November 11, 2025

Switches and VNA Software for Automation

Charlie G Tolley[1,2]

[1] *Department of Astronomy, University of California Berkeley, Berkeley, CA 94720, USA*
[2] *Radio Astronomy Laboratory, University of California Berkeley, Berkeley, CA 94720, USA*

1. INTRODUCTION

While the EIGSEP antenna is suspended in the canyon, it is necessary to regularly measure the S11 reflection coefficient of the antenna, the receiver, and the noise source pathways. To this end, the software for the Vector Network Analyzer must be automated on a cycle of measuring these pathways and taking sky measurements. I present the software overview and some common commands for the Vector Network Analyzer (VNA) (GitHub: https://github.com/EIGSEP/CMT-VNA.git) and switch network (GitHub: https://github.com/EIGSEP/switchNW.git) code.

2. SWITCHES

We are using Teledyne CCR-33S30-T Single-Pull, Double-Throw (SPDT) failsafe switches - SPDT refers to switches with one input and two outputs, and failsafe means that to remain in the high power state, a constant actuation current of 200mA at 12V must be applied. To interact with these switches, I've connected their logic pin with a Raspberry Pi Pico flashed with MicroPython. Using command line command `mpremote`, I copy and run a script that listens for commands from the LattePanda Mu Single-Board Computer (SBC).

Figure 1. Teledyne CCR-33S30-T SPDT Failsafe Switch

This script, `ctrl_gpio.py` must be hard-coded with the GPIO pin numbers that are connected to each switch (i.e. `PINS = [0,1,2]`) where 0, 1, and 2 are the GPIO pin numbers the switches are attached to. Once you have changed the `PINS` list, you can copy over the new file with `mpremote cp ctrl_pico.py :ctrl_pico.py`, and run it with `mpremote run ctrl_pico.py`. Once your pico is listening, you should create a dictionary of "paths" that define each path in terms of the switch state (0 is low power state and 1 is high power state). The index of the state should correspond to the index of the GPIO pin number in the PINS list you hard-coded to the Pico, and it should be a string of 0s and 1s. For example, a 3-switch network might be defined like so:

```
paths = {
        'antenna':'000',
        'load': '010',
        'open': '100',
        'short':'101'
        }
```

Where the first index corresponds to the switch connected to the GPIO pin at the first index of your `PINS` list.

Corresponding author: Charlie G Tolley
tolley412@berkeley.edu

When initializing your SwitchNetwork, be sure to know what `/dev/ttyACM*` number your pico is connected to. You can check this by running `ls /dev/ttyACM*`, which should only turn up one item (unless you're connected to multiple serial ports). The default is `/dev/ttyACM0`, but occasionally it will switch itself. For an example of initializing a SwitchNetwork:

```
snw = SwitchNetwork(paths=paths, serport='/dev/ttyACM0')
```

Once this is initialized, `snw` has attributes `state`, which holds a tuple with the path's key (i.e. 'antenna') and the power sum, which sums the number of switches in the high-power, or ON state. To switch to a new path, one should run:

```
snw.switch('load') #replace with chosen path
```

Make sure that all switches are switched off after finishing with your application. The function `snw.powerdown()` is a handy way of going to the lowest-power state on the switch network without you having to remember what the path is that is in the lowest power state.

3. CMT-VNA

Within the `cmt_vna` package, there are two modules: a) `vna`, with a class `VNA` that controls the VNA, and b) `calkit`, which contains calibration code and a class `CalStandard` that holds the open, short and 50Ω load models. I will talk about the `calkit` module in a future memo.

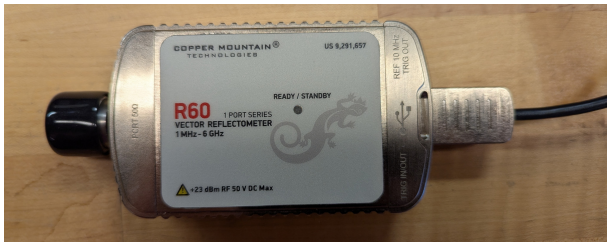

Figure 2. Copper Mountain Technology R60 1-port VNA

3.1. *Instrument Code*

We use the Copper Mountain Technology R60 1-port VNA, which is a compact VNA capable of reading the reflection coefficient of the port it is connected to - this is often called the S11 parameter (for a 1-port system), which it gets by sending a broadband signal and measuring what is reflected back at it as a fraction of voltage in. We talk to the VNA with Standard Commands for Programmable Instruments, referred to here as SCPI. SCPI is organized in a command tree, so the commands each follow branches. For example: `FORM:DATA REAL` is composed of `FORM` for the FORMat branch, then `DATA` specifies what the format is referring to, and `REAL` is the argument that tells the data what format to be in. If you're curious, feel free to reference the manual scp (2021), but you should not need to write any SCPI code to work with the VNA.

Requirements: All requirements are up to date in the `setup.cfg` file. In brief, you need `numpy`, `matplotlib`, `pyserial`, `PyVISA`, `pyvisa-py`, and `switch_network` (which I talk about in section 2).

The `VNA` class has two important arguments: `ip` and `port`. Both of these have defaults hard-coded into the module, and so should not need to be specified. The `setup` function must be run to ensure you are working within your specifications - you can specify the frequency range in Hz (`fstart`, `fstop`), the number of points to sweep through (`npoints`), the intermediate frequency bandwidth (`ifbw`) and the power in dBm (`power_dBm`), all of which have appropriate defaults. It returns a frequency array and adds that array to the `VNA` object you instantiated. To set up your measurements, you might run:

```
from cmt_vna import VNA
vna = VNA(ip='127.0.0.1', port=5025)
freqs = vna.setup(fstart=1e6, fstop=250e6, npoints=100, ifbw=100, power_dBm=0)
```

Note that all values given in the last line above are the defaults and do not need to be specified. Running `vna.freqs` should return the same array as is assigned to the `freqs`.

Once the instrument is set up, then you can sweep through the reflection coefficient measurements. `VNA` has a built in function called `measure_S11` which calls the SCPI command to sweep:

```
s11 = vna.measure_S11()
```

These functions are the most important ones to understand in order to operate the VNA, with a few others for ease of use. For example:

- `measure_OSL` switches through the open, short and load standards which
- `add_OSL` calls and stores to save once done with the measurement in `vna.data`. Calling `auto=True` will implement the automatic switching protocol.
- `read_data` calls `measure_S11` and adds the gammas to `vna.data`.
- `write_data` writes all the data to a file and clears the data out of the VNA object.

3.2. *Scripts*

The two most useful scripts for operating the VNA are `measure_s11.py` and `get_cable_network_sparameters.py`. The former is an example of how the VNA code will be implemented in the field - for a given number of files, it measures the OSL standards, then takes a set number of S11 measurements of the antenna itself, then sleeps for a given amount of time. Whether or not to use the automatic switch network needs to be specified, as should the cadence, the number of S11 files to take, the number of calibration cycles to go through, and whether or not to measure the OSL standards. A generally useful script call for running one set of calibrated test measurements is:

```
python3 measure_s11.py --osl --npoints 1000 --fstart 50e6 --fstop 250e6 -c 0 -m 1
--outdir /home/charlie/eigsep/CMT-VNA/data/unit_tests/
```

Note that many of these values are defaults and do not need to be specified.

REFERENCES

2021, RVNA and RNVNA SCPI Programming Manual, 21st edn., Copper Mountain Technologies, Indianapolis, IN