Portland State University

Capstone

Instruction Manual

EM Characterization of Radar Absorbing Materials

Kirk Jungles
kjungles@pdx.edu

Chris Toner
ctoner@pdx.edu

David Eding
deding@pdx.edu

Jose Alvarez
jaa@pdx.edu

Sponsor: Tangitek
Faculty Advisor: Dr. Branimir Pejcinovic

June 9, 2020

"Knowledge is Good"

Contents

1 Introduction

This instruction manual serves to describe in detail the progress made by the 2020 Capstone team working under Dr. Brano Pejcinovic, sponsored by Tangitek. This manual describes how to perform RCS simulations in ANSYS HFSS, how to perform RCS measurements, and how to calculate RCS results from measured data and compare with simulated RCS results. The document also describes how to use the modified NRL Arch for reflectivity measurements, in addition to VNA and turntable automation using VEE software.

Example data, results, and other related files are all available in the team Github repository at https://github.com/KirkJungles/TangitekCapstone2020

2 Radar Cross Section Measurement

Radar Cross Section(RCS) is a measure of the effective area of an object under the view of a radar system. It is dependent on a variety of factors, including the geometry of the object under test and the material the object is composed of. Tangitek has developed a material that can be placed on the exterior of various objects that would decrease their effective RCS and reflectivity without making alterations to their surface area and geometry, thereby reducing their visibility on radar systems.

Here, we propose a measurement system to determine the RCS of a target object using the S-parameter data gathered from two experimental setups. Ideally, both of these experiments will be performed in an anechoic chamber.

The first experiment will be the measurement of $S21$ for a pair of antennas separated by a measured distance R_1 for a specified frequency range. The next experiment will be the measurement of $S21$ for the same pair of antennas, this time approximately co-located, pointing at the target object at the previous distance R_1 for the same frequency range as in the first experiment.

After the experimental data is gathered, it will be processed by a MATLAB script that will return the RCS(σ) as a function of frequency. The experimental RCS values will then be compared to simulated RCS values as calculated by ANSYS HFSS.

To perform these experiments, one will need the following items:

- Network analyzer
- Pair of antennas
- Target object
- Antenna and target object stands suitable for EM testing environment(nonconductive)

2.1 Tx-Rx Gain Experiment

The purpose of this first experiment is to determine the experimental gain coefficients of the transmitting and receiving antennas that will be used to solve for σ from the data collected from the next experiment. We begin by noting the general case described by Frii's Equation, which is shown in Figure 1.

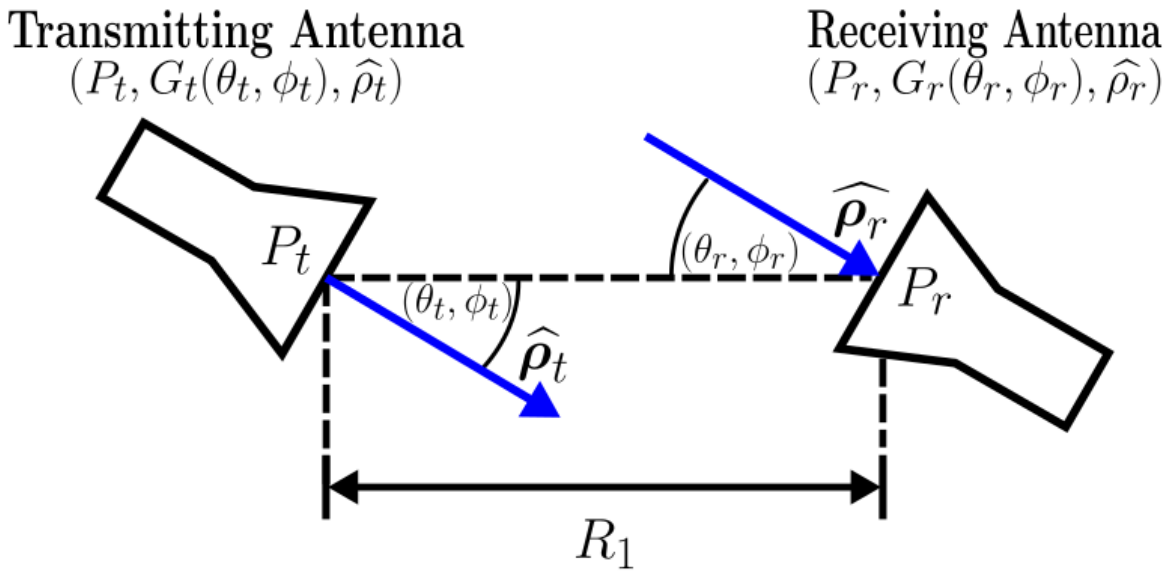


Figure 1: Tx-Rx Experimental Setup

Note that in Figure 1, P_t is the power transmitted by the transmitting antenna and P_r is the power received by the receiving antenna. Frii's Equation then gives us an analytical solution for P_r as

$$P_r = P_t \frac{G_t(\theta_t, \phi_t) G_r(\theta_r, \phi_r) \lambda^2}{(4\pi)^2 R_1^2} |\widehat{\rho_\mathbf{t}} \cdot \widehat{\rho_\mathbf{r}}| \tag{1}$$

For our experiment, we will be placing the two antennas on their respective stands at opposite sides of an anechoic chamber at a measured distance R_1[1]. The antennas will be directly facing each other such that $\theta_t = \phi_t = \theta_t = \phi_t = 0$. Then Equation 1 can be simplified as

$$P_r = P_t \frac{G_t G_r \lambda^2}{(4\pi)^2 R_1^2}$$

Dividing through by P_t yields $\frac{P_r}{P_t}$, then we can use the relationship $|S21|^2 = \frac{P_r}{P_t}$ to find the antenna pair gain coefficients.

$$|S21_{frii}|^2 = \frac{P_r}{P_t} = \frac{G_t G_r \lambda^2}{(4\pi)^2 R_1^2} \tag{2}$$

After successfully measuring $S21_{frii}$ in this experiment and saving our data, we may proceed to the next experiment[2].

[1]Ensure that the length R_1 has been properly measured before proceeding to the next experiment.

[2]G_t, G_r, and λ are all frequency dependent variables. Pay careful attention that the network analyzer performs the same frequency sweep for $S21_{frii}$ as in the next experiment to ensure accuracy of the final result.

2.2 Bistatic RCS Experiment

We begin this experiment by noting the general case described by the Radar Equation, which is shown in Figure 2.

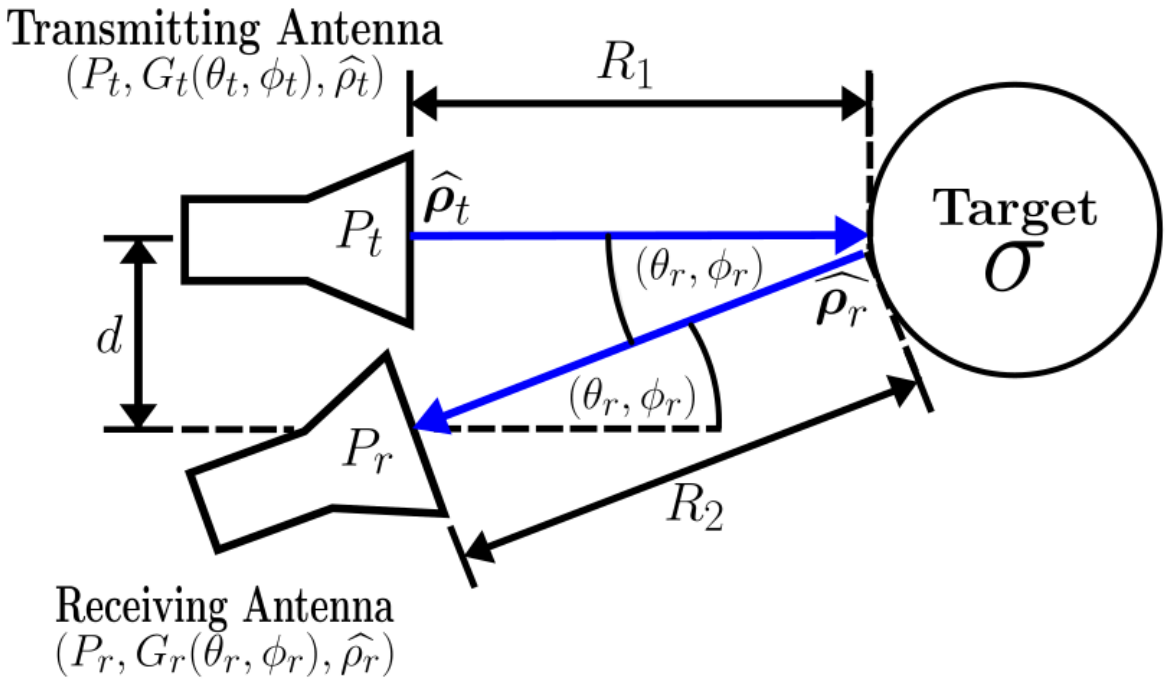


Figure 2: Radar Experimental Setup

In this general case, the Radar Equation gives us an analytical solution for P_r as

$$P_r = P_t \frac{G_t(\theta_t, \phi_t) G_r(\theta_r, \phi_r) \lambda^2}{(4\pi)^2 R_1^2} \frac{1}{4\pi R_2^2} |\widehat{\rho_\mathbf{t}} \cdot \widehat{\rho_\mathbf{r}}| \sigma \tag{3}$$

For our experiment, we will set the transmitting and receiving antenna as close together as possible and facing approximately the same direction, and the target under test will be set apart from the transmitting antenna the same distance R_1 as in the previous experiment used to measure $S21_{frii}$. Then with $R_1 >> d$, $R_2 \to R_1$, and $\theta_t, \phi_t, \theta_r, \phi_r \to 0$. Then we may use the simplified Radar Equation, which gives us P_r as

$$P_r = P_t \frac{G_t G_r \lambda^2}{(4\pi)^2 R_1^2} \frac{\sigma}{4\pi R_1^2}$$

Dividing through by P_t yields $\frac{P_r}{P_t}$,then we can use the relationship $|S21|^2 = \frac{P_r}{P_t}$.

$$|S21_{rad}|^2 = \frac{P_r}{P_t} = \frac{G_t G_r \lambda^2}{(4\pi)^2 R_1^2} \frac{\sigma}{4\pi R_1^2} \tag{4}$$

After successfully measuring $S21_{frii}$ in this experiment and saving our data, we can then solve for the RCS(σ) of the target[3].

2.3 Analytical Solution for RCS

Having collected the data from the previous experiments, we begin with an analytical approach. We note that when the frequencies, antennas, and distance R_1 are the same in both the Tx-Rx and Radar experiments, we can divide Equation 4 by Equation 2 yielding

$$\frac{|S21_{rad}|^2}{|S21_{frii}|} = \frac{\sigma}{4\pi R_1^2}$$

Solving for σ and noting the frequency dependence, we can then write

$$\sigma(f) = 4\pi R_1^2 \frac{|S21_{rad}(f)|^2}{|S21_{frii}(f)|^2} \tag{5}$$

Having arrived at an analytical solution for σ from our collected data, we can then begin to process the data in MATLAB to produce meaningful results.

2.4 RCS Extraction Script

After collecting the data from the previous experiments, we may use a simple MATLAB script to extract RCS as a function of frequency using the results from Equation 5.

The script follows a simple algorithm, which is outlined as follows:

- Accepts distance R_1 in meters
- Accepts filename and filepath of $S21_{frii}$, $S21_{rad}$, and RCS output files
- Reads frequency and S21 data for each file
- Checks that frequency arrays are identical for both, else ERROR
- Calculates RCS for each frequency using Equation 5
- Plots RCS vs frequency
- Writes RCS data and corresponding frequency array onto new csv file

The resulting RCS and paired frequency csv file will be saved for later use and imported by the RCS Comparison script to compare the experimental RCS with the HFSS simulated RCS.

The MATLAB code that performs this action is uploaded in the team Github account as S21_to_RCS.m and is embedded as follows:

[3]Before proceeding, be sure that both the distance R_1 and the frequencies used to collect $S21_{rad}$ are the same as those used in the Tx-Rx experiment to collect $S21_{frii}$

```
%Kirk Jungles
%S21_to_RCS.m
%4/27/2020
%
%Program accepts S21_frii measurements from Tx-Rx
    Experiment
%and accepts S21_rad Measurements from Radar Experiment
%Calculates RCS(sigma) as function of frequency

clc, clear, close all

%% User Input
%Experimental Parameters described in RCS document
R1 = 5 %Measured R1 for both experiments (meters)

%Accept S21 CSV from Tx-Rx setup
fname_frii = '2-40GHz-S21_frii.csv' %filename of S21_frii
    file
fpath_frii = '\\thoth.cecs.pdx.edu\Home03\kjungles\My
    Documents\MATLAB\Capstone\' %Folder where frii file is
    stored

%Accept CSV from Radar setup
fname_rad = '2-40GHz-S21_rad.csv' %filename of S21_rad file
fpath_rad = '\\thoth.cecs.pdx.edu\Home03\kjungles\My
    Documents\MATLAB\Capstone\' %Folder where radar file is
    stored

%Output filename and path
fname_out = '2-40GHz-RCS.csv' %filename of RCS output file
fpath_out = '\\thoth.cecs.pdx.edu\Home03\kjungles\My
    Documents\MATLAB\Capstone\' %Folder where radar file is
    stored

fpath_name_out = [fpath_out fname_out]

%% Extract data from files

%Assumes files are in csv format as vertical vectors [
    frequency, S21]
%May need changed
%If filetypes not CSV, do something similar(TBA)

%Otherwise, if as assumed, do:
```

```
%%%%frii data%%%%
%Read frii S21 and drequency data from csv file
fpath_name_frii = [fpath_frii fname_frii] %concatenate file path and file name
file_data = csvread(fpath_name_frii, 1,0) %Read data starting at Row offset = 1, Column offset = 0; omits text header

%Store frii data in respective frequency and RCS vectors
freq_frii = file_data(:,1)
S21_frii = file_data(:,2)

%%%%rad data%%%%
%Read S21_rad and drequency data from csv file
fpath_name_rad = [fpath_rad fname_rad] %concatenate file path and file name
file_data = csvread(fpath_name_rad, 1,0) %Read data starting at Row offset = 1, Column offset = 0; omits text header

%Store rad data in respective frequency and RCS vectors
freq_rad = file_data(:,1)
S21_rad = file_data(:,2)

%% Verify that frequency vectors are identical

%Check if frequency vectors same length, else error message
if length(freq_rad) ~= length(freq_frii)
    error('Frequency vectors not same length! Check data and consider repeating experiment.')
end

%Check if frequency vectors identical, else error message
for k = [1:length(freq_rad)]
    if freq_rad(k) ~= freq_frii(k)
        error('Frequency vector elements not matched! Check data and consider repeating experiment')
    end
end

%If network analyzer simply can't reproduce identical frequency sweeps for
%some reason, comment out previous error checks and interpolate the S21_rad
```

```
%array to match the S21_frii frequency components(RARE/
    Unexpected)

%S21_rad = interp1(freq_rad, S21_rad, freq_frii) %y_new =
    interp1(old x vector, old y vector, new x vector)

%% Calculate sigma as function of frequency

%Using equation for RCS(sigma) in RCS document, calculate
    sigma from data

sigma = 4*pi*R1^2*(S21_rad./S21_frii) %Units in meter^2
%A./B divides elementwise: [A(1)/B(1), A(2)/B(2), ... ]


%% Export results to CSV

%format will be same as expected csv files with 'frequency(
    MHz)' and 'RCS(m^2)' headers on first line
%and vertical frequency and RCS vectors:
%[frequency, RCS]
header = ['frequency(MHz)', 'RCS(m^2)']
data = [freq_frii, sigma]

%Concatenate header and data to single matrix
CSV_matrix = [header; data]

%Write matrix to user-specified filepath\filename
writematrix(CSV_matric, fpath_name_out)

fprintf('RCS data successfully written to %s \n',
    fpath_name_out)
```

3 Radar Cross Section Simulation in Ansys HFSS

3.1 Simulation Instructions

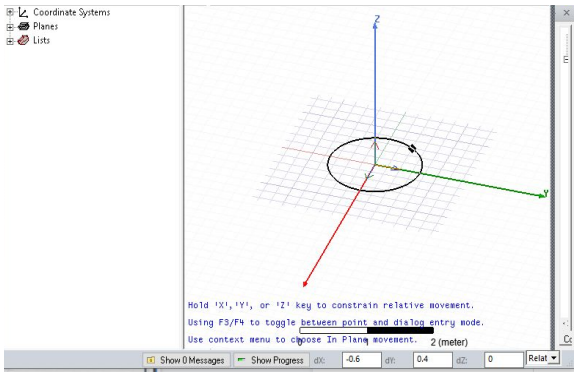
Monostatic and Bistatic RCS simulations can be performed in ANSYS HFSS. The geometry of the RCS targets and experiments can be adjusted for any experimental setup. What follows is a set of instructions that performs monostatic RCS simulations of a copper sphere using ANSYS Electronics Desktop 2018.1. The original powerpoint is also available in the team Github repository under the title `Simulation_Instructions.pdf`.

ANSYS HFSS RCS Simulation Instructions

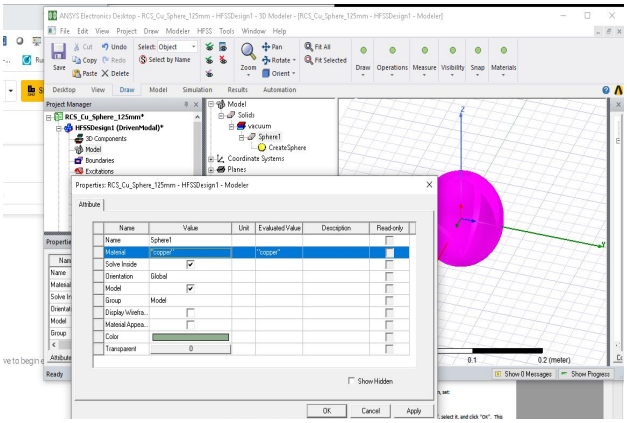
- Launch ANSYS HFSS from Start Menu or Icon
- Set Tool Options
  - On top menu bar, select Tools > Options > General Options
    - HFSS > Boundary Assignment
      - Check boxes that say “Use Wizards for data input when creating new boundaries” and “Duplicate Boundaries/mesh operations with geometry”
    - 3D Modeler > Drawing
      - Chex boxes that say “Edit properties of new primitives”

- Create and Save HFSS Design
  - On top menu bar, select **Project > Insert HFSS Design**
  - Select **File > Save As** and choose filename RCS_Cu_Sphere_125mm.aedt
- Set Solution Type and 3D Model
  - Select **HFSS > Solution Type**
    - Check Solution Type: Modal
    - Check Driven Options: Network Analysis
  - Select **Modeler > Units**
    - Select Meters
  - In the Modeler Toolbar, select **Materials**
    - Set default to Vacuum

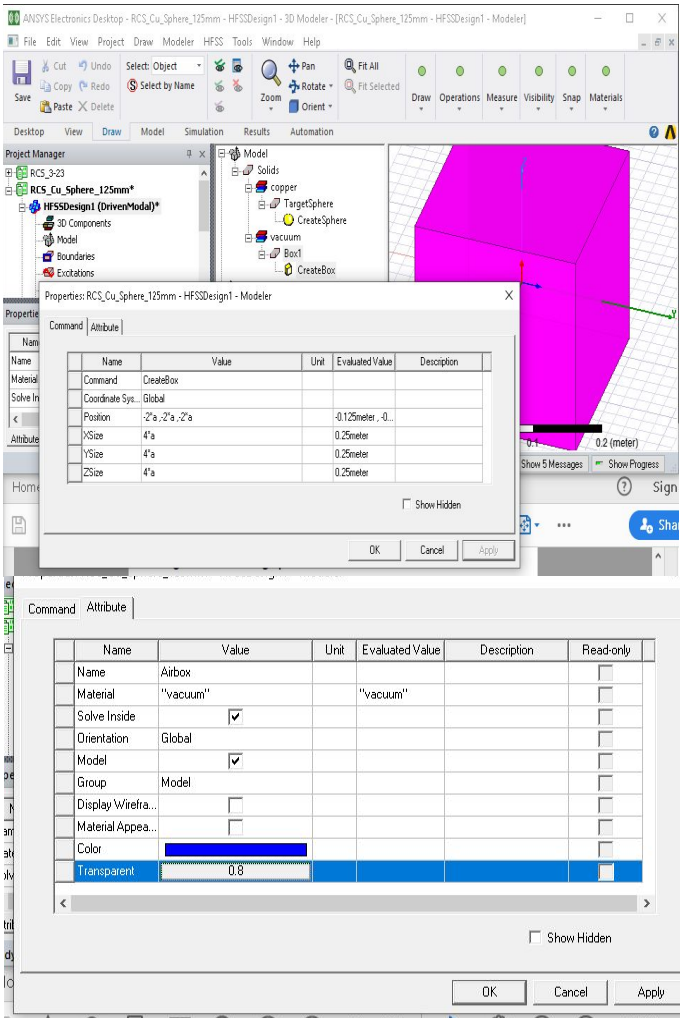
- Draw the Conducting Sphere
  - Select **Draw > Sphere**
    - Click on Origin of XYZ coordinates, then move mouse away from origin and click again to draw sphere
    - A **Properties** window should appear
      - Center Position: (0,0,0)
      - Radius: type ‘a’ as thevariable for radius
      - Unit Type: Length
      - Unit: meter
      - Value: 0.0625



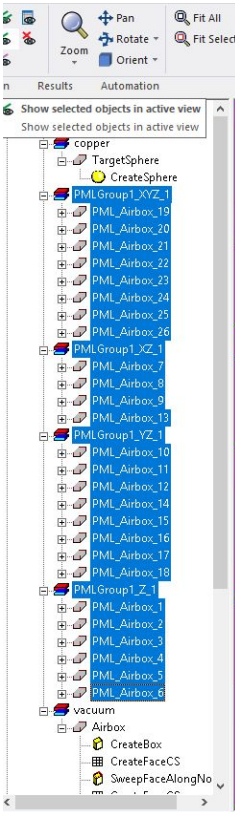
- Edit the Sphere Properties
  - In the modeler window, right-click **Sphere1 > Properties**
    - Name: TargetSphere
    - Material: “copper”
    - Color: Red



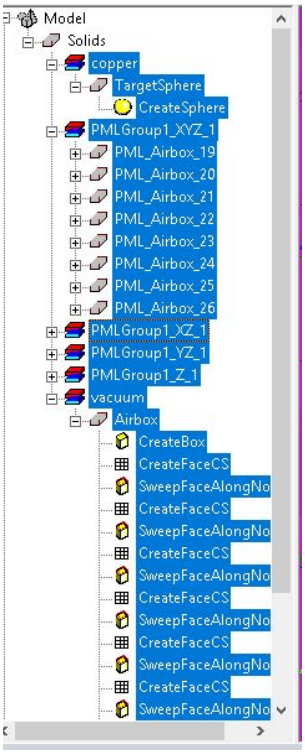
- Create the Airbox
  - Select **Draw > Box**
    - Draw 3D box anywhere in coordinate system
    - **Properties** window should come up again
      - Position: (-2*a, -2*a, -2*a)
      - XSize: 4*a
      - YSize: 4*a
      - ZSize: 4*a
  - In the Modeler Window, right click **Box1 > Properties > Attribute**
    - Name: Airbox
    - Material: Vacuum
    - Color: Blue
    - Transparent: 0.8

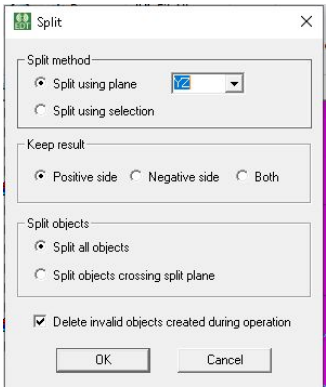


- Create PML (Perfectly Matched Layer)
  - A PML Box emulates an infinite vacuum or ideal anechoic chamber
  - In the Toolbar, select **Edit > Selection Mode > Faces**
  - **Edit > Select Objects > By Name**
    - Select **Airbox** then highlight all Faces and click OK
  - In the Toolbar, select **HFSS > Boundaries > PML Setup Wizard**
  - Check “Create PML Cover Objects On Selected Faces”
    - Uniform Layer Thickness : 0.250 meter (4*radius length)
  - Click **Next**
  - Check “PML Objects Accept Free Radiation”
    - Min Frequency: 0.04 GHz
    - Minimum Radiating Distance: 0.125 meter (2*radius length)
  - Click **Finish**
  - In Modeler Window, highlight all PMLGroup___ items
    - Click Green Eye up top to make visible

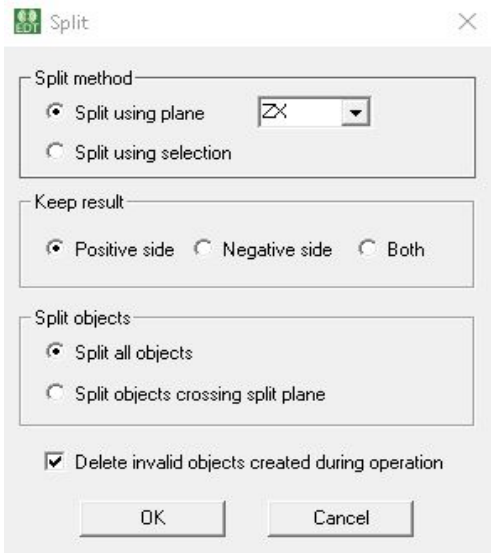


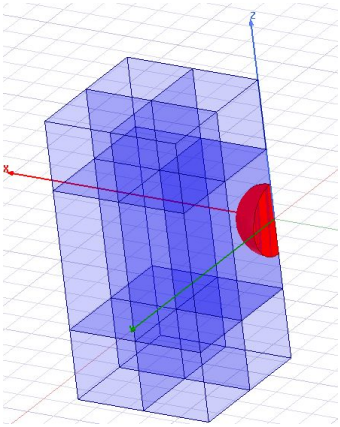
- Use symmetry to simplify solution
  - Because our target object is symmetrical, we can apply a symmetrical boundary condition on only a quarter of the sphere to reduce computation time
  - We will cut sphere, then set Perfect E and H boundaries on the appropriate sphere faces
  - In Modeler Window, highlight ALL Solids(include all items in drop-downs of PMLGroup_ objects)
    - In Toolbar, select **Modeler > Boolean > Split**
      - Split Plane: YZ
      - Keep Result: Positive Side
      - Splot Objects: Split all objects
      - Delete invalid objects



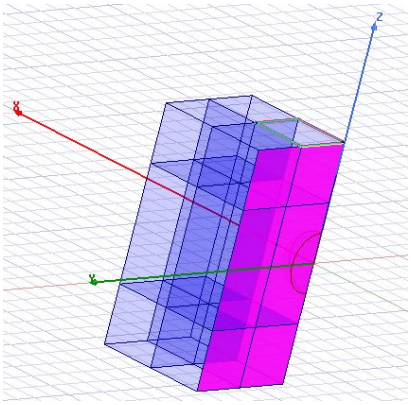


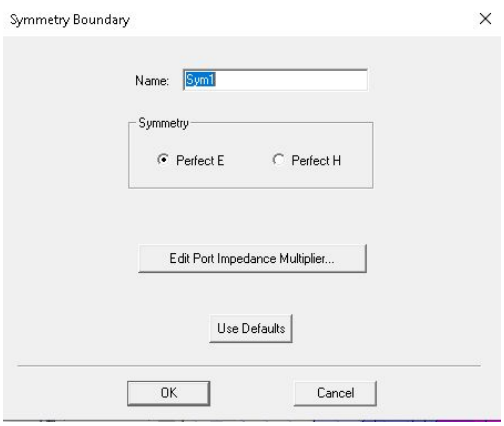
- Use Symmetry(continued)
  - In Modeler Window, highlight ALL Solids again (include all items in drop-downs of PMLGroup_ objects and new objects created from last split)
    - In Toolbar, select **Modeler > Boolean > Split**
      - Split Plane: ZX
      - Keep Result: Positive Side
      - Splot Objects: Split all objects
      - Delete invalid objects
  - You should now have a perfectly quartered sphere, Airbox, and PML Layer



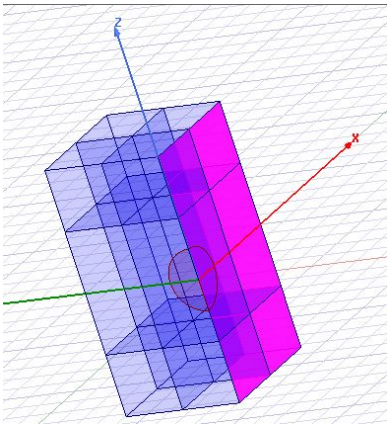


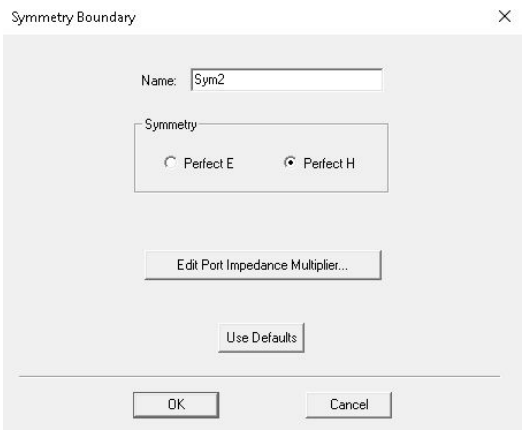
- Assign Boundary Conditions (YZ)
  - Select all the Faces in the YZ Plane:
    - On the Toolbar, select **Edit > Selection Mode > Faces**
    - In the 3D Model Window, hold CTRL and select all 6 faces on the YZ Plane
    - On the Toolbar, select **HFSS > Boundaries > Assign > Symmetry**
      - Select the “Perfect E” Button



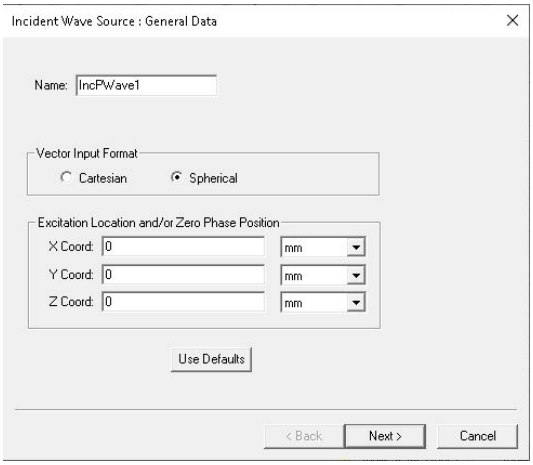


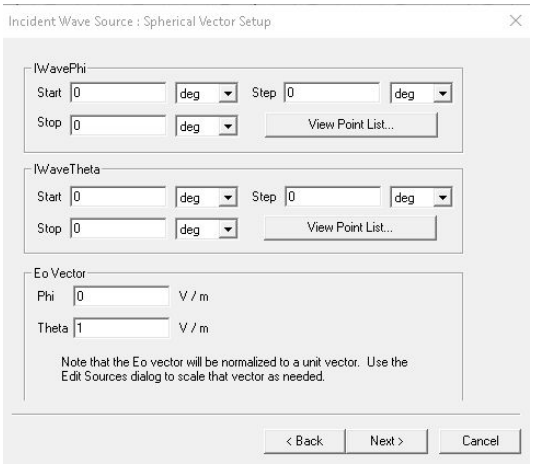
- Assign Boundary Conditions (XZ)
  - Select all the Faces in the XZ Plane:
    - On the Toolbar, select **Edit > Selection Mode > Faces**
    - In the 3D Model Window, hold CTRL and select all 6 faces on the XZ Plane
    - On the Toolbar, select **HFSS > Boundaries > Assign > Symmetry**
      - Select the “Perfect H” Button



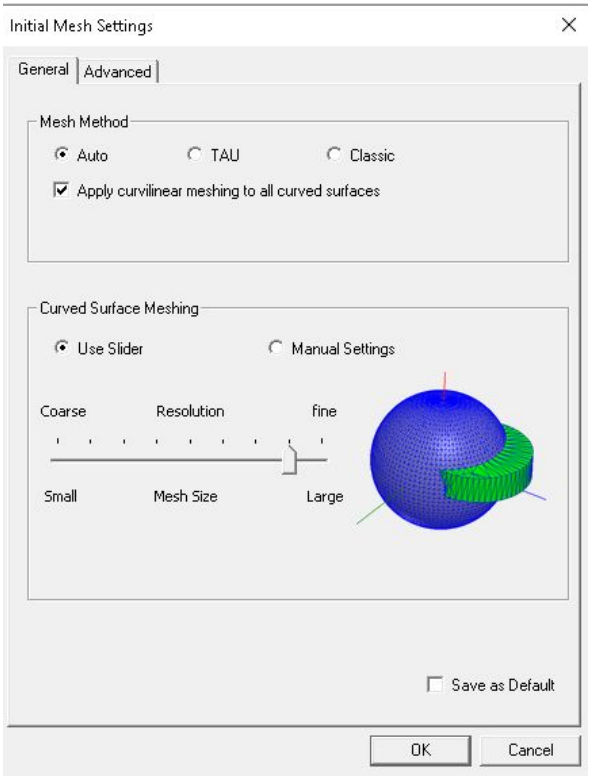


- Assign Plane Wave
  - On the Toolbar, select **HFSS > Excitations > Assign > Incident Wave > Plane Wave**
    - Vector Input Format: Spherical
  - Click Next
    - IWavePhi Start, Stop, Step: 0
    - IWaveTheta Start, Stop, Step: 0
    - Eo Vector
      - Phi: 0
      - Theta: 1
  - Click Next
    - Type of Plane Wave: Regular/Propagating
  - Click Finish

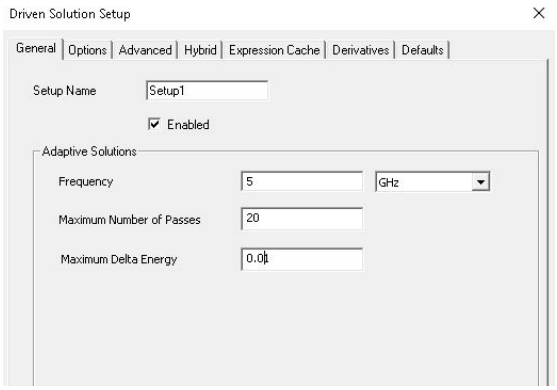


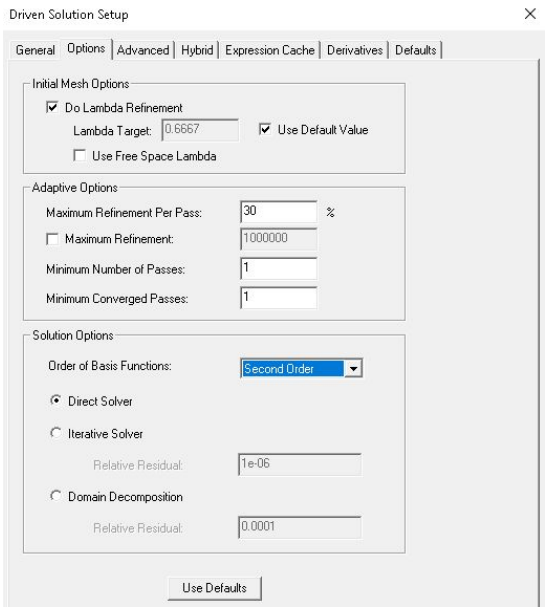


- Assign Meshing
  - On the Toolbar, select **HFSS > Mesh Operations > Initial Mesh Settings**
    - Check “Apply curvilinear meshing to all curved surfaces”
    - Set Mesh Size to Large/fine for greater accuracy at higher frequencies but longer computation time

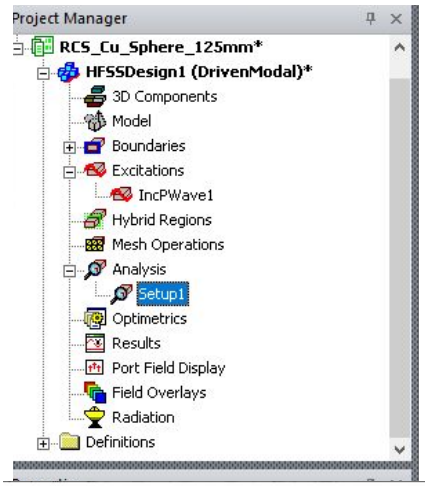


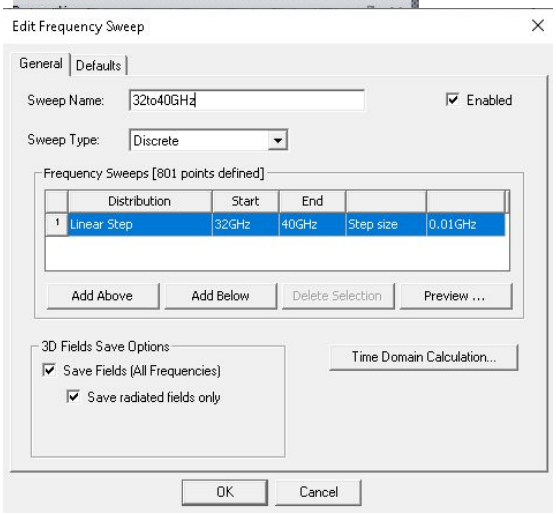
- Add Solution Setup
  - In the Toolbar, select **HFSS > Analysis Setup > Add Solution Setup**
    - Frequency: 5 GHz (This is not the excitation frequency)
    - Maximum Number of Passes: 20
    - Maximum Delta Energy: 0.01
  - Select Options tab
    - Order of Basis Functions: Second Order



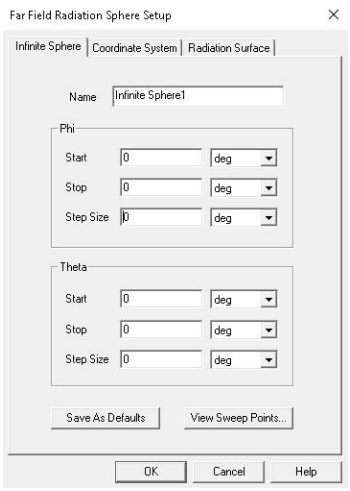


- Add Solution Setup
  - In the Project Manager window, expand the HFSSDesign1 dropdown menu
  - Expand “Analysis”
  - Highlight “Setup1”
    - In the Toolbar, select **HFSS > Analysis Setup > Add Frequency Sweep**
      - Sweep Name: 32to40GHz
      - Sweep Type: Discrete
      - Distribution: Linear Step
      - Start: 32 GHz
      - End: 40 GHz
      - Step Size: 0.01 GHz
      - Check “Save Radiated Field Only” box

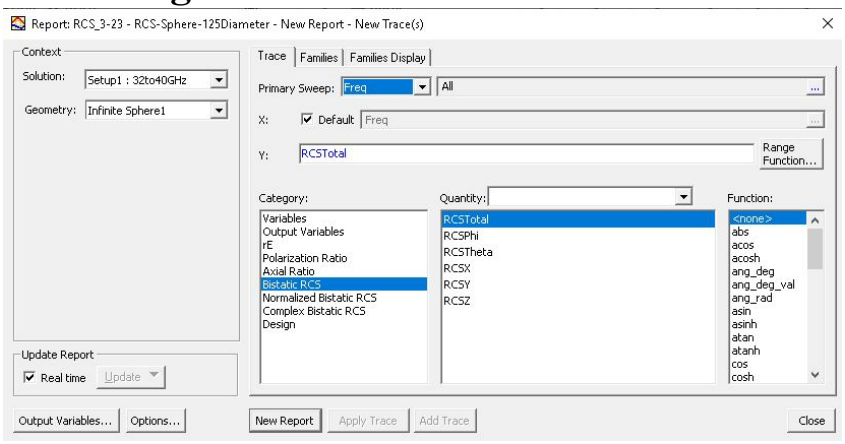




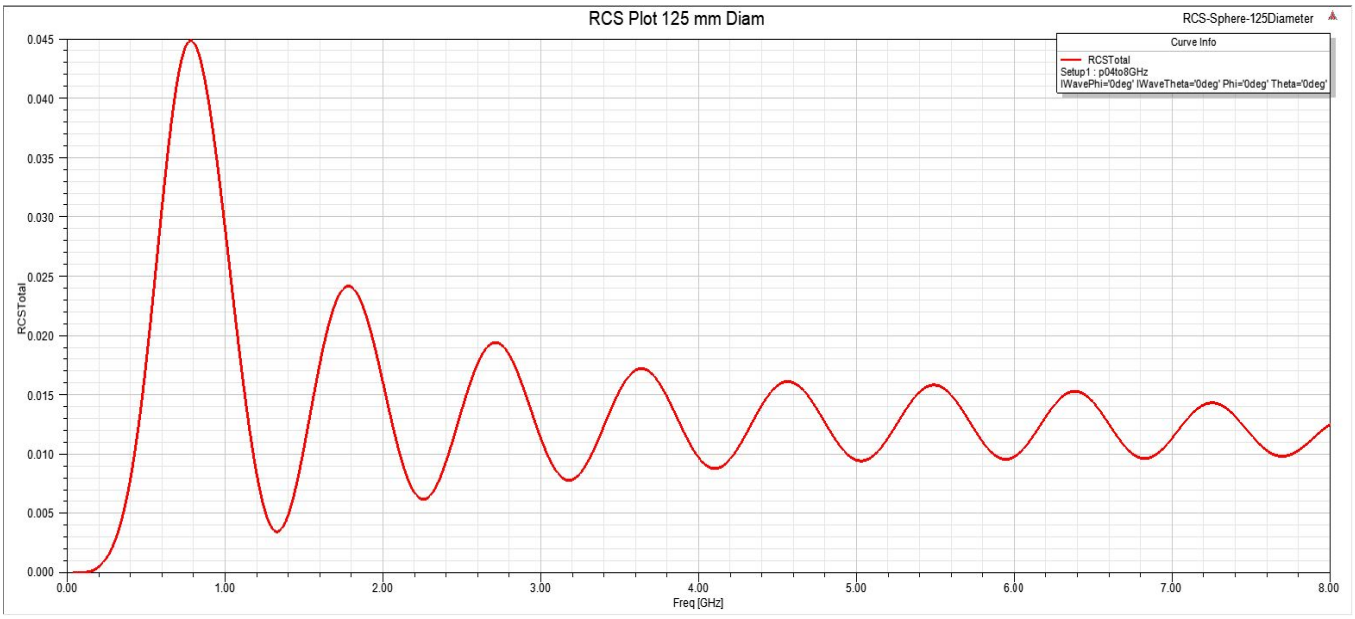
- Add Far-Field Setup
  - In the Toolbar, select **HFSS > Radiation > Insert Far Field Setup > Infinite Sphere**
    - Phi Start, Stop, Step Size: 0, 0, 0
    - Theta Start, Stop, Step Size: 0, 0, 0



- Add Plot
  - In the Toolbar, select **HFSS > Results > Create Far Fields Report > Rectangular Plot**
    - Primary Sweep: Freq
    - Category: Bistatic RCS
    - Quantity: RCSTotal
    - Click New Report



- Save RCS Data for Processing
  - Last step should have resulted in a Plot of RCS(in meters^2) as a function of Frequency
  - Right click plot to Export as image and/or .csv file in desired location



3.2 Plotting Normalized RCS from Simulated RCS

After running the simulations in ANSYS and exporting the results in csv format to a known directory, the Normalized RCS is plotted by running the MATLAB script `Plot_Normalized_RCS.m`, which is available in the team Github repository. An example output plot from `Plot_Normalized_RCS.m` is given in Figure 3.

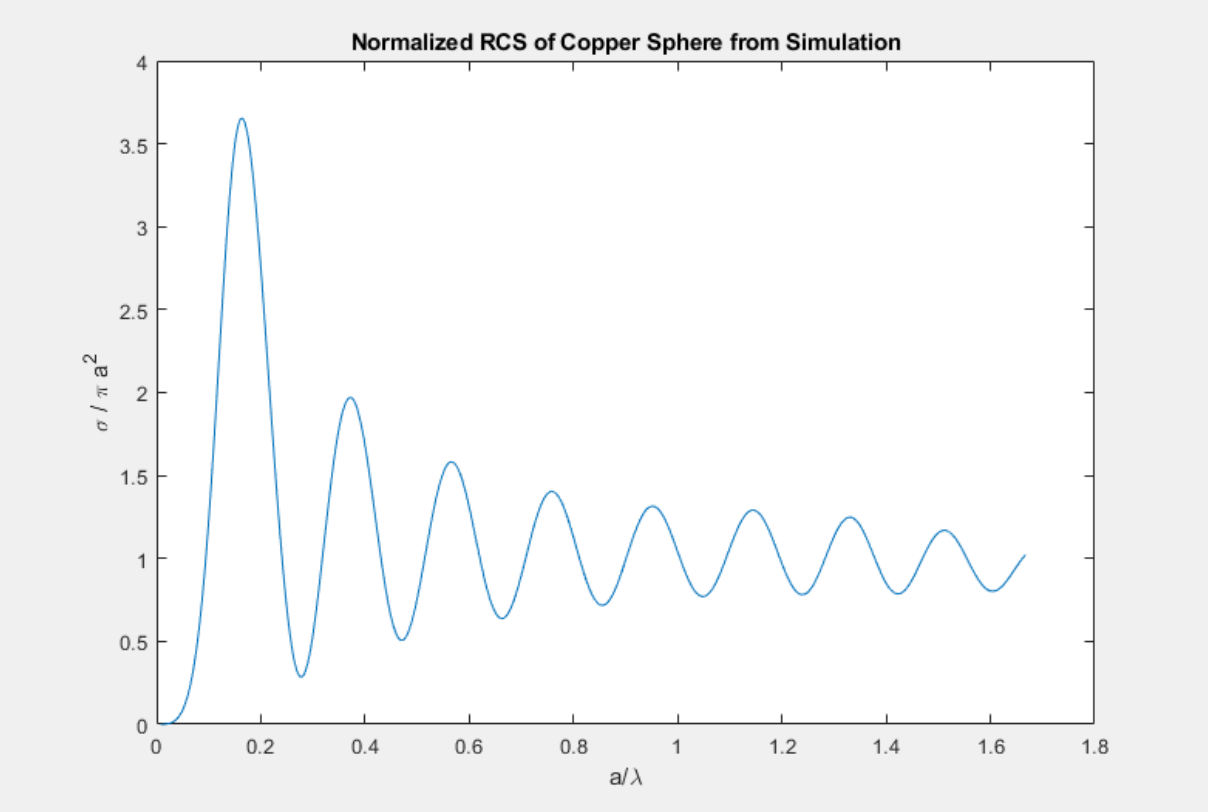


Figure 3: Normalized RCS from HFSS Simulations of 125 mm Diameter Copper Sphere

The code is is embedded as follows:

```
%Program accepts RCS csv files from ANSYS simulations and plots normalized
%RCS.

clc, clear, close all

%% User Parameters and filenames/filepaths

sphere_diam = 125; %Diameter of sphere in mm

%Load simulated frequency and RCS data into N x 2 matrix: [frequency, RCS_values]
```

```
file_name = '2-40GHz-full-125mm-diameter.csv'; %Name of csv file
%file_path = '\\thoth.cecs.pdx.edu\Home03\kjungles\My Documents\MATLAB\Capstone\'; %Folder where file is stored
file_path = '' %Leave uncommented if destination is PWD
fpath_name_sim = [file_path file_name]; %concatenate file path and file name

%% Read SIMULATED Data Files From CSV

file_data = csvread(fpath_name_sim, 1,0); %Read data starting at Row offset = 1, Column offset = 0; omits text header

%Store file data in respective frequency and RCS vectors
freq_sim = file_data(:,1);
RCS_sim = file_data(:,2);

%Plot RCS vs Frequency
figure(1)
plot(freq_sim,RCS_sim);
title('RCS of Copper Sphere, Simulated')
xlabel('Frequency (GHz)')
ylabel('Monostatic RCS (m^{2})')

%Plot normalized RCS vs Frequency
a = sphere_diam*10^-3/2
lam = 299792458./(freq_sim*10^9)
y_sim = RCS_sim/(pi*a^2)
x = a./lam

figure(2)
plot(x,y_sim);
title('Normalized RCS of Copper Sphere from Simulation')
ylabel('\sigma / \pi a^2')
xlabel('a/\lambda')
```

4 Comparing Experimental and Simulated RCS

Final analysis of the RCS measurements is performed by comparing the experimental RCS data with simulated RCS data. The MATLAB Script `RCS_compare.m`, which is available in the team Github repository, performs this action. With small changes to the code, experimental RCS data can also be compared to other experimental data. Such changes are indicated in the comments of the code.

An example output plot from `RCS_compare.m` is given in Figure 4.

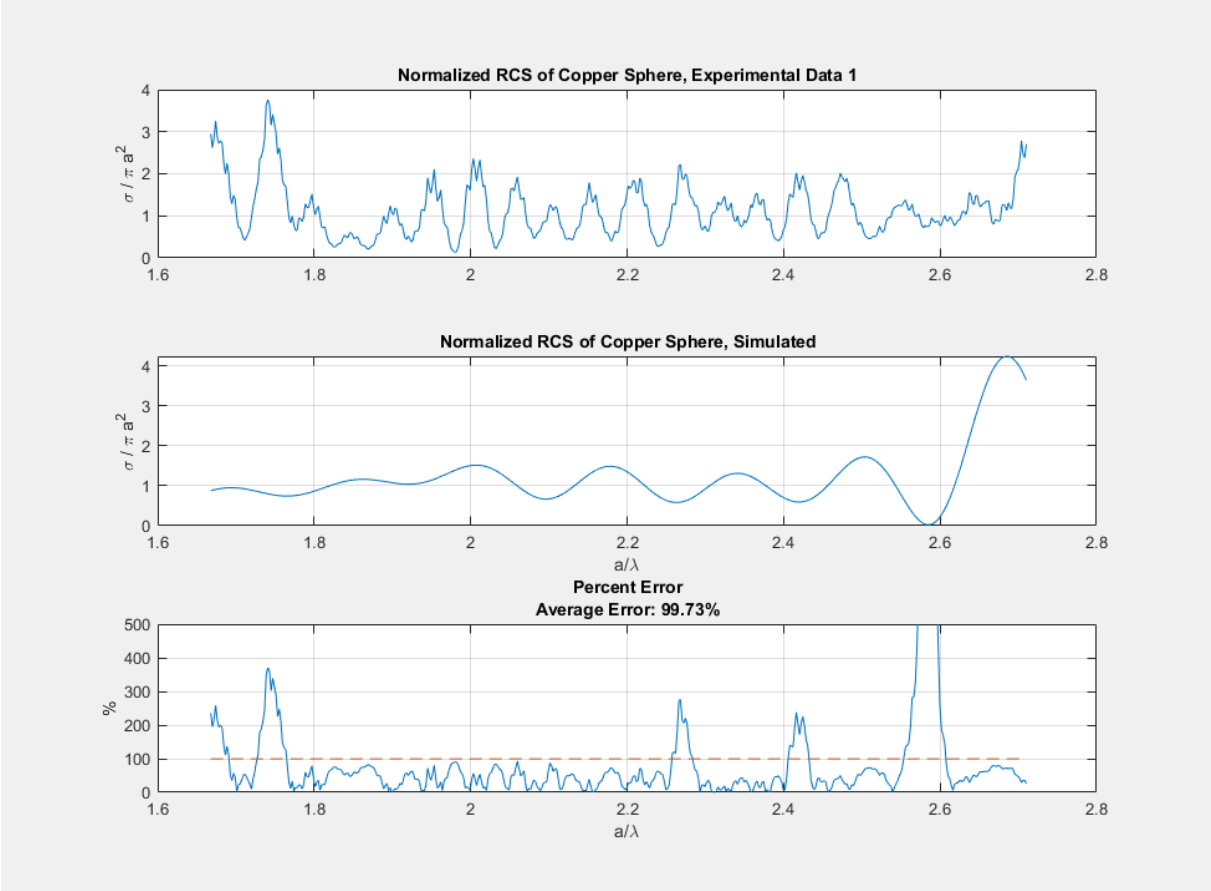


Figure 4: Normalized RCS Comparison

It is worth noting that the experimental RCS for Figure 4 was gathered in a living room, and the simulated RCS was performed with a coarse mesh, so this example plot should not to be regarded as ideal.

The program is embedded as follows:

```
%Program reads CSV files of RCS vs frequency for both measured and
%simulated data. Compares RCS values and determines how well measurement
```

```
%matches simulation.
clc, clear, close all

%% User Parameters and filenames/filepaths

sphere_diam = 125; %Diameter of sphere in mm

%Load frequency and EXPERIMENTAL RCS dataset 1 into N x 2 matrix: [frequency RCS_values]
file_name = '8-13GHz_RCS-EXP_5-19_NO-GATE.csv'; %Name of csv file
%file_path = '\\thoth.cecs.pdx.edu\Home03\kjungles\My Documents\Capstone\RCS_results\'; %Folder where file is stored
file_path = '' %Leave uncommented if destination is PWD
fpath_name_exp1 = [file_path file_name]; %concatenate file path and file name

% %Load frequency and EXPERIMENTAL RCS dataset 2 into N x 2 matrix: [frequency RCS_values]
% file_name = '8-13GHz_RCS-EXP_5-19_GATED.csv'; %Name of csv file
% %file_path = '\\thoth.cecs.pdx.edu\Home03\kjungles\My Documents\Capstone\RCS_results\'; %Folder where file is stored
% file_path = '' %Leave uncommented if destination is PWD
% fpath_name_exp2 = [file_path file_name]; %concatenate file path and file name

%Load frequency and SIMULATED RCS data into N x 2 matrix: [frequency RCS_values]
file_name = '2-40GHz-full-125mm-diameter.csv'; %Name of csv file
%file_path = '\\thoth.cecs.pdx.edu\Home03\kjungles\My Documents\MATLAB\Capstone\'; %Folder where file is stored
file_path = '' %Leave uncommented if destination is PWD
fpath_name_sim = [file_path file_name]; %concatenate file path and file name

%% Read EXPERIMENTAL Data File 1 From CSV

%Read and plot RCS for Experimental Dataset 1
file_data = csvread(fpath_name_exp1, 1,0); %Read data
```

```
    starting at Row offset = 1, Column offset = 0; omits
    text header

%Store file data in respective frequency and RCS vectors
freq_exp1 = file_data(:,1);
RCS_exp1 = file_data(:,2);

%Plot RCS vs Frequency
figure(1)
plot(freq_exp1,RCS_exp1);
title('RCS of Copper Sphere, Experimental Data 1')
xlabel('Frequency (GHz)')
ylabel('Monostatic RCS (m^{2})')

%% Read EXPERIMENTAL Data File 2 From CSV
% file_data = csvread(fpath_name_exp2, 1,0); %Read data
    starting at Row offset = 1, Column offset = 0; omits
    text header
%
% %Store file data in respective frequency and RCS vectors
% freq_exp2 = file_data(:,1);
% RCS_exp2 = file_data(:,2);
%
% %Plot RCS vs Frequency
% figure(1)
% plot(freq_exp2,RCS_exp2);
% title('RCS of Copper Sphere, Experimental Data 2')
% xlabel('Frequency (GHz)')
% ylabel('Monostatic RCS (m^{2})')

%% Read SIMULATED Data Files From CSV

file_data = csvread(fpath_name_sim, 1,0); %Read data
    starting at Row offset = 1, Column offset = 0; omits
    text header

%Store file data in respective frequency and RCS vectors
freq_sim = file_data(:,1);
RCS_sim = file_data(:,2);

%Plot RCS vs Frequency
figure(2)
plot(freq_sim,RCS_sim);
title('RCS of Copper Sphere, Simulated')
```

```
xlabel('Frequency (GHz)')
ylabel('Monostatic RCS (m^{2})')

%%Plot normalized RCS vs Frequency
% a = sphere_diam*10^-3/2
% lam = 299792458./(freq_sim*10^9)
% y_sim = RCS_sim/(pi*a^2)
% x = a./lam
%
% figure
% subplot(2,1,1)
% plot(x,y_sim);
% title('Normalized RCS of Copper Sphere, Simulated')
% ylabel('\sigma / \pi a^2')
% xlabel('a/\lambda')
% xlim([0.5,3])
% ylim([0,4])
%
% subplot(2,1,2)
% plot(x,y_sim);
% title('Normalized RCS of Copper Sphere, Simulated')
% ylabel('\sigma / \pi a^2')
% xlabel('a/\lambda')



%% Shorten Simulated data to frequency range of
   experimental

a = find(freq_sim == freq_exp1(1));
b = find(freq_sim == freq_exp1(end));

freq_sim = freq_sim(a:b);
RCS_sim = RCS_sim(a:b);

%% Plot Both Experimental and Simulated RCS

figure(3)
subplot(2,1,1)
plot(freq_exp1,RCS_exp1);
title('RCS of Copper Sphere, Experimental Data 1')
xlabel('Frequency (GHz)')
ylabel('Monostatic RCS (m^{2})')

```

```
subplot(2,1,2)
plot(freq_sim,RCS_sim)
title('RCS of Copper Sphere, Simulated')
xlabel('Frequency (GHz)')
ylabel('Monostatic RCS (m^{2})')

%Uncomment and adjust subplot() arguments on previous to plot Experimental
%Data 2
% subplot(3,1,2)
% plot(freq_exp2,RCS_exp2);
% title('RCS of Copper Sphere, Experimental Data 2')
% xlabel('Frequency (GHz)')
% ylabel('Monostatic RCS (m^{2})')

%% Interpolate Experimental RCS to match Simulated

%Interpolate experimental data to match same frequency points as
%simulation
RCS_exp1_matched = interp1(freq_exp1, RCS_exp1, freq_sim);
%RCS_exp2_matched = interp1(freq_exp2, RCS_exp2, freq_sim);
freq = freq_sim;

figure(4)
subplot(2,1,1)
plot(freq,RCS_exp1_matched)
title(['RCS of Copper Sphere, Experimental Data 1'])
xlabel('Frequency (GHz)')
ylabel('Monostatic RCS (m^{2})')

subplot(2,1,2)
plot(freq,RCS_sim)
title(['RCS of Copper Sphere, Simulated'])
xlabel('Frequency (GHz)')
ylabel('Monostatic RCS (m^{2})')

%Uncomment and adjust subplot() arguments on previous to plot Experimental
%Data 2
% subplot(3,1,2)
% plot(freq,RCS_exp2_matched)
% title(['RCS of Copper Sphere, Experimental Data 2'])
% xlabel('Frequency (GHz)')
```

```
% ylabel('Monostatic RCS (m^{2})')

%% Calculate Normalized RCS and Percent Error

%Calculate normalized RCS and a/lambda x axes
a = sphere_diam/2*10^-3;
lam = 299792458./(freq*10^9);
RCS_exp1_norm = RCS_exp1_matched/(pi*a^2);
%RCS_exp2_norm = RCS_exp2_matched/(pi*a^2)
RCS_sim_norm = RCS_sim/(pi*a^2);
x = a./lam;

%Calculate percent Error and average
percent_err = abs(RCS_exp1_norm-RCS_sim_norm)./RCS_sim_norm
    *100;
avg_err = mean(percent_err);
avg_err_plot = ones(1,length(x))*avg_err;

figure(5)
subplot(3,1,1)
plot(x,RCS_exp1_norm)
title(['Normalized RCS of Copper Sphere, Experimental Data
    1'])
ylabel('\sigma / \pi a^2')
grid on

subplot(3,1,2)
plot(x, RCS_sim_norm)
title(['Normalized RCS of Copper Sphere, Simulated'])
ylabel('\sigma / \pi a^2')
xlabel('a/\lambda')
grid on

subplot(3,1,3)
plot(x, percent_err,x,avg_err_plot,'--')
title({'Percent Error',['Average Error: ' sprintf('%.2f',
    avg_err) '%']})
ylabel('%')
ylim([0 500])
xlabel('a/\lambda')
grid on

%Uncomment and adjust subplot() arguments on previous to
    plot Experimental
```

```
194 %Data 2
195 % subplot(3,1,2)
196 % plot(x,RCS_exp2_norm)
197 % title(['Normalized RCS of Copper Sphere, Experimental
    Data 2(GATED)'])
198 % ylabel('\sigma / \pi a^2')
199 % grid on
```

5 NRL Arch

An NRL Arch Reflectivity Measurement System has an NRL Arch style apparatus that supports a pair of opposing antennas at varying zenith angles. The arch itself is constructed of mostly wood, which is largely transparent in the frequency range of interest. The NRL Arch system is composed of several detachable wooden parts.

At all angles of incidence of the pairs of antenna, a 2-40 GHz VNA will inject signals into the transmitting antenna, which will then propagate towards a flat target at the center of the arch. The reflected signals will then be received by an identical receiving antenna and fed back into the VNA.

An NRL Arch system that has the turntable and VNA automation system integrated into it is shown in Figure 25.

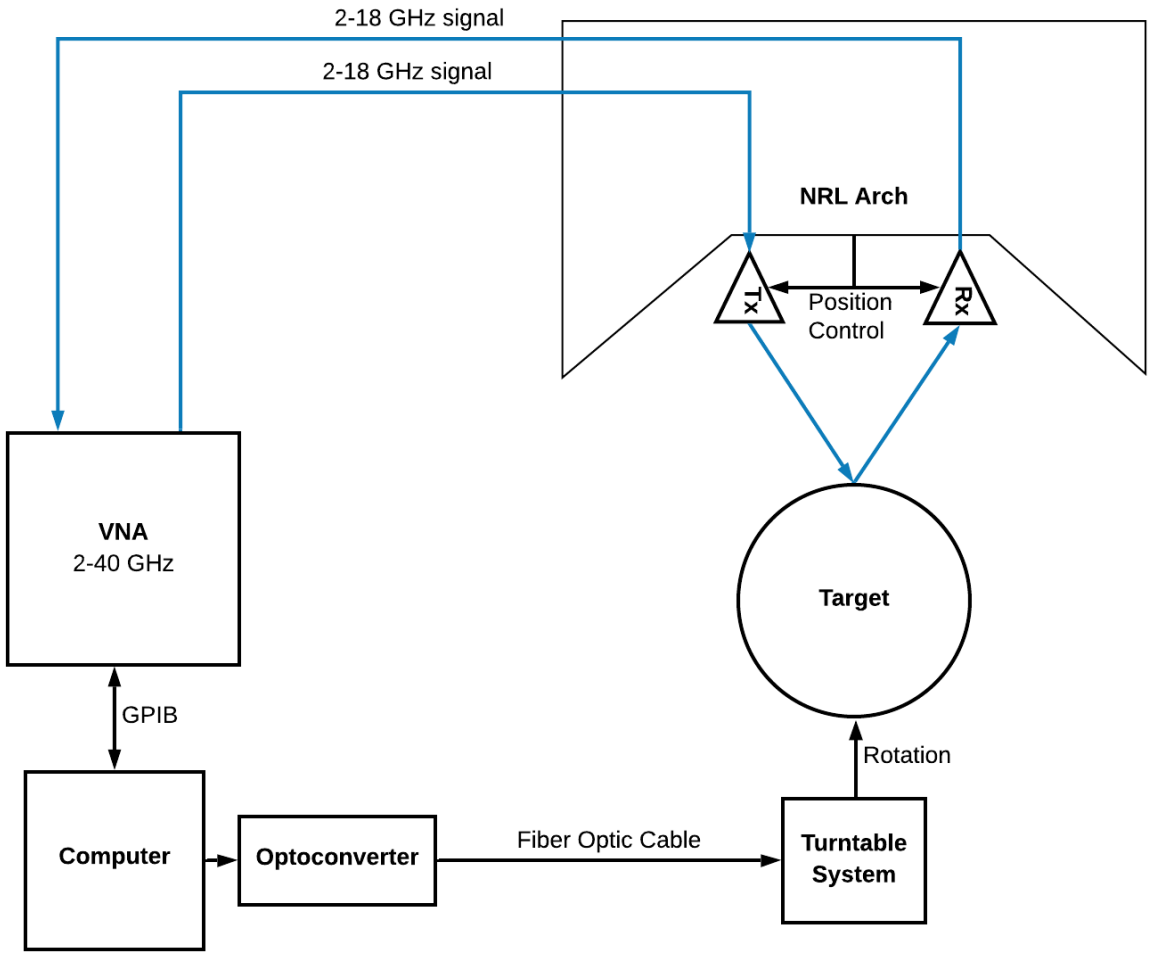


Figure 5: NRL Arch with Automation

Automation of the VNA and turntable is addressed in separate sections. The focus of this section is to describe how the arch structure itself has been modified and is operated.

Due to the pandemic, no recent pictures of the arch are available.

5.1 NRL Arch Assembly

Proper and safe assembly of the arch requires two persons, and is performed as follows:

- Erect the two bases of the arch and separate them about 8 feet apart
- Mount the semi arch onto one base and screw them together with the wooden screw to form a semi arch and repeat the same for the other base as well
- Carefully align the two separate semi arches together with a focus on the top wooden bars been aligned together
- Screw the wooden joint at their designed locations together
- Mount the two antennas at (opposite) required angles, one at the receiving end and the other at the transmitting end

5.2 NRL Arch Disassembly

To dismantle the setup, two people are needed to perform the following:

- The two antennas are taken off from the arch
- With extra care, separate the arch board into two separate semi arches by unscrewing the wooden joint holding the semi arches on top line of the arch
- Separate the semi arch from its base for each side by unscrewing them
- Carefully set the semi arch and the base on the floor
- Carefully unscrew the wooden locks that hold both the arch board and the frame together at their respective joints
- Set pieces aside at a proper and convenient location

5.3 NRL Arch Transportation

The NRL Arch should be carefully transported into the anechoic chamber in separate pieces after the setup has been completely dismantled. Care should be taken not to bend or break any of the individual parts.

If the NRL Arch setup is not dismantled and is instead transported in its assembled state, it may loosen and cause misalignment of the antenna mounts. Transporting the assembled arch may also result in damage to the antennas or the structure of the arch.

6 VNA Automation with VEE

6.1 Downloading VEE and Applications

1. In order to start using VEE Pro application you need to download VEE Pro by Keysight which can be accessed at this link: https://www.keysight.com/en/pd-1476554-pn-W4000D/vee-pro-932?pm=DL&nid=-32811.806312&cc=US&lc=eng

2. You have the option of entering a product key if available or getting a free 30-day student evaluation

3. The next application that needs to be downloaded to be able to communicate with an instrument is Keysight IO libraries suite which can be accessed at this link: https://www.keysight.com/main/software.jspx?cc=US&lc=eng&ckey=2175637&nid=-33330.977662&id=2175637

4. Once both these applications are installed you can begin on to the next step which involves connecting an instrument to work with VEE

6.2 Enabling Instrument Connection

1. In order to start the process, you need to open the VEE pro application and get to the home page to start a new window. This initial welcome home screen will appear which is shown in figure 1. Select the close button to continue onto the main window.

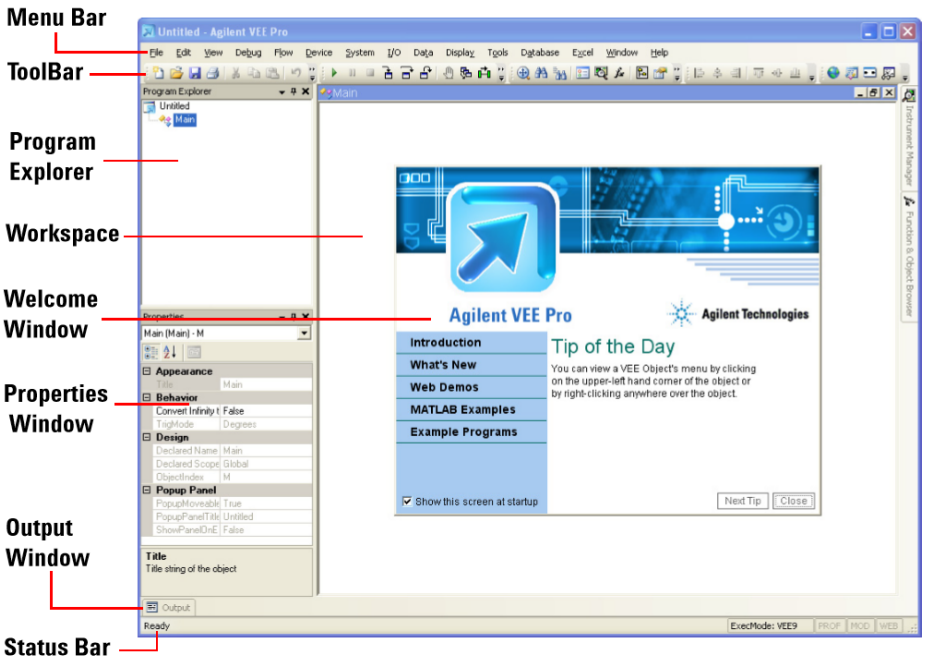


Figure 6: Vee home screen

2. In order to add an instrument to the program you can do it by opening the instrument manger tab on the right side of the screen.

Figure 7: Instrument manager tab

3. If the instrument manager tab is not there than you can access it from the I/O drop down menu and selecting instrument manger.

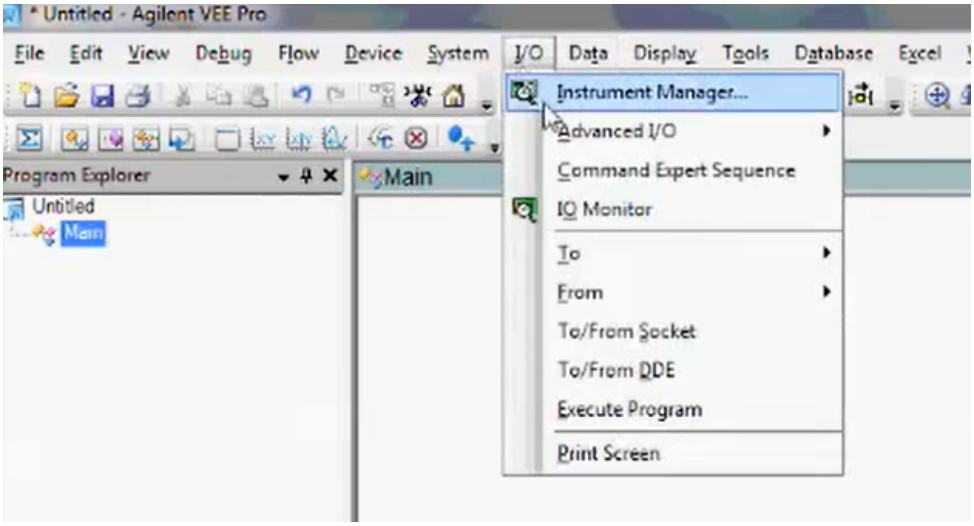


Figure 8: Turning on the instrument manger tab

4. To find an instrument click on find an instrument button which will find all instruments connected to computer or you can manually add an instrument with add instrument button.

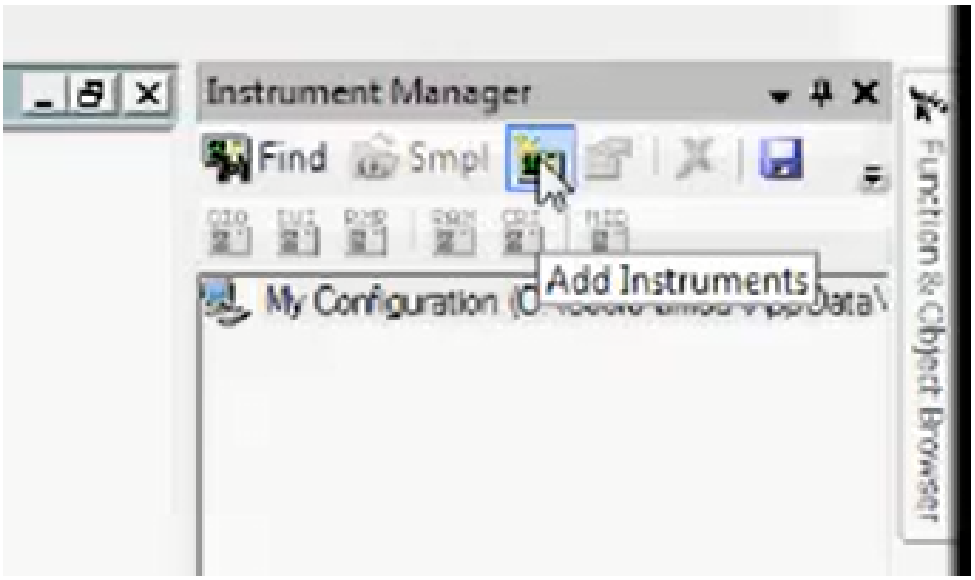


Figure 9: Adding and finding and instruments button

5. If you are going to add an instrument which is usually the case you will then have to add the interface type and select "OK" once selected.

Figure 10: Selecting interface type

6. You will then be directed to the instruments properties page and you have the option to change the name of the instrument and you can specify VISA Alias or VISA Address.

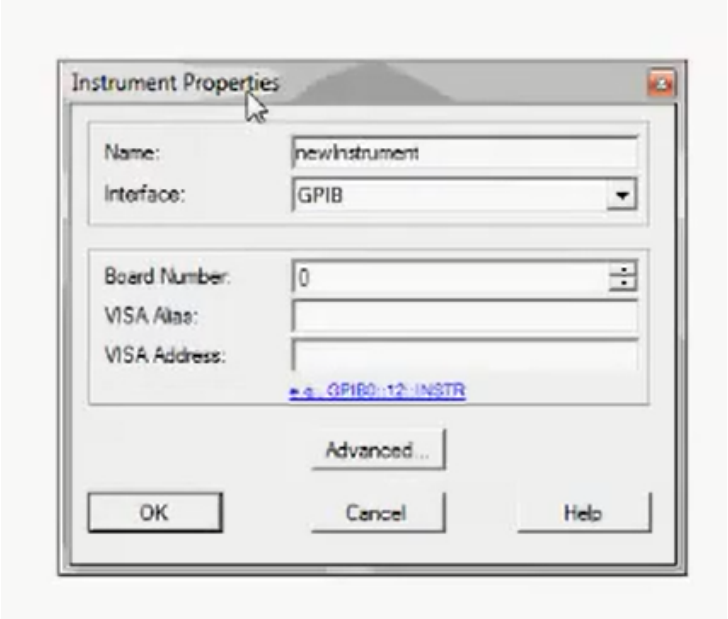


Figure 11: Instrument properties window

7. In order to get the VISA address or Alias you can open the Keysight IO libraries suite application to retrieve it. You would then copy and paste either one onto the appropriate section in the instrument properties window and hit "OK".

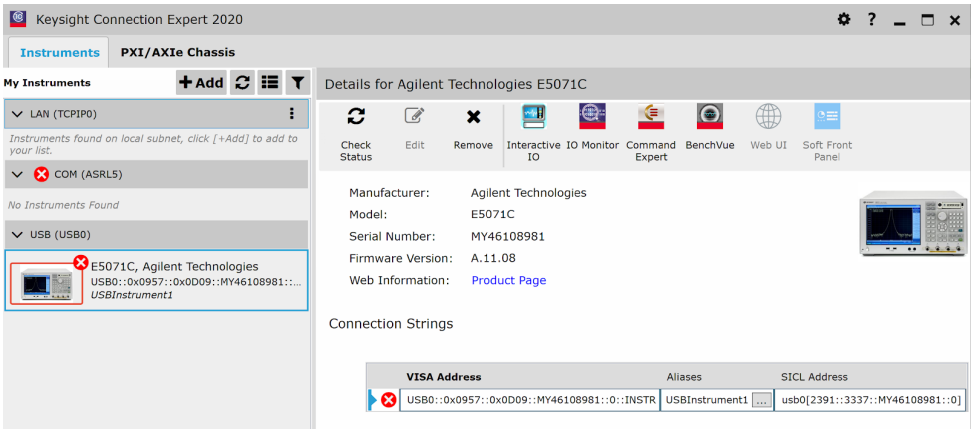


Figure 12: Finding VISA address or alias in IO libraries suite

8. You will then have the instrument added to the program as shown in figure 13

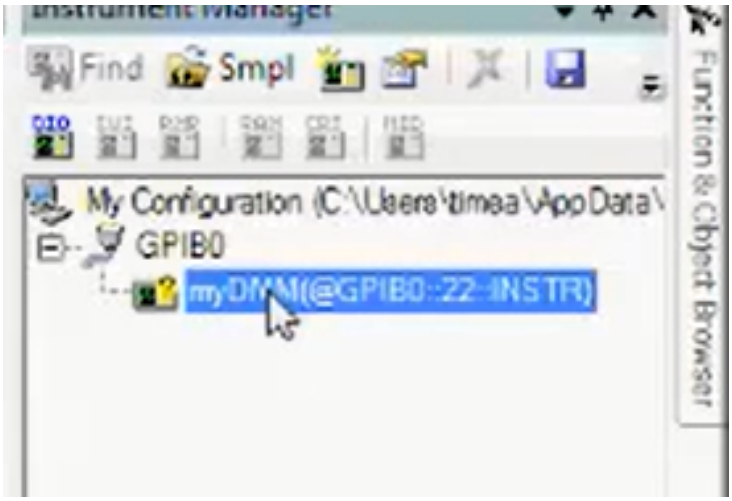


Figure 13: Instrument created

9. You can then create a direct IO object by right clicking on the instrument you just added as shown on the left of figure 14 and the dragging the Direct IO block onto the program window as shown on the right of figure 14.

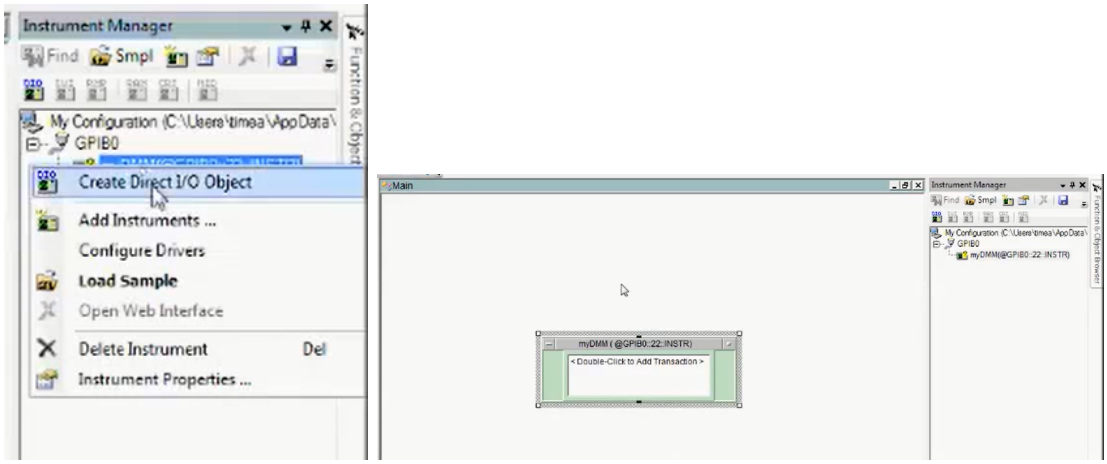


Figure 14: Selecting interface type

10. You will then be able to add transactions to the Direct IO block as a write or read commands by double clicking. You will the be able to create a small program that will enable you to recognize if the instrument is connected to the program by retrieving its identification number. You start by with a write transaction and enter what is shown in the figure below and hit “OK”.

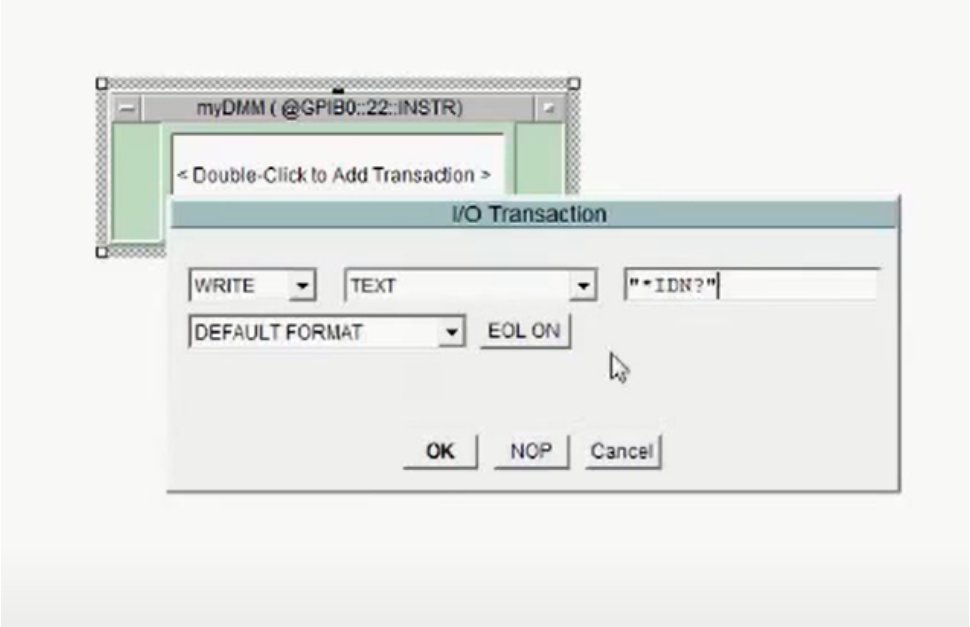


Figure 15: Adding write transaction

11. Next you will add another command that is a read transaction and enter what is shown in the figure below and hit "OK".

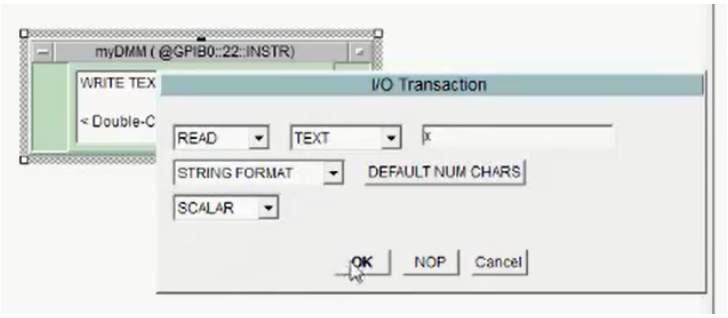


Figure 16: Adding read transaction

12. You will then go to the top and click on display and in the drop-down menu choose the alpha numeric display and place it onto the program window.

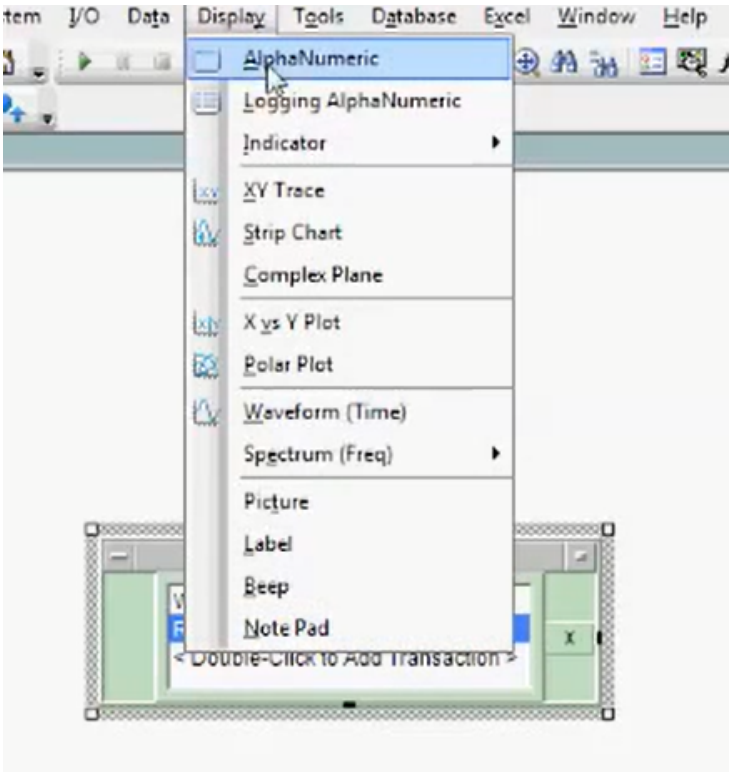


Figure 17: Adding alpha-numeric display

13. You will then wire it up to the output of the direct IO object as shown below.

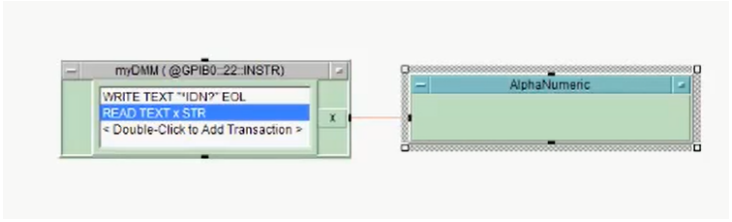


Figure 18: Adding a wire between the two blocks

14. You then hit the green play button at the top of the screen to run the program and in the alpha numeric display it will show the instruments identification number if it is hooked up correctly.

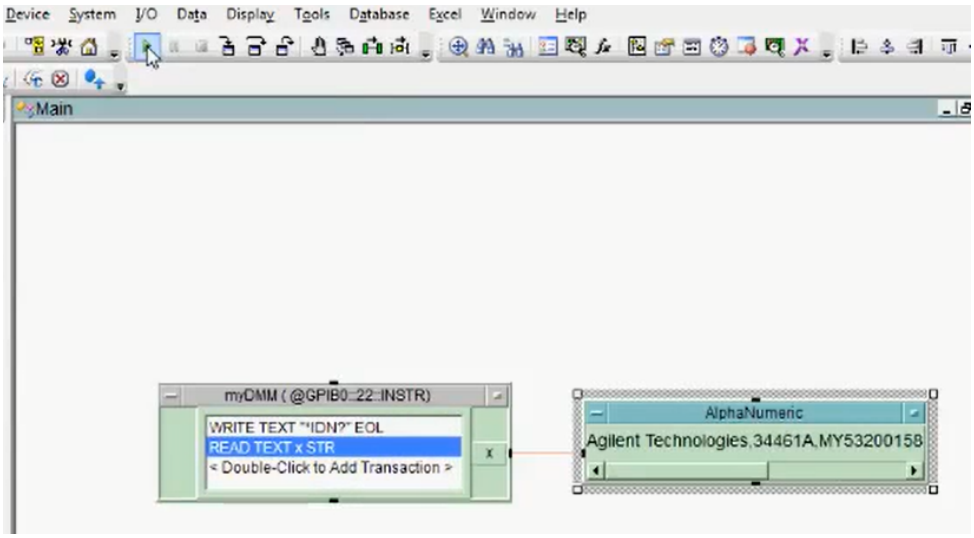


Figure 19: Result of retrieving identification number

6.3 Using VEE to control the VNA

1. In the figure below you can see the entire program built with all the components working to control the VNA.

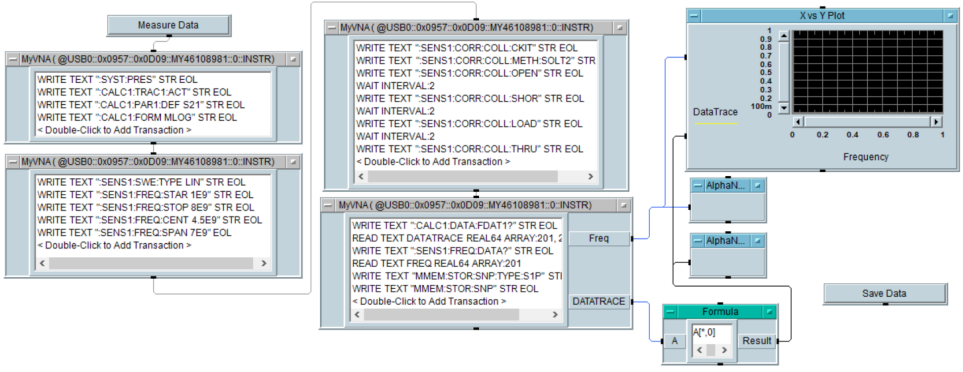


Figure 20: Completed program for controlling VNA

2. I separated the Direct IO block into different categories so that the program may be easier to read and modify. In the first block I have defined Channel 1 to be used and have turned on Trace 1 as well. I have also defined the S parameter to be measured which in this case is S21. If the S parameter needs to be changed simply enter S11, S12, or S22 where the S21 is.

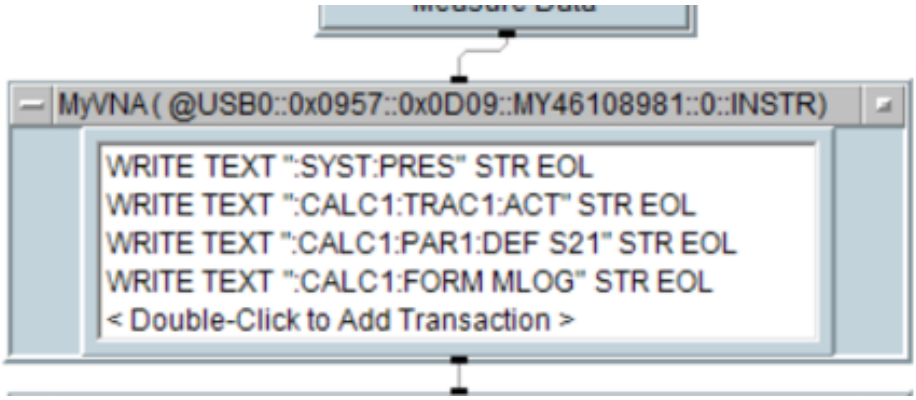


Figure 21: First direct IO block

3. In the next block I have defined everything that will control the frequency in VNA and have specified the frequency sweep to be linear. In order to change the frequency to whatever is desired you can do so by entering the number. 1E9 means 1 GHz so just change the beginning number and last number which is the order of magnitude. The "E" must remain between the numbers.

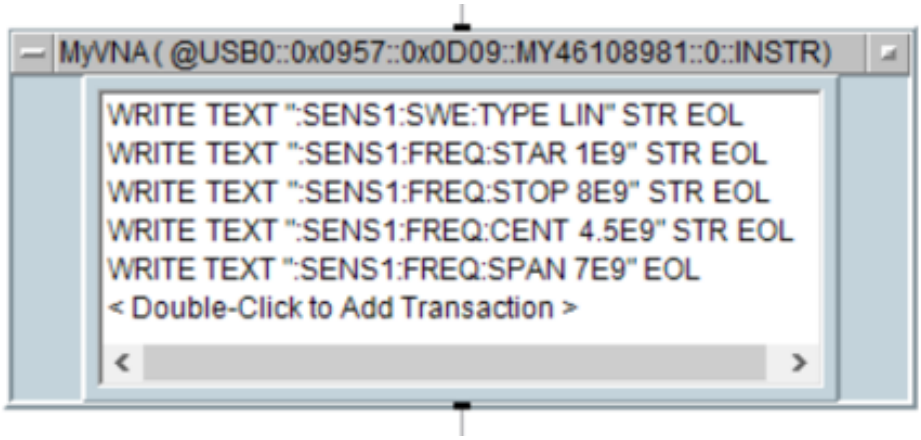


Figure 22: Second direct IO block

4. The next block defines the calibration kit to be used for this measurement and this is the most basic type of calibration for the VNA. A wait time is put between each transaction since the calibration needs to happen sequentially.

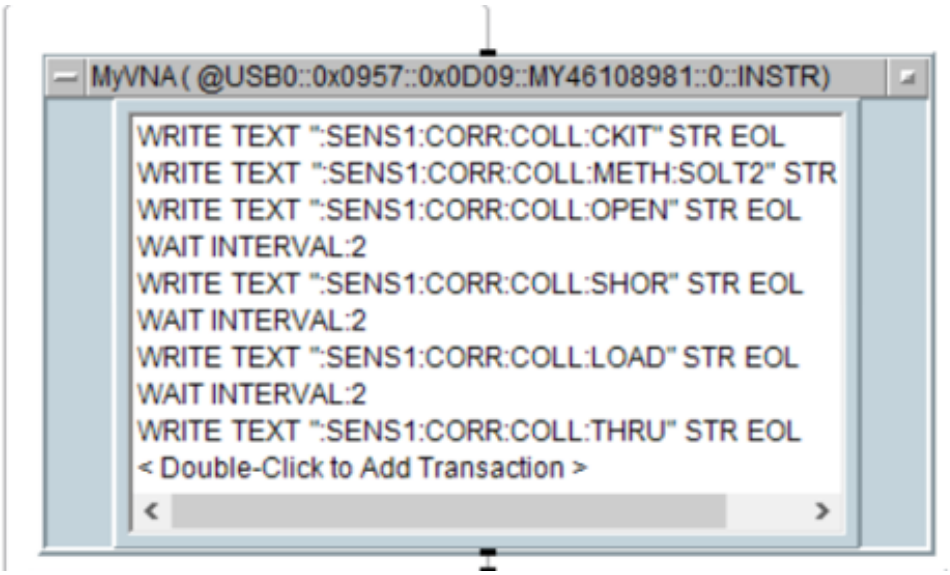


Figure 23: Third direct IO block

5. In this final direct IO block, we write the data of the S parameter and read it so the we can display it in an alpha-numeric display and in the X vs Y plot which is a miniature version of what the VNA displays. We do the same thing for the frequency so that it gets displayed on the alpha-numeric display and in the X vs Y plot. The only thing that does not currently work is the saving of SNP file which is also specified in this block.

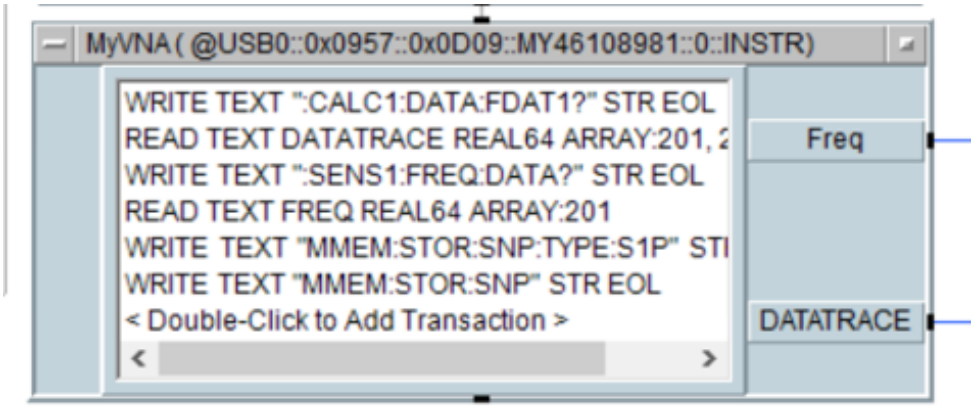


Figure 24: Fourth direct IO block

6. Now that you understand how the program works you can click on the "Measure Data" button to run the program assuming that the VNA is already connected. The program would look like the figure below. I wasn't able to get a clearer picture due to license issues and remote access problems. Other than that, the VNA and VEE program display looked identical with alpha-numeric display working properly for frequency and S21 parameter.

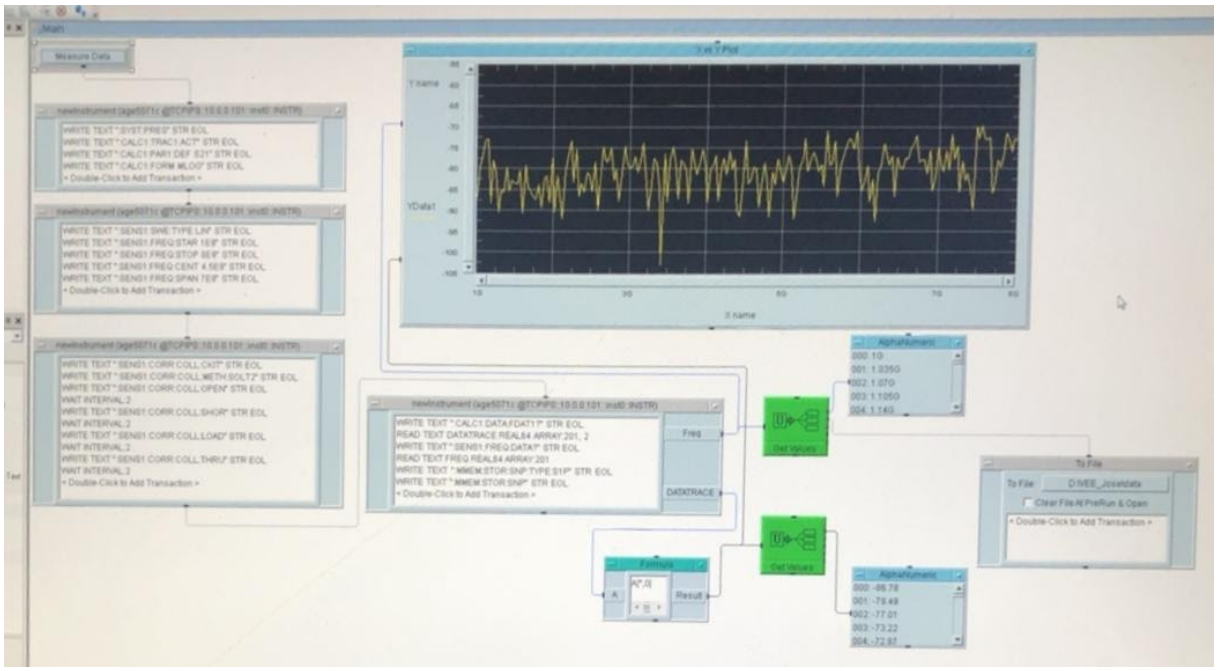


Figure 25: Program in action

6.4 Command Descriptions for VEE

The commands are described more in depth individually on what they do below in the order they are listed in the direct IO blocks. Each line of code was taken straight from the E5071C VNA instruction manual.

Line 1: Presets the VNA
Line 2: Activates channel 1 and trace 1
Line 3: Defines parameter S21 for measurement
Line 4: Defines measurement format as magnitude log
Line 5: Performs a linear sweep
Line 6: Set start frequency
Line 7: Set stop frequency
Line 8: Set center frequency
Line 9: Set span frequency
Line 10: Activate calibration kit
Line 11: Select full 2 port calibration

*There is a wait interval between (line 12-15) each measurement so that the commands don't overwrite each other and happen sequentially

Line 12: Measure open for calibration
Line 13: wait interval 2 seconds
Line 14: Measure short for calibration
Line 15: wait interval 2 seconds
Line 16: Measure load for calibration
Line 17: wait interval 2 seconds

Line 18: Measure thru for calibration
Line 19: wait interval 2 seconds
Line 20: write the formatted data array for the S21 parameter
Line 21: Read the data array of the S21 parameter in Real 64 bit with 201 points
Line 22: write the formatted data array for the frequency
Line 23: Read the data of the frequency array with 201 points
Line 24: determines a file type in touchstone file format and specifies a port
Line 25: saves measurement data in touchstone format
The program currently does not save that data in SNP file

7 Turntable Process Manual

This manual will allow users to control, edit, and troubleshoot the turntable and motor as built for Tangitek. The motor and its Pi controller can be modified for many uses, but the context of this manual is specifically for use by Tangitek's purposes, and as such will be situated within this framework.

7.1 Purpose

Tangitek's RAM material requires a lengthy testing process, and prior capstones have developed a form of automation to ensure more precise and accurate results. Testing requires proper positioning of the MUT at various angles through a full circular range. Furthermore, this testing has to be done within an anechoic chamber to achieve maximum accuracy, and thus any sort of electronic leakage into the testing chamber can disrupt results. As such, the motor and control apparatus (Raspberry Pi) are situated within a metal box within the chamber, and user control is through a computer outside of the chamber connected through fiber optic cable (which minimizes noise leakage).

The Raspberry Pi utilizes C++ and an SDK developed by Robotis to control the motor itself. The VEE software developed by Agilent handles the brunt of collecting data and storing it as appropriate SNP files. As testing and measurements can take a long time, the VEE software is interfaced with the Pi in order to send commands when measurements are done to alert the motor to move to the next position. This whole process is described in Figure 26.

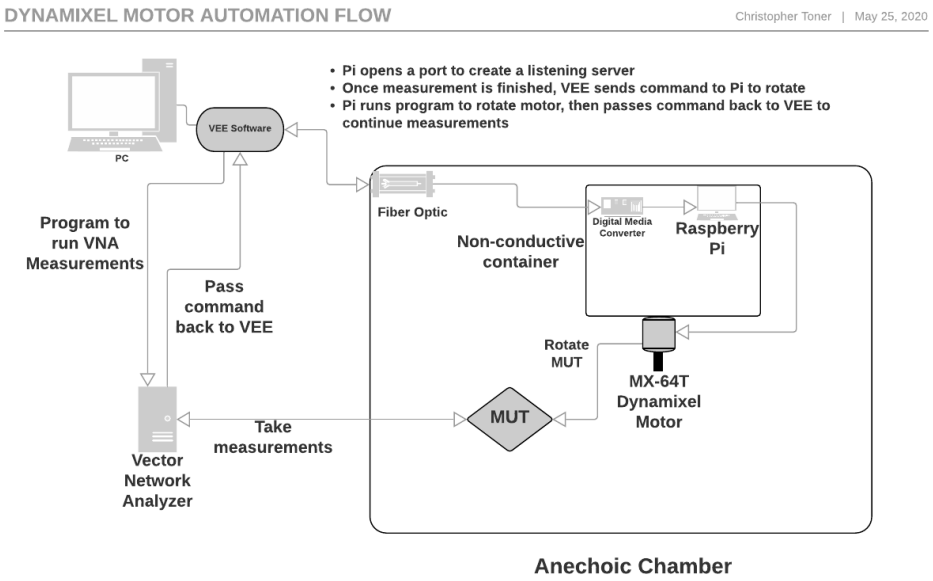


Figure 26: Dynamixel Motor Automation Flow Diagram

7.2 Background Knowledge

7.2.1 VEE Software

VEE is a product distributed by Keysight Technologies. It is a GUI based programming system best utilized for flow control and measurement automation, and specifically can interface easily with Agilent technologies such as a VNA. Unlike text- or terminal-based programming, VEE is object-oriented. As such, there is a bit of a learning curve for traditional programmers, but Keysight's goal is to make the entire process more approachable for engineers with less of a programming background.

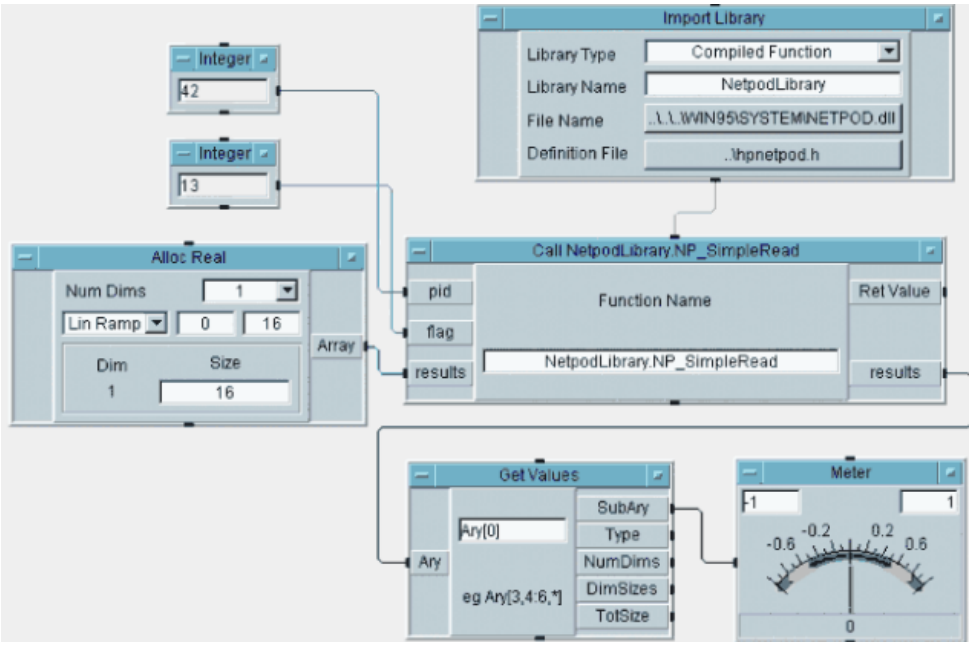


Figure 27: Example VEE UI From Keynes Controls [4]

VEE is run through a series of command-based objects, as seen in Figure 27. VEE offers a bevy of instructions at the users command. For Tangitek purposes, VEE allows the user to specify a frequency sweep to run RCS measurements, and then save the measurements taken by the VNA as an SnP file (amongst other file types, such as CSV, etc.).

VEE also allows the easy interfacing of instruments. While designed for specific Agilent products in mind, such as the Agilent VNA used in this project, VEE can interface with a range of products that are VXI compliant or - as with the scope of this project - can be connected via some bus such as ethernet or USB. The interfacing with these other instruments, though, can be a bit clunky and requires some more user set up to get

[4]The SDK library is rather large. When referring to the build folder, this specifically is referencing the toolbox build folder wherein the example and user-generated programs are kept, location:

```
$ cd ~/dynamixel-workbench/dynamixel_workbench_toolbox/examples/build
```

functioning properly. The VEE User Guide is included in the git repo.

7.2.2 Respberry Pi

Raspberry Pi is a single-board computer (SBC) that functions as the integrating device between the PC/VEE software outside of the anechoic chamber and the Dynamixel motor. It is controlled via ethernet cable and SSH. It sends commands to the motor using C++ executables across a USB-2-Serial proprietary hardware dongle. The C++ programs are developed with an SDK provided by Robotis, the developer of the motor. A separate document, "Raspberry Pi Manual," is provided. This document is separate due to its length and so it can be generalized for setup by the Electronics Prototyping Lab at PSU for projects unrelated to Tangitek. As such, explanation of the SDK programming will be at a minimum in this document and will instead focus on basic knowledge to modify the existing programs to tailor to specific measurement needs.

7.2.3 SSH

Secure Shell (SSH) is a method to securely remote into a server. While a Raspberry Pi is most easily accessed with the KVM method (keyboard-video-mouse), that is not possible with an anechoic chamber setup. SSH was developed to remote into a terminal from a different client, and there exists many GUI-based softwares such as VNC Viewer (recommended by this manual) that allow you to produce the Pi desktop on your client desktop. When connected through ethernet, a Raspberry Pi's address can be finicky and sometimes dynamic. The "Raspberry Pi Manual" has more detailed instructions, including troubleshooting.

7.2.4 Dynamixel Motor

The Dynamixel motor is developed by Robotis, a South korean company, and is considered one of the premier robotics motors in the world. The Dynamixel is not a servo, but more like a stepper motor. It does not use PWM to determine position goal like a servo. It has the ability to give feedback on many high-level features such as current position, velocity, and torque control. The MX-64T has several advantages over the prior capstone models, including:

- Better gear ratio with a resolution of .088°
- Full 360° rotation ability
- Better torque

Of important note: the small resolution of the MX-64T implies that this motor does not turn in exact 1° increments, and is part of the reason why this capstone's deliverables came with a rigid testing plan. The MX-64T also has several modes to increase/decrease the gearbox ratio. Appendix B contains product specs. A separate Excel spreadsheet contains a conversion chart for motor positions to physical degree separations.

7.2.5 Robotics SDK

A Software Development Kit (SDK) is a software package typically developed by a manufacturer that takes care of the bulk of the minutiae of coding and allows a user to quickly implement their own code to suit their purposes. The SDK is usually accompanied with an Application Programming Interface (API) which is essentially a set of coding protocols/procedures that allow for the development of applications for the hardware and/or software.

Robotis has developed an open-source SDK with an API for the Dynamixel that makes interfacing with the motor very fast. The SDK exists as a Github repo and is cloned on the Raspberry Pi (which also exists on the private Tangitek git). The SDK utilizes make and Cmake to build out directories and executables and comes with many built-in examples that make a great starting point for developing new programs.

7.2.6 Cmake/Make

Make and Cmake are ways to automate and organize compilation projects and build out executables and directories. It is highly recommended that the user familiarize themselves with the Make and Cmake protocols to understand how to insert their own programs into the SDK (trust me...you'll feel like a programming expert once you understand how these processes function). See Appendix A for useful links.

7.2.7 Dynamixel Wizard

The Dynamixel Wizard is a proprietary free software developed by Robotis, It is GUI based and allows full manual control of a Dynamixel motor. The Wizard is recommended for debugging purposes, initial setup, and potentially training for users unfamiliar with Linux/C++ languages.

7.2.8 U2D2

The Dynamixel uses serial communication, a method the Pi does not easily handle. It is possible to use the GPIO pins to manually instantiate serial communication: Tangitek Capstone 2019 attempted something similar with an older version of this process using an Arduino. However, this method has its flaws and can have some learning setbacks which is not ideal for a project that is short on learning time. The U2D2 is a USB-to-Serial converter developed by Robotis that was decided as the simplest method to ensure cross-platform communication.

7.2.9 Server Protocol

While VEE software can interface naturally with many systems, a bit more leg work is needed to communicate with the Raspberry Pi. In order for VEE to recognize the Pi, a TCP/IP socket protocol must be used (refer to handling of IP address in "Raspberry Pi Manual" for issues retrieving this). The socket protocol in VEE can send information,

but a listening port on the Pi must be opened to receive the information.

Within the SDK toolbox build folder there is a server.c program (and executable) that creates a basic listening port on the Pi. When run, the port is opened and the Pi listens for commands, and the VEE software can then send commands. As the server program is simplistic, it only accepts integer commands currently. This is an area a future Capstone group could look to improve this process.

VEE uses an integer "1" to command the Pi to run the rotate program and a "2" to inform the Pi that the testing process is complete. The server.c file contains loops to execute the desired program upon receiving the integer command. When completed, the listening port is closed.

7.3 Procedure

To run the turntable with VEE in a standard measurement-1° rotation loop:

1. Ensure VEE has correct Raspberry IP for its socket protocol to write to
2. SSH into Raspberry Pi using login credentials provided elsewhere
3. Use terminal to navigate to toolbox build folder
4. Run server executable
   (a) If error after a recent clean Make, may need to open USB port: `$sudo chmod a+rw /dev/ttyUS`
5. Once terminal confirms it is listening, run VEE program

7.4 Building Out Programs

It is likely the user may want to utilize different programs rather than the 1° rotation loop used for Capstone 2020's RCS measurements. The SDK toolbox contains many example programs for which to build off of. It is recommended the user minimize altering the code outside of the main program body. This code generally is for interfacing purposes and ensures the motor receives commands properly. The main exception is the "baud rate" which can change depending on the model of Dynamixel motor (see Appendix C for table of baud rates).

This brief tutorial will not go through the concepts of building a C++ program and it is recommended the user fully understand what the example program is doing before proceeding with altering a program. After a program is written, the Cmake and Make files need to be properly altered if the user wishes to follow the build out procedure utilized by the SDK. In the directory above the build folder, there is a Cmake.txt file which contains the protocol for Cmake to build out the executable files and directories. It is enough to simply copy+paste+modify the existing code and add the user program where applicable. The same process is fine for the Make file, as long as care is taken to add code for every command pertaining to the new program.

Upon completion, in the terminal while in the directory above the build folder, run

`$ Cmake` to begin the automated process of building out the new directory. Once completed, navigate into the build directory, run `$ Make`, and an executable file will be created for your program. Test program using:

`$ ./[PROGRAM_NAME] /dev/ttyUSB0 57600 1`

/dev/ttyUSB0 : default USB port for U2D2. `$ls -l /dev` to find other USB path

57600 : baud rate for MX-64T, may need to change if different model

1 : Dynamixel ID. Use Dynamixel Wizard or `Model_Scan.cpp` if ID has changed.

7.5 Future Suggestions

We take it upon ourselves to offer some suggestions for future capstones to improve on this design.

- **SDK**
  - More program functionality
  - Move user-based programs to separate directory so SDK source material can maintain link to source repo in case of updates
- **Maintain more complete Git repo**
- **Insert USB/Baud Rate/ID directly into C++ programs**
- **Server program**
  - Have program exist outside of SDK so as not to interfere with source material
  - See if VEE can send strings to make communication more clear
  - Utilize other commands
  - Research to see if VEE can send direct terminal commands to Raspberry Pi (unlikely)
- Possibly able to scrap VEE and automate using Python (see links on PyVISA and GPIB)

8 Appendix A

Useful links for VNA and Turntable automation:

- Excellent tutorial on Cmake (including source for Raspberry Pi version)
- Tutorial on Make
- Dynamixel SDK Github
- Dynamixel SDK Manual
- Dynamixel Workbench Manual
- Dynamixel Wizard Manual
- Dynamixel MX Series Information
- U2D2
- Sockets Tutorial (Basic server program found here)
- Keysight VEE forums
- Forum on PyVISA (possible alternative to sockets control?)
- Talking to Raspberry Pi with GPIB (VEE alternative)
- Talking to VXI11 Machines Using LAN
- More Raspberry Pi Server Setup
- Mostly outdated (for this project) info on using Arduino and Serial
- Forum: Problems with Arduino, Software Serial, and Dynamixel
- Basics on controlling instrumentation from computer
- C++ TutorialsPoint

Appendix B

Specification

Model Name		**MX-64T**
MCU		Cortex-M3 (72 [Mhz], 32 [bit])
Input Voltage	**Min. [V]**	10.0
	Recommended [V]	12.0
	Max. [V]	14.8
Performance Characteristics	**Voltage [V]**	12.0
	Stall Torque [N·m]	6.00
	Stall Current [A]	4.1
	No Load Speed [rpm]	63.0
	No Load Current [A]	0.15
Continuous Operation	**Voltage [V]**	-
	Torque [N·m]	-
	Speed [rpm]	-
	Current [A]	-
Resolution	**Resolution [deg/pulse]**	0.0879
	Step [pulse]	4,096

	Angle [degree]	360
Position Sensor		Contactless absolute encoder (12 [bit], 360 [deg]) Maker : ams(www.ams.com), Part No : AS5045
Operating Temperature	**Min. [°C]**	-5
	Max. [°C]	80
Motor		Coreless (Maxon)
Baud Rate	**Min. [bps]**	8,000
	Max. [bps]	4,500,000
Control Algorithm		PID
Gear Type		Spur
Gear Material		Metal
Case Material		Engineering Plastic
Dimensions (WXHXD) [mm]		40.2 X 61.1 X 41
Dimensions (WXHXD) [inch]		1.58 X 2.41 X 1.61
Weight [g]		126.00
Weight [oz]		4.44
Gear Ratio		200 : 1
Command Signal		Digital Packet

Protocol Type	Half duplex Asynchronous Serial Communication (8bit, 1stop, No Parity)
Link (Physical)	TTL Level Multi Drop Bus
ID	0 ~ 253
Feedback	Position, Temperature, Load, Input Voltage, etc
Protocol version	1.0(Default) 2.0
Operating Mode / Angle	Wheel Mode：Endless turn Joint Mode：360 [deg] Multi-turn Mode：±28 [rev]
Output [W]	-
Standby Current [mA]	100

Appendix C

Reference BPS	Actual BPS	Error(%)
9,600	9,600	0.00
57,600	57,588.4823	-0.02
115,200	115,246.0984	0.04
1,000,000	1,000,000	0.00
2,000,000	2,000,000	0.00
3,000,000	3,000,000	0.00
4,000,000	4,000,000	0.00
4,500,000	4,571,428.571	1.56
6,000,000	6,000,000	0.00