Brady Volkmann

8/1/2019

University of Missouri Kansas City

Load Frame, VNA, Power Supply Usage Guide

Intro:

The load frame of this usage guide was designed at UMKC in 2019 by Steve Siegal (Hardware), Steve Young (Conceptual), Brady Volkmann (Software and Electrical). Although there is a few README.md’s in the directories containing the code, I intend for this usage guide to be more general and not just the software.

Motivation:

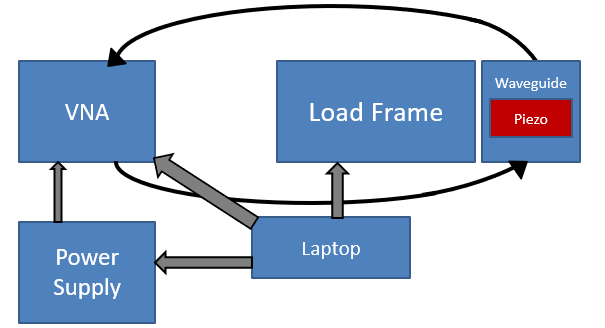
The load frame was designed so that piezoelectric materials may be methodically strained. To measure strain/polarization properties of the piezoelectric, a TTR506A VNA takes a transmission measurement across the piezoelectric material in a waveguide. The VNA also may receive a bias voltage from a TP3005P Power Supply. The goal is to automate all these instruments and to record data by stepping through forces applied by the load frame and by bias voltages courtesy of the power supply.

Figure Overview of Setup

Where:

The load frame now resides in Flarsheim Hall, Lab 110. It will be mounted on the optics table to the right when you enter the room.

Hardware Connections:

The laptop that controls all the instruments is currently the Lenovo in 110. You must make three USB connections with the laptop to ensure the load frame is working. The laptop must plug into the VNA, the power supply, and the Arduino within the junction box. If you are not using the VNA and just debugging the code, using the VectorVu-PC simulator is often easier than using the TTR506A.

The load frame is controlled by the plastic junction box. There are five connections that you must establish for the junction box to function fully. The box must receive 24V (minimum 300mA) from the DC input on the side of the box. Then there are three holes: the first connects the Arduino to laptop; the second connects the load cell to the load cell amplifier; the third connects the stepper motor driver to the stepper motor on the load frame. Lastly, there is the relay which needs to be attached to the Power Supply with the gray cables as the input and the white and orange cables as the output; this is to switch the polarization of the power supply if needed. Presuming all the connections within the box are intact, the junction box is ready to go.

See the HardwareDescription.md in VolkmannSummer2019/Development/ if more detail is needed.

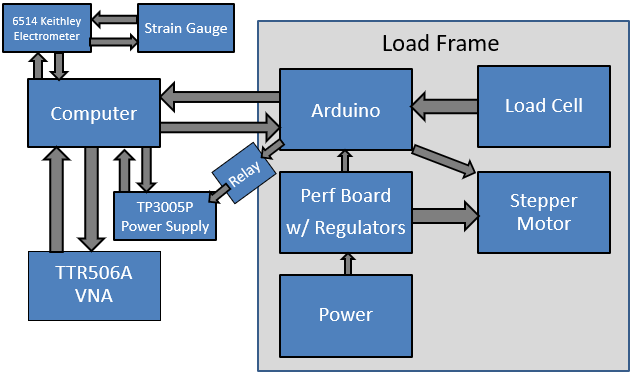


Figure Hardware Detailed - Note: The electrometer and strain gauge have not been implemented.

Software Setup:

Note: This software expects some familiarity with Python.

There are many programs you will need to download if you are starting this on a laptop other than the Lenovo. These are detailed in the README.md files contained in /VolkmannSummer2019/Development/LoadframeVNAControls and /…/PowSupVNAControls. Such software includes:

* Python 3.6.5
* Arduino IDE
* PyVISA 1.2.4
* VectorVu-PC 1.2.4 (other versions don’t seem to work with PyVISA)
* NI-VISA 19.0
* TekVISA.

The primary trouble-maker in this software Py-VISA. Py-VISA seems to have difficulty communicating with any version of VecorVu-PC other than 1.2.4. To check if PyVISA is working. Go to PowerShell and type Python. Then, once Python has launched typed the following inputs:

>> import visa

>> rm = visa.ResourceManager()

>>print(rm.list\_resources())

(‘GPIB8::1::INSTR’)

If you do not get a GPIB option, then PyVISA is failing to talk to your VNA. This can take a while to trouble shoot, and I’ve had difficulties with this myself a few times. I don’t know any quick fixes, but I’ve found Tektronix’s support team and live chats to be helpful.

Using the Software:

There are currently two ways to use the software. You can control the Load frame and VNA at the same time or the VNA and the Power Supply. These two modes have not yet been unified.

The load frame and VNA setup can be found in /VolkmannSummer2019/Development/LoadframeVNAControls. LoadframeController.py is the master file that you would execute to begin an experiment. To configure the experiment, at the bottom of LoadframeController.py where the constructor Controller() is called, change the arguments of the constructor according to your needs. Read the docstring of the class and the constructor \_\_init\_\_() for information on the parameters. The LoadframeController.py has two ways to execute: .runByForce() and .runByStep(). .runByForce() is a method you would call on an instance of the controller class to step through different forces applied by the load frame and take a VNA measurement at each step. The .runByStep() works similarly except it steps by motor steps instead of force. The data are recorded in three ways: plots, csvs, and logs. All these are located in intuitively named folders with timestamped files in the same directory as the executed script. The load frame and VNA setup has files that it depends on: ttrvna.py, pyLoadControl, and pyMotorControl. Each of these is a child class of the parent class Instrument, contained in instrument.py. ttrvna.py controls the VNA and pyLoadControl and pyMotorControl control the load cell and stepper motor respectively, both through the same serial communication. Any advanced modifications to the VNA should occur in ttrvna.py.

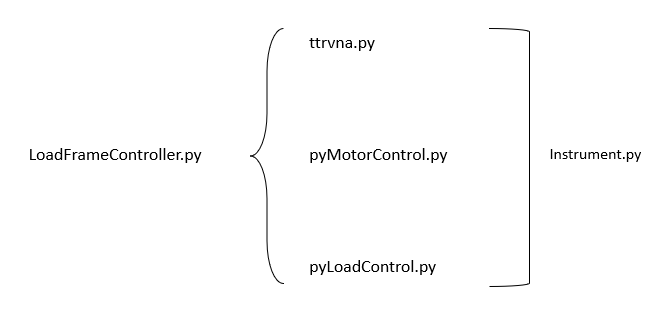


Figure 3 Load frame and VNA software architecture

Now the alternative mode is the VNA and Power Supply mode. /VolkmannSummer2019/Development/PowSupVNAControls contains the code for this mode. VNAandPowSup.py is the file you would execute to run an experiment. Again, to configure the experiment, go to the bottom of this file and change the parameters of the constructor PowSupVNA().There are three ways to run this. There is the method .sweepUpDown() to go up to 15V, back to 0V, down to -15V, back to 0V at interval specified by self.\_voltageStep. A VNA measurement is made between each voltage step. This method records data as intensity v. frequency plots. There is .sweepUp() to simply go up to 0V to 15V at an interval specified by self.\_voltageStep. Then there is .sweepDownUp which goes from -15V to 15V at an interval specified by self.\_voltageStep. This last method encourages plotting voltage v. intensity graphs at a specified frequency through .plotFreqSpecific(<freq>) since the data is recorded easily for this kind of plotting in this mode. The data is again recorded in plots, CSVs, and logs in subdirectories of the same directory as the script you are executing. Here VNAandPowSup.py relies on two other files in the directory: ttrvna.py, which controls the VNA as before, and TP3005P which controls the power supply and the relay to change polarizations. The class in ttrvna.py is an child of the instrument class again.

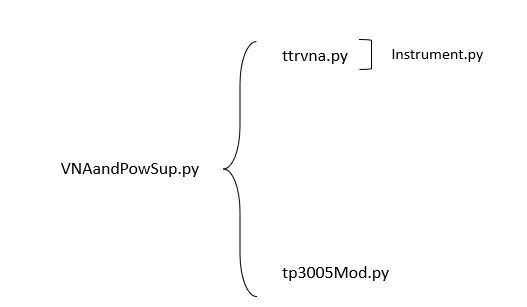


Figure VNA and power supply software architecture

Now to execute, launch VectorVu-PC and ensure the COMs are set correctly in the Constructor call of whichever file you are executing. Then, in PowerShell navigate to the file location of the file you wish to execute and type `python <filename>.py` to execute the file.

Notes on Arduino ⬄ Python communication:

The Arduino script is located in /VolkmannSummer2019/LoadFrameArd/.

When it is uploaded to the Arduino, it only acts when it receives serial communication either from the serial monitor in the Arduino IDE or from Python via PySerial. PySerial talks to the Arduino and gives it commands by sending it singular, predetermined command characters using <serialObject>.write(b’<insertCommandCharacter’>). Python can readout anything the Arduino prints to its serial monitor via <serialObject>.readline().

Timing:

The timing is primarily limited by the speed at which the VNA can sweep, which appears to be around 8 seconds if you sweep 200 points on normal mode.

Deficiencies:

There are many deficiencies in this software that I would love to fix, but my time at UMKC is coming to an end. I will put a non-exhaustive list here in case any user wants to fix them or add features to the software.

* Add .findContact() method to LoadframeController.py to more quickly find where the load frame contacts the device it is tensioning or compressing
* Add a method to guess how many motor steps are needed to achieve the desired force. Since it does it linearly now, it can be very slow.
* Get the VNA to plot more or less than 200 points. .retrieveData(<points>) should do this in ttrvna.py, but there appears to be some issues with it.
* Create force v. intensity plots in LoadframeController.py in a similar way to how VNAandPowSup.py creates voltage v. intensity plots. This ends up being a fairly large change, because all the data is recorded in the module files currently, not the LoadframeController.py file due to my strange architecture when I set out on this project.
* Change ttrvna.py such that the imaginary component of the measurement is recorded and can be examined independent of the magnitudes.
* Make this controllable with a GUI. This would be a large change but would make the program much more user-friendly to those not experienced with Python.

I am sure there exist many other issues with the code. If you need my help in making these changes, feel free to contact me at bvv4@cornell.edu.