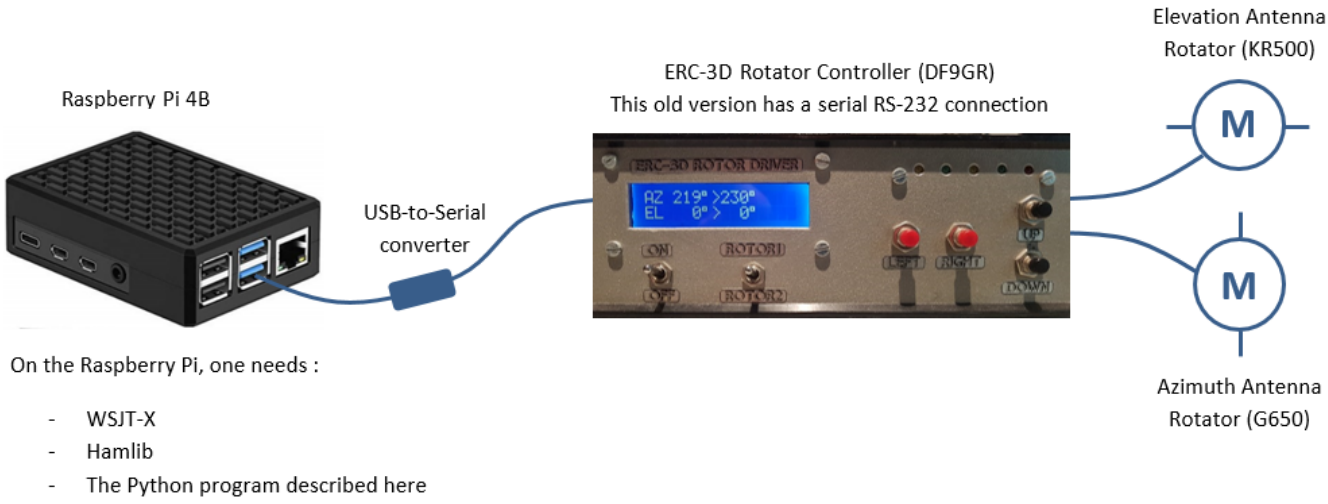
Antenna rotator control program for Raspberry Pi and WSJT-X

For the digital modes traffic, I run WSJT-X on a Raspberry Pi (4B). So, no need to devote a “big” computer for that purpose, the Raspberry does the job perfectly well, and for less power consumption and noise. For my 144 MHz moonbounce operations, I wanted a simple system to drive my existing ERC-3D rotator controller (by DF9GR). That model is now obsolete but works flawlessly and I wanted to keep it as-is, without changing anything. The ERC-3D is linked to azimuth (Yaesu G650) and elevation (Kenpro KR500) antenna rotators.



The search for a ready made program I did on the Internet was not successful, since I wanted to keep as-is my exisiting system. Having in mind that WSJT-X provides an azel.dat file including the coordinates of the moon, I decided to write a program to extract these data and to use them to command my ERC-3D, in first place to track the moon.

To be able to do it, a prerequisite is to have Hamlib installed on the Raspberry. What is hamlib : “the Ham Radio Control Library –**Hamlib**, for short– is a project to provide programs with a consistent Application Programming Interface (API) for controlling the myriad of radios and rotators available to amateur radio and communications users”. To date the current hamlib version is 4.2. and can be downloaded here. How to install it on the Raspberry is out of the present scope,

there a several tutorials on the Internet that clearly explain it (it is very easy).

Once Hamlib was installed, the program had to be written (it is available at the end of this post). It has been written in Python 3 and, since I'm not at all a programmer, it is my friend Didier (ON4KDV), who wrote the backbone of the program (many thanks to him !). I then customized it and added some features, a.o. the possibility to command the rotator controller not only to automatically track the moon but also to rotate/elevate the antennas wherever needed.

The program is named "Rotator Controller" and an icon placed on the desktop of the Raspberry allows to launch it easily. It opens in the command-line terminal.

Here is below how it works.

First you select the USB port to be used and then you have the choice :

- Manual rotation/elevation
- Automatic tracking of the moon, as from the azel.dat file out of WSJT-X
- Exit

We start first with the manual rotation/elevation. The possible options are self-explanatory. See also the code at the end, it provides many explanations (comments) too.

```
USB port to use (0-3) : 1

MANUAL rotation/elevation, type m
AUTOMATIC tracking, type a
EXIT, type e

m

ROTATE antenna (azimuth only) and set elevation to 0°, type r
ROTATE and ELEVATE antenna (azimuth and elevation), type l
Get current POSITION, type p
STOP, type s
EXIT, type e
```

When the option "r" is selected, if it was elevated before, the antenna will come back to the horizon (0° elevation) and towards the wished azimuth. "s" will stop immediately any rotation/elevation.

On the example here, we set the azimuth to 113°. Then we ask for the position ("p")

and we see that the antenna is effectively well on Az. 113°.

```
r

Rotate antenna to azimuth (°) : 113
set_pos: 113 0|RPRT 0

ROTATE antenna (azimuth only) and set elevation to 0°, type r
ROTATE and ELEVATE antenna (azimuth and elevation), type l
Get current POSITION, type p
STOP, type s
EXIT, type e

p
get_pos:|Azimuth: 113.00|Elevation: 0.00|RPRT 0

ROTATE antenna (azimuth only) and set elevation to 0°, type r
ROTATE and ELEVATE antenna (azimuth and elevation), type l
Get current POSITION, type p
STOP, type s
EXIT, type e
```

Then we select “l” and we set Az. to 113° and El. to 10°. Requesting the position by typing “p”, we get well 113° Az. and 10° El.

```
l

Rotate antenna to azimuth (°) : 113
Elevate antenna to elevation (°) : 10
set_pos: 113 10|RPRT 0

ROTATE antenna (azimuth only) and set elevation to 0°, type r
ROTATE and ELEVATE antenna (azimuth and elevation), type l
Get current POSITION, type p
STOP, type s
EXIT, type e

p
get_pos:|Azimuth: 113.00|Elevation: 10.00|RPRT 0

ROTATE antenna (azimuth only) and set elevation to 0°, type r
ROTATE and ELEVATE antenna (azimuth and elevation), type l
Get current POSITION, type p
STOP, type s
EXIT, type e
```

To come back to the main menu, you have to exit (“e”) and restart the program. Before restarting the program, we must start first WSJT-X, so that the azel.dat file

is available. We then restart the program and we select a USB port not already in use by WSJT-X (for the CAT for example).

We select “a” for the automatic moon tracking. We are prompted to set a “refresh duration”, it is the time interval (in minutes, but can be customized in seconds, see comments in the code) between two refreshes of the moon position (access to the azel.dat file in WSJT-X).

```
USB port to use (0-3) : 1

MANUAL rotation/elevation, type m
AUTOMATIC tracking, type a
EXIT, type e

a

Refresh duration (min) : 2
Moon azimuth (°) : 114.4
Moon elevation (°) : 3.1
set_pos: 114.4 3.1|RPRT 0

get_pos:|Azimuth: 113.00|Elevation: 4.00|RPRT 0

Waiting for the next update or CTRL+C to exit
```

Every x (2 in the example) minutes, as per defined by the “refresh duration”, the moon position is derived from the azel.dat file, the antenna is rotated and elevated accoringly, and the actual antenna position (azimuth and elevation) is reported 5 seconds later.

```
Moon azimuth (°) : 115.3
Moon elevation (°) : 3.7
set_pos: 115.3 3.7|RPRT 0

get_pos:|Azimuth: 115.00|Elevation: 4.00|RPRT 0

Waiting for the next update or CTRL+C to exit

Moon azimuth (°) : 115.7
Moon elevation (°) : 4.0
set_pos: 115.7 4.0|RPRT 0

get_pos:|Azimuth: 115.00|Elevation: 4.00|RPRT 0

Waiting for the next update or CTRL+C to exit
```

If the moon is below the horizon, nothing happens and the program indicates “The moon is below horizon, antenna parked !”, and so on until the moon rises above the horizon. To exit, press “CTRL” and “C”.

```
a

Refresh duration (min) : 2
Moon azimuth (°) : 107.0
Moon elevation (°) : -2.3

The moon is below horizon, antenna parked !

get_pos:|Azimuth: 143.00|Elevation: 0.00|RPRT 0

Waiting for the next update or CTRL+C to exit

Moon azimuth (°) : 107.4
Moon elevation (°) : -2.0

The moon is below horizon, antenna parked !

get_pos:|Azimuth: 143.00|Elevation: 0.00|RPRT 0

Waiting for the next update or CTRL+C to exit
```

Here is the heavily commented python 3 program :

```
#!/usr/bin/python3
# coding: utf-8

# This script allows to rotate an antenna system either by manually typing in a heading (azimuth and/or elevation)
# you want the antenna to rotate to or by automatically tracking the moon azimuth and elevation.
# The Moon azimuth and elevation are extracted out of the file azel.dat provided by the WSJT-X suite.
# Beside WSJT-X, this script also makes use of the rotator controller daemon embedded in the Hamlib
# libraries. So, the prerequisites for this script to run are WSJT-X opened and running, and the
# Hamlib librairies installed on your Raspberry Pi.
# This script has been written in Python 3 by Didier, ON4KDV and Gaëtan, ON4KHG.
# My personnal system makes use of a Raspberry Pi 4B and an ERC-3D (DF9GR) antenna
```

```
rotator controller.
# Both are linked with a USB (on the Raspberry Pi side) to RS232 serial (on the controller side) connection.

# First, import the time, system and subprocess functions needed in this script

import time
import os, sys
import subprocess

# Select the USB port of the Raspberry Pi onto which the rotator controller is connected.
# Normally, as WSJT-X is started beforehand, the USB0 port is already assigned for the CAT & PTT of WSJT-X.
# In my case, I usually select USB1, so, I type “1” below.

usb_port=(input(“USB port to use (0-3) : “))
print()

if usb_port == “0”:

# The Rotctld(aemon) is opened with the command “rotctld -m 601 -r /dev/ttyUSB0 &”.
# -m 601 is related to the GS-232A rotator protocol.
# You could have to have to select another number according to the protocol and associated hardware in use on your side.
# The list of supported protocols can be obtained by typing “rotctld -l” in the terminal.
# ttyUSB0 is the serial port (USB0) of the Raspberry onto which the rotator controller is connected.

        subprocess.call(‘rotctld -m 601 -r /dev/ttyUSB0 &’, shell=True)

# And so on for the other USB ports.

elif usb_port == “1”:

        subprocess.call(‘rotctld -m 601 -r /dev/ttyUSB1 &’, shell=True)

elif usb_port == “2”:
```

```
        subprocess.call('rotctld -m 601 -r /dev/ttyUSB2 &', shell=True)

elif usb_port == "3":

        subprocess.call('rotctld -m 601 -r /dev/ttyUSB3 &', shell=True)

# Select the working mode : manual, automatic (moon tracking) or exit.

Mode=(input("MANUAL rotation/elevation, type m\nAUTOMATIC tracking, type a\nEXIT, type e\n\n"))

if Mode == "m":

# If "m" is chosen, the manual rotation mode is selected.

    while (True):

# Then, select what you want to do : rotate/elevate the antenna, get its current position, stop the rotation in case
# of emergency or exit.

      Submode=(input("\nROTATE antenna (azimuth only) and set elevation to 0°, type r\nROTATE and ELEVATE antenna (azimuth and elevation), type l\nGet current POSITION, type p\nSTOP, type s\nEXIT, type e\n\n"))

      if Submode == "r":

# If "r" is selected, instruct towards which azimuth the antenna has to rotate. With this selection, if the antenna was previously elevated,
# it will be set back to 0° (no elevation) too.

         az=(input("\nRotate antenna to azimuth (°) : "))
         az=az.strip()

# Build and execute the command "set_pos". The default port used by the rotctld(aemon) is 4533.
```

```
         command_az='echo "|\set_pos ' + az + ' 0" | nc -w 1 localhost 4533'
         os.system(command_az)

      elif Submode == "l":

# If "l" is selected, you can set whatever azimuth and elevation angle.

         az=(input("\nRotate antenna to azimuth (°) : "))
         az=az.strip()
         el=(input("Elevate antenna to elevation (°) : "))
         el=el.strip()

          command_azel='echo "|\set_pos ' + az + ' ' + el + '" | nc -w 1 localhost
4533'
         os.system(command_azel)

# If "p" to get the position of the antenne is selected, build and execute the
command "get_pos".

      elif Submode == "p":

         command_azpos='echo "|\get_pos" | nc -w 1 localhost 4533′
         os.system(command_azpos)

# If "s" to (emergency) stop is selected, build and execute the command "S" (Stop).

      elif Submode == "s":

         command_stop='echo S | nc -w 1 localhost 4533′
         os.system(command_stop)

      else:

# If none of the above is selected, then exit the script.

         exit()

elif Mode == "a":

# If "a" is chosen, the automatic (moon tracking) mode is selected.
```

```
# Set the refresh rate in minutes of the moon azimuth and elevation.
# With narrow beamwidth antennas (microwaves), the refresh rate has to be fast.
# With wide beamwidth antennas, the refresh rate can be slower. On 2m, I chose 4 minutes.
# If you want a refresh time in seconds, remove the line “ref=ref*60” and rename “min” into “sec”
# in the line ref=int(input(“\nRefresh duration (min) : “)).

    ref=int(input(“\nRefresh duration (min) : “))
    ref=ref*60

    while (True):

# Open the file azel.dat of WSJT-X in read mode. Make sure the path “/home/pi/.local/share/WSJT-X/azel.dat” is the same
# than the path defined in the settings of WSJT-X (see the WSJT-X user guide).
# The first line of the file is read and placed in the variable “txt” and then the file is closed.

      f=open(“/home/pi/.local/share/WSJT-X/azel.dat”,”r”)
      txt=f.readline()
      f.close()

      if (“Moon” in txt):                 # Check that the first line contains the word “Moon”.
        p=txt.find(“,”)                   # Search for the 1st comma (,).
        if (p > 1):                       # If the 1st comma has been found
            txt=txt[p+1:]                 # what is in front of the 1st comma is removed, including the comma.
           p=txt.find(“,”)                # Search for the 2nd comma.
           if (p > 1):                    # If the 2nd comma has been found

                az=txt[0:p]               # the text in between the 1st and 2nd commas is saved in “az” (azimuth).
               az=az.strip()            # Spaces before and after are removed, only the figures/sign are kept.
              print(“Moon azimuth (°) :”,az)   # The moon azimuth is displayed.
               txt=txt[p+1:]              # What is in front of the 2nd comma is
```

removed, including the comma.

```
                p=txt.find(“,”)          # Search for the 3rd comma.
                if (p > 1):              # If the 3rd comma has been found

                     el=txt[0:p]           # the text in between the 2nd and 3rd
commas is saved in “el” (elevation).
                    el=el.strip()        # Spaces before and after are removed, only
the figures/sign are kept.
                     print(“Moon elevation (°) :”,el)   # The moon elevation is
displayed.

# Build the command “set_pos” as from the az and el variables extracted above.

                    command_azel=’echo “|\set_pos ‘ + az + ‘ ‘ + el + ‘” | nc -w 1
localhost 4533’

# Convert az and el from text to float (numerical) format.

                    f_az=float(az)
                    f_el=float(el)

# Execute the command only if the azimuth is between 0 and 360° and if the elevation
is between 0 and 90°.

                    if(f_az >0 and f_az <361 and f_el >0 and f_el <91):
                         os.system(command_azel)

# Otherwise the moon is below the horizon and nothing is executed but displaying
that the moon is below horizon.

                    else:
                         print(‘\nThe moon is below horizon, antenna parked !’)

# We wait 5 seconds, the time for the antenna to rotate (it may actually take
longer, depending on the position of
# the antenna at startup of the tracking) and then we display the position of the
antenna.
# Finally, we indicate that we wait until the next update, according to the refresh
```

```
duration (ref) defined above.

                    time.sleep(5)
                    print()
                    command_azelpos='echo "|\get_pos" | nc -w 1 localhost 4533´
                    os.system(command_azelpos)
                    print('\nWaiting for the next update or CTRL+C to exit\n')

      time.sleep(ref)

# If none of the working mode (manual or automatic moon tracking) was selected, exit and close the script.

else:
    exit()
```

Antenne 144 MHz 2×9 éléments DK7ZB (2012)

Ce système d'antenne est pourvu d'un mécanisme d'élévation et est destiné à un usage EME. Il est constitué de 2 fois 9 éléments DK7ZB mises côte-à-côte ("bayed" en anglais). La distance entre les antennes est de 3,5 m.

Les constituants de l'antenne ont été achetés en kit chez Nuxcom ; très pratique pour disposer des pièces de l'antenne "toutes en un".

Tous les éléments (y compris le dipôle) sont réalisés en tube d'aluminium de 8 mm de diamètre. Le système d'adaptation d'impédance (50 <> 28 ohm) comprend 2 sections d'un quart d'onde de câble coaxial RG59 (75 ohm) mises en parallèle. C'est loin d'être le meilleur câble qui soit mais étant donné la longueur de 34,5 cm mise en

jeu ici (quart d'onde*facteur de vélocité du câble), elle n'affectera pas de manière significative le gain de l'antenne. Telle quelle, une antenne pourra supporter une puissance maximale de 350W et, de fait, 700W pour le système complet.

Antenne 70 MHz 5 éléments YU7EF (2010)

Les exigences pour cette réalisation étaient légèreté (max 3 m de longueur de boom et système d'adaptation simple) et minimum 8 dBd de gain. J'ai opté pour un design 50 ohm de YU7EF, la EF0405C.

Tous les éléments sont constitués de tubes d'alu. de 10 mm de diamètre. Aucune retouche n'a été nécessaire par rapport aux dimensions d'origine (large bande-passante). Les supports d'éléments ont été achetés chez Nuxcom.

Antenne très efficace qui m'a permis de contacter OY9JD en Tropo sur plus de 1400 km.

Antenne 144 MHz 12 éléments DK7ZB (2005)

C’est un design 28 ohms de Martin, DK7ZB. Les supports des éléments ont été achetés chez Wimo ; tous les autres éléments ont été trouvés chez le détaillant du coin. Le bras de support horizontal est simplement constitué d’un profilé en bois traité et peint. Voici les dimensions finales après réalisation de l’antenne. Elles sont identiques au design original de DK7ZB (radiateur 12 mm de diamètre & éléments de 8 mm de diamètre), excepté la taille du dipôle que j’ai dû raccourcir de 972 à 965 mm, sans quoi l’antenne résonnait dans les 143 MHz.

En service fiable depuis août 2005, c’est la meilleure antenne utilisée pour le DX Tropo jusqu’à présent. Diagramme de rayonnement un peu trop étroit pour le MS à courte et moyenne distance (comme toutes les longues yagis). Une “tueuse” pour la Tropo, l’amélioration (2 dB de plus) comparée à la 9 él. Wimo (basée sur un design DK7ZB) est nettement perceptible. Le diagramme étroit est également perceptible en MS et en contest (taux de réponse aux CQ’s plus faible mais meilleur pour la chasse aux DX’s).

Longueur des éléments (mm)	Position des éléments (mm)
1013	0
965	405
948	680
922	1275
904	1970
890	2800
880	3685

874	4570
868	5485
868	6385
879	7275
873	7980

Le gain se monte à 14,2 dBd, ce qui est très optimal pour une yagi de 3,83wl. Néanmoins, la bande passante est très étroite (voir la mesure du VSWR dans galerie d’images ci-dessus), ce qui en fait une antenne très sensible, principalement à la neige et la glace. En juillet 2010, Martin a publié une version encore davantage optimisée, ayant apparemment une bande passante plus large.