Antenna Analyzer

Silje Fuglerud
(siljesf@berkeley.edu/siljesf@stud.ntnu.no)

May 2015

1 Introduction

The Antenna Analyzer was based of the K6BEZ project (https://sites.google.com/site/k6bezprojects/antenna-analyser) and made for the Ion Group at Berkeley in the Spring of 2015 as a student project. This document contains instructions on how to use it, as well as some information about the code and bugs. A python program made by Weerapat Pittayakanchit at UCLA was used as a basis and a user interface was added with PyQt4. The arduino code was also made by Mr. Pittayakanchit and was only subject to a few minor changes.

The Antenna Analyzer uses Arduino micro to talk with a DDS. The DDS used was AD9851, which can generate signals at frequencies up to 70MHz.

See Section 2 for information on how to run the program, Section 3 for information on building the analyzer, and Section 4 for information on the code.

2 Running the program

All that is needed is a computer with Python (and some specific packages, see Section 2.1) installed and the Antenna Analyzer box - make sure the usb cable is connected to the arduino and to the computer in use.

In order to choose the right port, see line 44 in the python code `plotSNA_ui.py`. Because the program opens the serial port when initalizing the program, serial will throw an exception if the port is not right and the program will not be executed. As in Figure 1, change the first argument in serial.Serial() to call the right port.

Updated python code can be found at https://github.com/HaeffnerLab/Antenna-Analyzer. Run the program in the command line, and a GUI should pop up as in Figure 2. Select parameters and click the *Set Arguments* button. Frequencies can be chosen from 1 MHz to 60 MHz. *Steps* is in the range 50-500, depending on the computer in use a big value of *steps* might cause the program to run very slowly.

Figure 1: To choose the right port, find the port that the arduino is connected to at your computer and change code directly.

```
32  class Ui_Form(QtGui.QWidget):
33
34      def __init__(self):
35          QtGui.QWidget.__init__(self)
36          self.filename=""
37          self.txtfilename=""
38          self.txt=False
39          self.setupUi(self)
40          self.start=1
41          self.stop=2
42          self.steps=50
43          # Connect with arduino
44          self.ser = serial.Serial("COM3", 9600, timeout = 1)
45
46      def setupUi(self, Form):
```

When the figure has been updated, you can click *Save figure* or *Save data* to save as shown in Figure 3. The picture is saved as .png and the data as .txt. The data is saved in the format:
Frequency Reflection FWD(A1 port) REV(A0 port).

Reflection is calculated by the arduino by using

$$\gamma = \frac{100 \times \mathrm{REV}^2}{\mathrm{FWD}^2}. \tag{1}$$

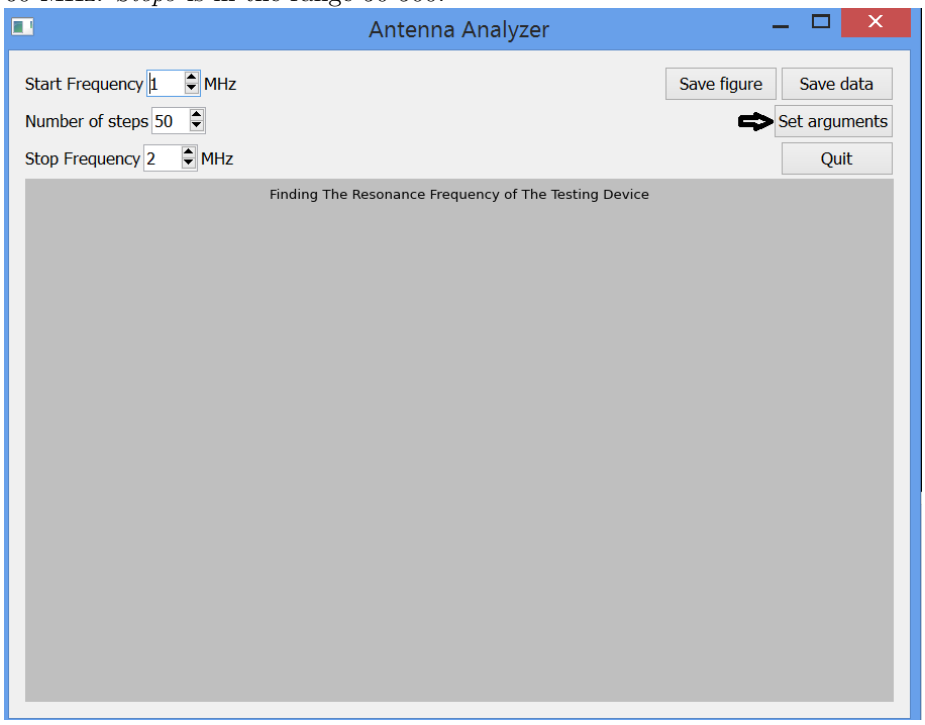
To change parameters, enter the new parameters in the spin boxes and click *Set Arguments*. The program will send new arguments to the arduino, and the window will rescale.

2.1 Required packages

To run the program, the computer must have python installed, with some packages. The code was made with python 2.7 and has not been tested for python 3.3 (as of yet, it should be). The packages serial, matplotlib, PyQt4 and numpy[1] is required to run the python code. One also needs the arduino IDE if one wants to view the arduino code.

[1] Links to downloads:
serial: http://pypi.python.org/pypi/pyserial/
matplotlib: http://matplotlib.org/downloads.html
PyQt4: http://www.riverbankcomputing.com/software/pyqt/download
numpy: http://sourceforge.net/projects/numpy/
arduino: http://www.arduino.cc/en/Main/Software

Figure 2: First frame of the Antenna Analyzer user interface. Select parameters and click the *Set Arguments* button. Frequencies can be chosen from 1 MHz to 60 MHz. *Steps* is in the range 50-500.



2.2 Lab computer

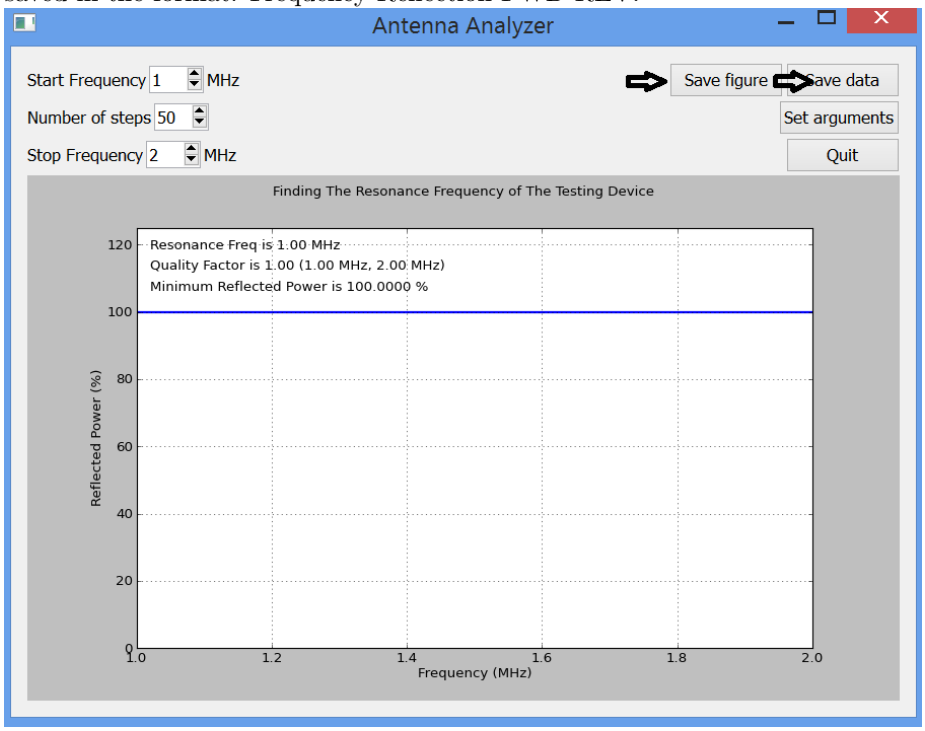
The necessary files has been installed on the TOSHIBA lab computer. Open the command window and type
`C:\Users\Lab-user\Desktop\Antenna Analyzer\plotSNA_ui.py`. The pyqt program does not run properly when used in Spyder, so just use the command line.

The COM port in use is COM15, as seen in Figure 5.

3 Hardware

Explaination by Mr. Pittayakanchit: *The process inside this analyzer is as followed. First, the DDS generates a signal at the specified frequency and sends it to the device at the BNC port. Then, the circuit splits the forward from the backward signal. The amplitudes of those signals are converted to DC and measured by Arduino. Finally, the network analyzer uses python to animate the data from Arduino and plot the graph between the frequency and the reflected*

Figure 3: When the figure has been updated, you can click *Save figure* or *Save data* to save. The picture is saved as .png and the data as .txt. The data is saved in the format: Frequency Reflection FWD REV.



power in real time. Note that the reflected power is the square of the reflected signal's amplitude divided by the square of the forward signal's. See Figure 6 for circuit drawing from the K6BEZ project. Note that a few components differ from the original K6BEZ project and this one. The DDS module is AD9851 instead of AD9850, which increases the maximum signal's frequency from 40MHz to 70MHz. The DDS was bought on ebay, see Figure 7 and had circuit drawing 8. The signal to the circuit is acquired from wave 5, labeled OUT2 on the board. The diodes are NTE110A instead of AA143 (it was easier to acquire).

The arduino and the DDS gets a 5V input through the connection with the computer.

Some notes on the op-amp by Mr. Pittayakanchit: *The problem regarding the op-amp is the supply voltage 5V from Arduino. I could provide a higher voltage to the op-amp, using separated voltage power supply, but for simplicity I choose to rely on the Arduino power supply which work well if the resistance of the device under test is not too low. (If the resistance is too low, the voltage at the positive leg of the op-amp would be too high to amplify at the correct ratio given that the power supply is only 5V). We can notice that the Op-Amp does*

Figure 4: When changing *start* and/or *stop* frequencies, the window rescales to fit the new parameters.

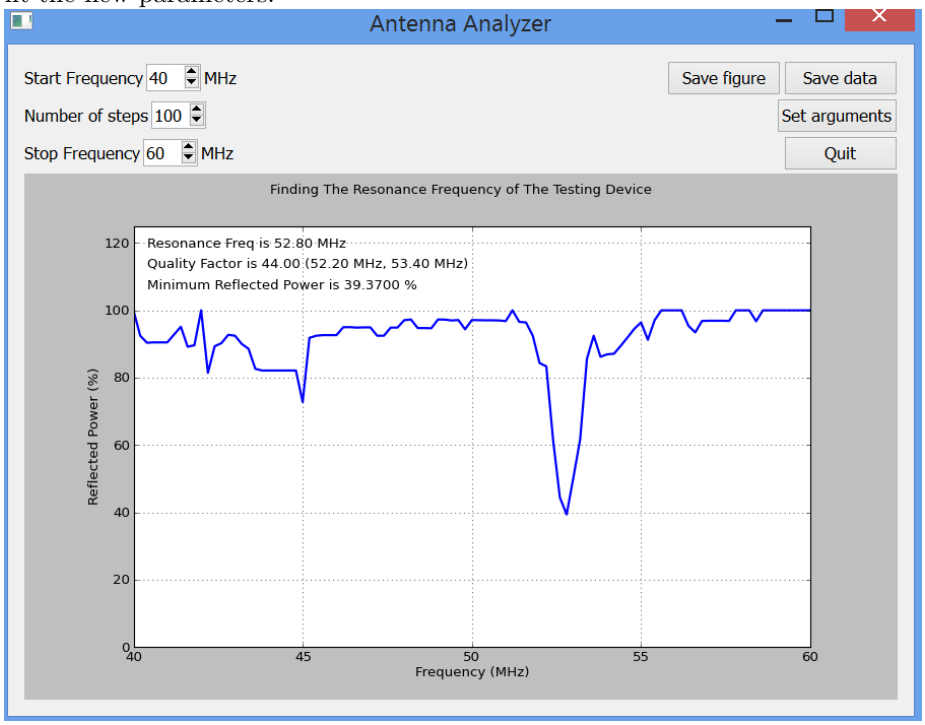


Figure 5: COM15 on lab computer.

not have high enough voltage supply when the forward and the backward signal read by arduino stay saturated at some particular number.

Figure 6: Circuit drawing from K6BEZ project.

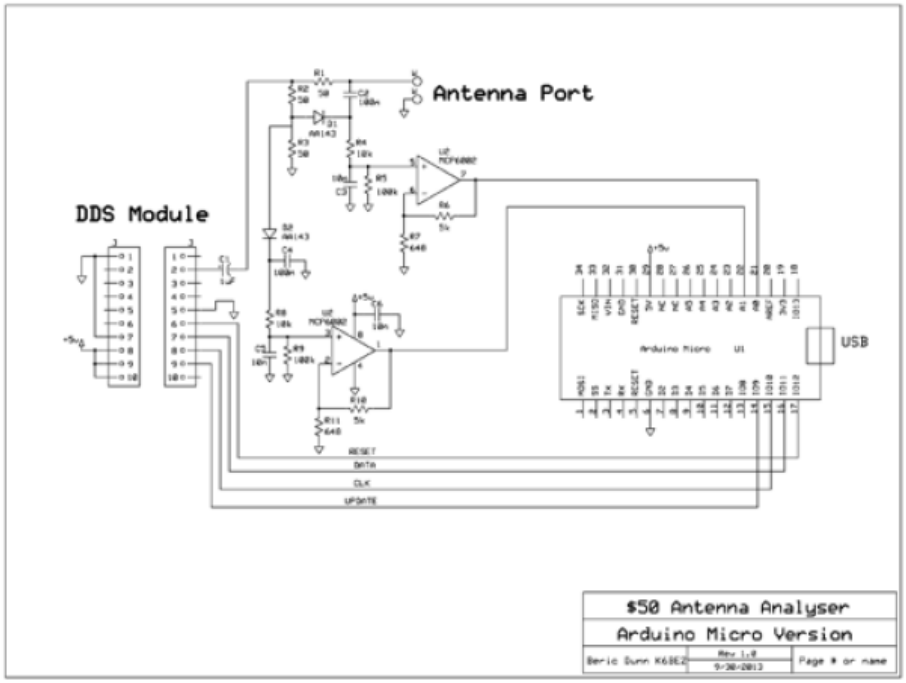
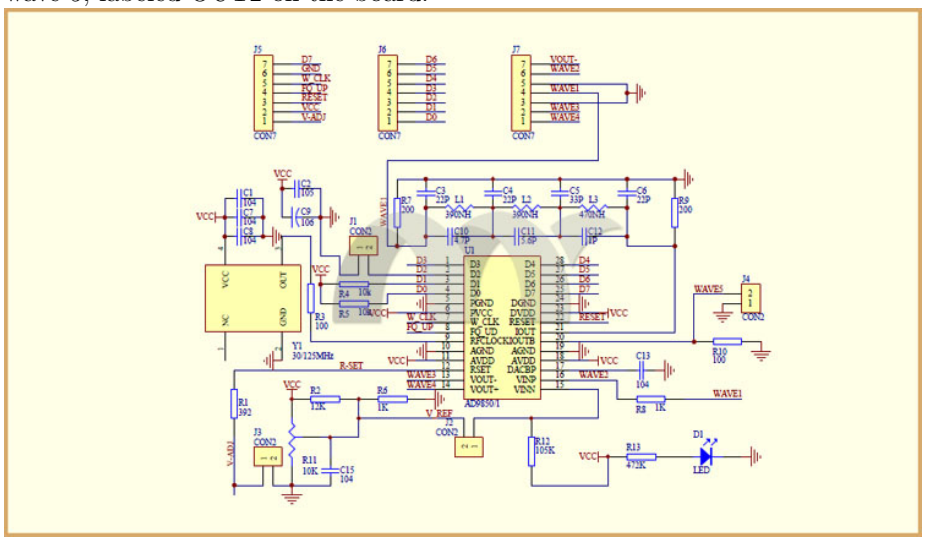


Figure 7: DDS AD9851 bought from ebay.

Figure 8: Circuit drawing for DDS AD9851 module. Note that G7 and GND is mislabeled, they should be switched. The signal to the circuit is acquired from wave 5, labeled OUT2 on the board.



4 Code

Both python code and arduino code can be found in the github repository of https://github.com/HaeffnerLab/Antenna-Analyzer.

4.1 Arduino code

The arduino talks to the computer by using serial. It doesn't do anything until it gets arguments from the computer. Arguments *start frequency*, *stop frequency* and *steps* are set, but it does not make a sweep until recieving the signal 's' from the computer. A signal to the DDS is sent and the DDS makes a contionous sweep within the desired frequencies until the arduino is disconnected.

Notice that the delay in the original code was changed from 0.5 ms to 0.2 ms as shown in Figure 9. If experiencing any connection problems, this could be changed back.

To see directly what the Arduino is sending to the computer, run Arduino IDE and open the serial monitor. Type *1A 60B 100N S* to sweep from 1 MHz to 60 MHz. This is the same that is written to a .txt-file if *Save data* is clicked and has the format

Frequency Reflection FWD(A1 port) REV(A0 port).

If the last two numbers get too small (around 100 or lower), the ratio of those two numbers (reflected power) will not be as accurate.

Figure 9: The delay has been changed from 0.5 ms in the original code, to 0.2 ms, to enhance speed.

```
current_freq = Fstart + i*Fstep;
// Set DDS to current frequency
sendFrequency(current_freq);
// Wait a little for settling (used to be 0.5)
delay(0.2);
// Read the forawrd and reverse voltages
REV = analogRead(A0);
FWD = analogRead(A1);
```

4.2 Python UI code

A great part of the python code is just the GUI setup, which does not have that much to do with performance. The serial connection is opened when starting the program, and the program must be closed in order to close the serial connection. There can be only one connection to the arduino, so do not attempt to open the arduino serial monitor while the python program is running.

The reest of the communication with the arduino happens in setArguments() and run(). SetArguments() sends the start and stop frequencies and the number of steps to the arduino and initalizes a figure with the right axes. run() contains a function update() that is called sequentially, makes arrays with the values and plots them using FuncAnimation(). If *Save data* has been clicked, it also writes the next array to a .txt-file.

saveffunc() opens a widget that gives the user the opportunity to choose a file directory and file name and saves the current figure as a .png file.

savedfunc() opens a widget that gives the user the opportunity to choose a file director and file name and changes a boolean variable so that update() writes to the file.

The q-factor is calculated by

$$Q = \frac{f_{res}}{\Delta f}, \tag{2}$$

where $\Delta f = f_{high} - f_{low}$, which are decided by $\sqrt{2}$ increase in reflected value from the minium reflected value.