting in rotator

Down in hamshack in Arduino control unit I use additional optocoupler (FOD817 from OnSemi) to avoid problems with overvoltages on long wires and to protect the microcontroller input. The FOD817 needs 20mA for driving of the IR diode, thus with 5V power supply and subtracting forward voltage on IR diode of 1.2V we get roughly 150Ohm for series resistor.

Control of the rotator DC motor

The simplest and most robust solution is using relays in H-bridge topology. For control by logic signals from microcontroller additional NPN or MOSFET-N transistors are needed. Don't forget antiparallel diodes on relays to protect the transistors from damage caused by spikes during switching-off.

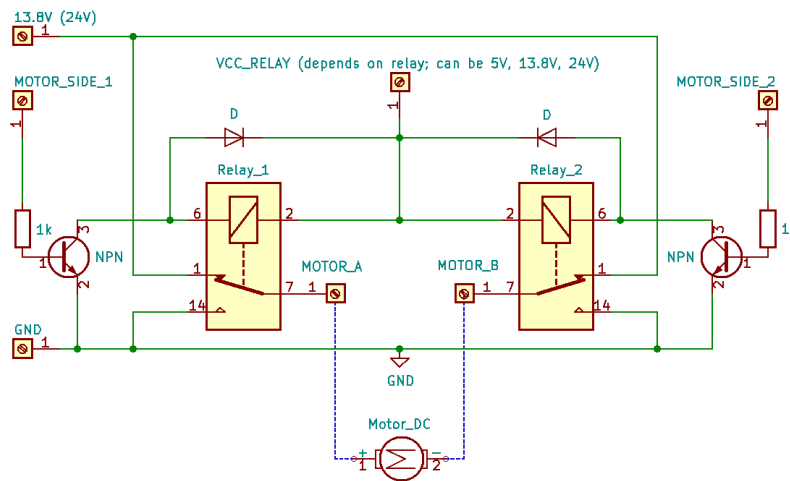


Figure - H-bridge implemented using relays

Today, semiconductor H-bridges are popular too, they have many protection features and allow much faster control. Simple and very popular device for stepper motor control is L298N from ST Microelectronics, it contains two independently controllable H-bridges rated to switch max. 45V / 2A. If more (double) current is needed they can be connected in parallel. I have used ready module with L298N which can be found on eBay – „Dual H Bridge DC Stepper Motor Drive Controller Board Module L298N“.



Figure - Dual H Bridge DC Stepper Motor Drive Controller Board Module L298N

This module has populated protection diodes which protect sensitive high power transistors against spikes. In addition, this module contains linear regulator 78M05 which I use to power Arduino Nano board. However, be careful with maximum allowed voltage for this regulator. For powering of DC motor (thus L298N too) I use 24V which would damage this regulator, therefore I have disconnected J8 and connected 13.8V to J8-2 pin (figure below).

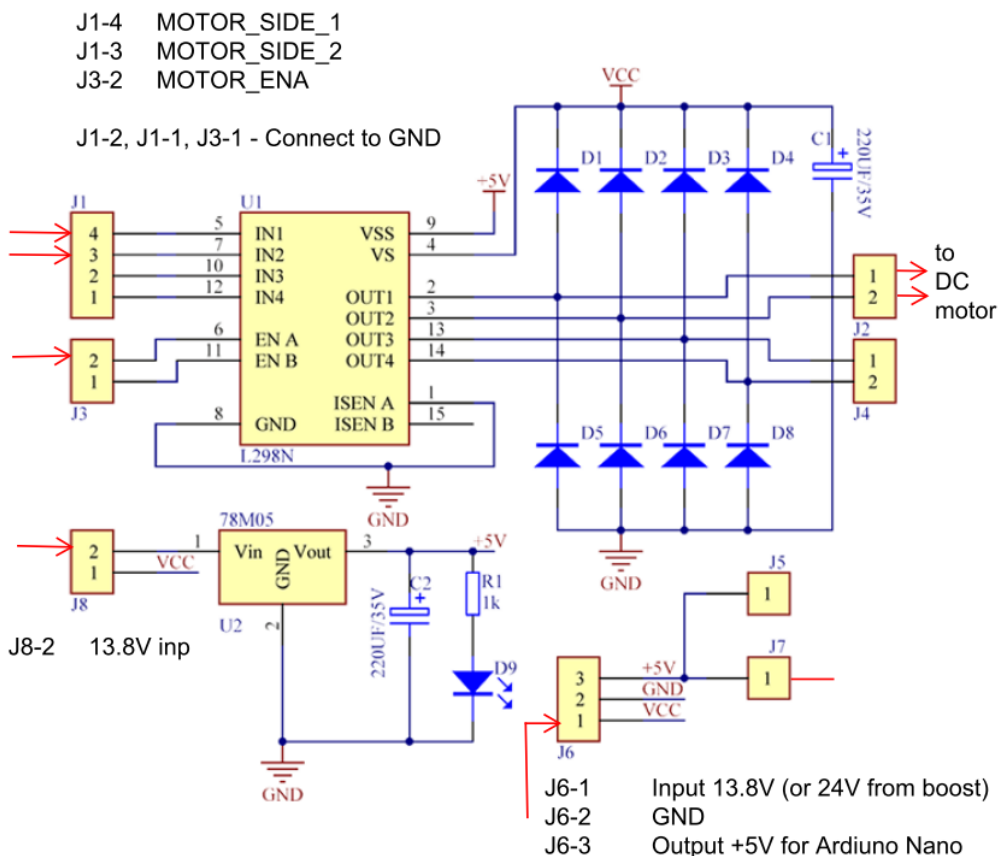


Figure – Schematic of Dual H Bridge DC Stepper Motor Drive Controller Board Module L298N

Each of the two H-bridges in L298N has separate enable signal which, if inactive completely disconnects the power part, thus power is saved. If only one H-bridge is used the Enable signal of the second one should be connected to ground.

Boost converter 13.8V → 24V

I need 24V for rotator DC motor. Again, I have found on eBay ready boost converter module XL6009 capable of boosting up to 35V with current of 2A. Output voltage is adjusted by multiterm potentiometer, the switching frequency is rather low, no problems with EMI to my TRX.



Figure – Module with boost converter XL6009

Brief description of Arduino schematic and function

As the schematic is simple I have used detachable wires to connect Arduino with rotary encoder, display and H-bridge.

The Arduino Nano pins have the following functions:

- D2 – pulses from incremental encoder; isolated by FOD817 optocoupler. On the output side I have used around 1k resistor to connect open collector to 5V.
- D3, D4 – A, B inputs from incremental encoder; pullup resistors are not needed
- D5 – input from rotary encoder push button; pullup resistor is not needed
- D6 – user output B – with transistor I'm switching 5V relay to turn on the power for 13.8V TRX supply. Don't forget antiparallel diode around relay. Note that the relay must be 5V as initially during power down Arduino board is powered only from USB (5V).
- D7, D8, D9, D10, D11, D12 – connection to 2x16 character LCD display used in 4-wire setup (RS, E, D4, D5, D6, D7). R/W input of LCD display is connected to ground as only write to display is executed. Setting of the display contrast is achieved with voltage divider
- D13 – Controls backlight of the display; in case of longer inactivity backlight goes off
- A0 – H-bridge control, side 1 (left)
- A1 - H-bridge control, side 2 (right)
- A2 – Enable for H-Bridge
- A3 – user output A; similar to user output B, however in my case I'm controlling 24V relay of antenna switch

Pins A4, A5 can be used to extend user outputs, in that case software needs to be slightly extended too.

Pins A6, A7 – due to limitation of microcontroller these can be used only as analog inputs; thus not used in our case.

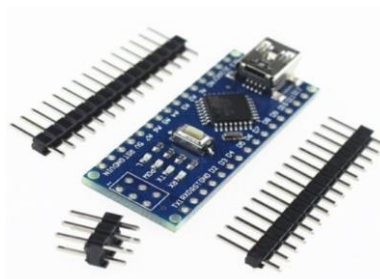


Figure - Arduino Nano board

Initial powering of Arduino board is via USB, thus it is always ready to communicate over serial port. Idea of using this control unit with serial port is following:

- PC/Notebook sleeps, only USB to which Arduino board is connected is powered
- Wake-up the PC/Notebook remotely
- Using serial console I turn on the User output B – 13.8V for TRX and Arduino Board is available
- Now rotator can be controlled as it has 24V. Similarly with antenna switch.
- Of course, local control of rotator using rotary encoder is always possible, information is displayed on connected terminal

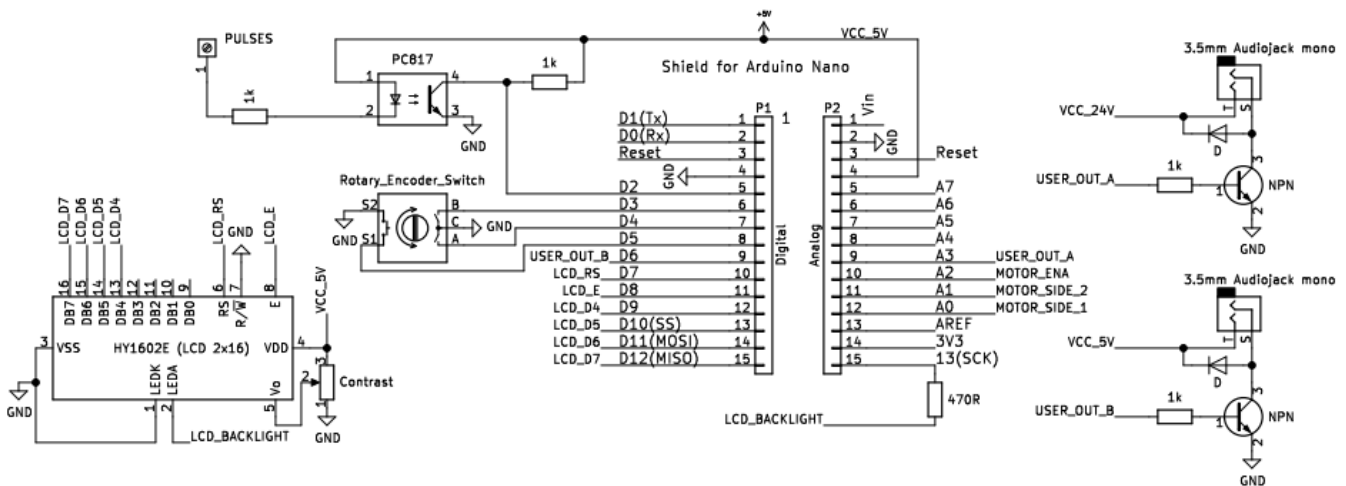


Figure – Components around Arduino Nano board

Arduino Nano board is not ideal solution to achieve extremely low power consumption, main reason is FTDI USB-to-serial converter which is always powered. To decrease the power I disconnected all unnecessary LEDs. I have noted that excessive current might be drawn by unpowered H-Bridge and/or Boost converter, therefore I have used power diodes to isolate power of these domains.

Brief description of software

Basic configuration parameters of the system are left-stop and right-stop azimuths. These are determined by stop switches / mechanical setup of the rotator and are also determining rotation range. In my implementation it can be from 180 to 360+180 degrees. During calibration, user turns the rotator to left-stop position, enters the left stop azimuth, initializes the pulse counter, turns to right-stop position and enters the right-stop azimuth.

Summary of the configuration steps:

- | | |
|---------------------------------------|---|
| - Turn to left-stop position | turn rotation knob once to show L<< and confirm |
| - Enter left-stop azimuth | push rotation knob while „M“, select menu 5. LEFT-STOP AZ |
| - Initialize pulse counter to 0 | push rotation knob while „M“, select menu 1. INIT PULS-CNT |
| - Turn rotator to right-stop position | turn rotation knob once to show >>R and confirm |
| - Enter right-stop azimuth | push rotation knob while „M“, select menu 6. RIGHT-STOP AZ |

During the last step all configuration data, including the value of pulse counter, are stored in EEPROM memory. If EEPROM is empty the display shows alternatively normal menu and alternative menu showing some parameters: L – left-stop azimuth, R – right-stop azimuth, CNT – pulse counter, Cmax – max. value of counter on right-stop azimuth. It is possible that wrong azimuth values are shown as these are calculated from non-valid or non-actual parameters. The parameters can be invalidated by menu item **9. ERASE EEPROM**.

As incremental encoder gives only information about relative position it is not possible to determine actual position after power-up. Therefore I have added menu item **2. STORE ACT. Az**, this value is used during power-up, it can be considered as parking position of the antenna. During normal operation it is always possible to re-calibrate the rotator by turning it to left-stop position and initializing the counter by menu item **1. INIT PULS-CNT**

Other menu items explained:

7. SENSE DELAY - delay in milliseconds during which pulse evaluation is ignored in order to avoid false detection of stuck rotator state. Note that pulses are counted in all cases, only their evaluation is delayed.

8. DEBUG 0/1/2 - switching between various debug levels.

Menu items **3. OUT-A TOGGLE** a **4. OUT-B TOGGLE** re toggling user outputs A or B

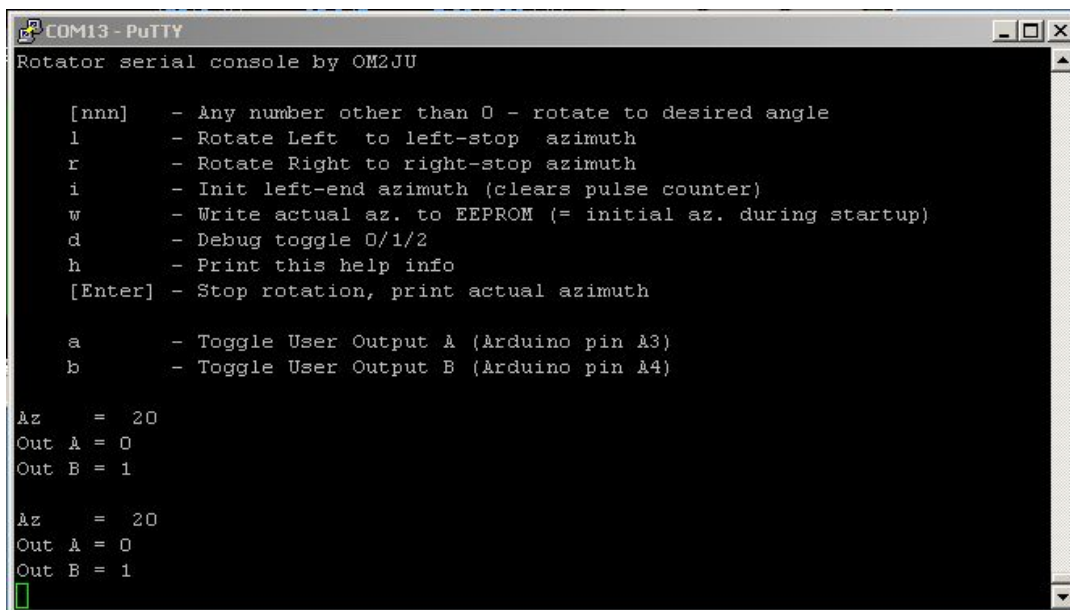
The last menu item **10.EXIT MENU** leave menu and returns to default view where actual and required azimuth are shown. In upper right corner actual state or intended state is shown:

MENU	- enter to menu
>>R	- turn to right-stop position; rotation can be interrupted by pressing push button
L<<	- turn to left-stop position; rotation can be interrupted by pressing push button
>Az	- turn to required azimuth; rotation can be interrupted by pressing push button
Az<	- turn to required azimuth; rotation can be interrupted by pressing push button

Serial console to control rotator

I have implemented simple interpreter to allow rotator control over serial console. If any number different than 0 is entered followed by [Enter] rotator will turn to given azimuth. During rotation actual azimuth is shown periodically, rotation can be interrupted by pressing [Enter].

Other commands are executed by entering single letter followed by [Enter], by pressing 'h' practical Help is shown:



```
COM13 - PuTTY
Rotator serial console by OM2JU

[nnn] - Any number other than 0 - rotate to desired angle
l      - Rotate Left to left-stop azimuth
r      - Rotate Right to right-stop azimuth
i      - Init left-end azimuth (clears pulse counter)
w      - Write actual az. to EEPROM (= initial az. during startup)
d      - Debug toggle 0/1/2
h      - Print this help info
[Enter] - Stop rotation, print actual azimuth

a      - Toggle User Output A (Arduino pin A3)
b      - Toggle User Output B (Arduino pin A4)

Az     = 20
Out A  = 0
Out B  = 1

Az     = 20
Out A  = 0
Out B  = 1
█
```

Figure – Rotator control over serial console

Serial console is connected by USB to serial converter which is on Arduino Nano board. In terminal program, I use Putty, it is necessary to configure proper serial port COMx with parameters 9600bps, 8N1. Note that after successful connection the Arduino Nano is reset, take this into account and check if initial azimuth is correct!

Bill of Materials

- Dual H Bridge DC Stepper Motor Drive Controller Board Module (L298N, ST Microelectronics)
- XL6009E1 Step-up Adjustable Voltage Regulator 5V / 12V / 24V (XL6009, XLSemi)
- Arduino Nano V3.0 ATmega328P CH340G 5V 16M Micro-controller Board
- TCST1230 Transmissive Optical Sensor with Phototransistor Output (Vishay)
- FOD817 4-Pin DIP Phototransistor Optocoupler (OnSemi)

1602A-1 LCD Module 16x2 (SHENZHEN EONE Electronics Co., Ltd)
EC11E1820402 Alps incremental encoder (18imp / rotation), or similar

There are other standard components used - such as NPN transistors, resistors, diodes, connectors etc.

Actual version of Arduino program/project together with 3D model of housing and control knob can be found on <http://om2ju.com>, tab #hamradio.

