Electronic Countermeasures for Radar

ECE 4007 Senior Design Project

Section L04, MW6
Project Advisor, Dr. Weitnauer

Hunter Scott, Group Leader
Om Kapoor
Thomas Helton

Submitted

May 1, 2013

Table of Contents

Executive Summary

The ECM team developed an electronic countermeasure module to allow hobbyists and educators to demonstrate and innovate methods of defeating tactical radar. The system consisted of a radar and an ECM module. The radar can image a 2D field of view that includes the ECM module. The module can detect that it is being imaged, collect data about the radar and classify it, and then attempt to jam the radar using techniques appropriate for the type of radar identified.

The radar was built using a modified version of a design originally pioneered at MIT. To reduce cost, it outputs data through an audio jack, eliminating the requirement of an external ADC. Instead, this data is captured directly from a PC sound card and analyzed using MATLAB or Octave. The ECM module is designed around a USRP, an inexpensive software defined radio that can be programmed using the open source framework GNU Radio. The frequency of operation is 2.4 GHz, so standard directional Wi-Fi antennas can be used.

The demonstration of this project involved imaging a 2D area with a radar and showing that by activating the ECM module, false targets can be created. Using a simple object tracker, the radar tracked incorrect information about the real target as well as incorrect information about other false targets.

Current countermeasure units cost at least a million dollars, require high level security clearances, and are unavailable to the majority of people. The ECM teams' module has 50% of the functionality of modern countermeasure modules at 0.1% of the cost.

Electronic Countermeasures for Radar

1. Introduction

The Electronic Countermeasures for Radar (ECM) team designed and tested a module that classifies and deceives radar using the principles of digital radio frequency memory (DRFM). The team requested $380 to develop a prototype of the system consisting of both a radar and a countermeasure module, $425 less than originally forecasted.

1.1 Objective

The objective of the ECM team was to design, build, and test an electronic countermeasure (ECM) module that uses the principles of DRFM to modify returning signals to a radar system. A radar capable of operating on different frequencies and incorporating a simple radar tracking algorithm was constructed to test the effectiveness of the ECM module. The team designed the ECM module to identify the target radar's type and frequency to determine the appropriate jamming technique with which to counter. These techniques were based on the concept of sending fake echoes back to the radar which are frequency or phase shifted to cause the radar to incorrectly assess the range and direction of the target.

1.2 Motivation

The motivation driving this project was to design a low cost radar and ECM module aimed at hobbyists and educators. Since radar and radar countermeasures are so closely related, educators often teach both concepts together. In the same way that the software defined radio field has benefitted from the availability of low cost devices [1], it is hoped that a low cost ECM module will spur innovation in a field previously relegated to large laboratories and companies.

1.3 Background

Radar systems are designed to identify a target's range, velocity, elevation (or altitude), and angle (or azimuth). Many implementations of radar systems are used in defensive systems for early warning, ground control intercept, airborne intercept, acquisition, and target tracking [2]. Jamming techniques allow a user to avoid radar detection and the implementations include: chaff (metallic strips for causing false signal returns), spot jamming (jamming power focused on a single frequency), stealth material, sweep jamming, pulse jamming, and DRFM jamming [3].

RADAR is defined as RAdio Detection And Ranging. Radar operates at various radio frequencies and their classifications along with their radar wavelengths are shown in Figure 1.

Band	Frequency Range	Radar Frequency	Radar Wavelength
VHF	30–300 MHz	220 MHz	1.36 m
UHF	300–1,000 MHz	425 MHz	0.71 m
L	1–2 GHz	1.3 GHz	23 cm
S	2–4 GHz	3.3 GHz	9.1 cm
C	4–8 GHz	5.5 GHz	5.5 cm
X	8–12 GHz	9.5 GHz	3.2 cm
K_u	12–18 GHz	15 GHz	2.0 cm
K	18–27 GHz	24 GHz	1.3 cm
K_a	27–40 GHz	35 GHz	0.86 cm

Figure 1. Radar frequency ranges and wavelengths. [1]

The ECM group's radar operates in the S band. The idea behind radar is to send a signal to a target and listen for the echo (or reflection) from the target. Figure 2 below shows what a typical pulsed radar waveform might look like and what the reflections off the target look like.

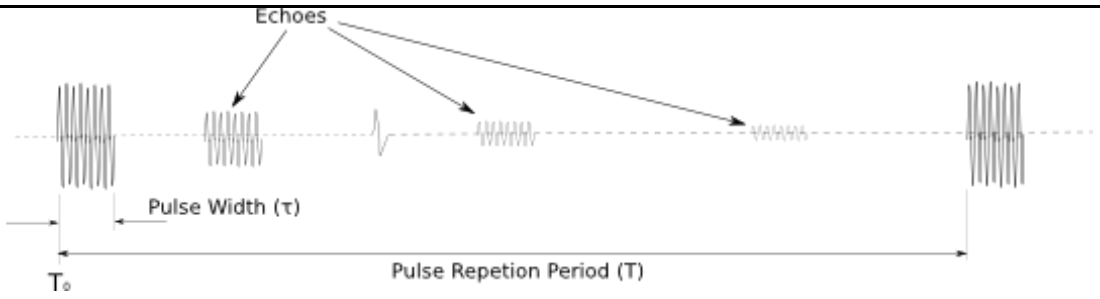


Figure 2. Typical pulsed radar waveform with reflections from target.

To calculate the wavelength (λ), the speed of light (3 x 10^8 meters/second) will be denoted by "c" and the radar frequency "f" to gain the equation λ = c / f . The basic radar equation can be seen in Figure 3, and the max range equation can be seen in Figure 4 [4].

$$R = \frac{P_p \tau G_t \sigma A_r}{(4\pi)^2 \, D^4 \, k \, T_s \, L}$$

P_p = transmitter power
τ = waveform duration
G_t = transmit antenna gain
σ = radar cross section
A_r = antenna aperture area
R = is radar range
K = Boltzmann's constant, 1.38 x 10^{-23} Joules/Kelvin
T_s = system noise temperature
L = radar system losses
D = distance from radar

Figure 3. Basic radar range equation.

$$R_{MAX} = \frac{\left[P_T G \sigma A_e\right]^{1/4}}{\left[(4\pi)^2 S_{MIN}\right]}$$

P_T = transmitted power
G = antenna gain
σ = RCS
A_e = antenna aperture area
R = range to the target

Figure 4. Max radar range equation.

The principles behind DRFM take advantage of the assumptions a radar system makes when identifying a target. The first assumption a radar makes is that the only possible reasons for

a shift in frequency, phase, amplitude, or polarization is due to a reflection off of a material with a very different permittivity than air [2]. DRFM defeats this assumption by artificially shifting any or all of those physical wave properties and sending them back to the radar [5]. This causes the radar to make incorrect deductions about the material, velocity, and range of what it thinks are reflections off of a surface. Another assumption that radar makes is that electronic countermeasures used against it will not be able to keep up with the sliding time slot that it samples in and that the countermeasure won't be able to find the radar's frequency precisely enough. This was true before 1999, when DRFM systems first came into use [6]. With the proliferation of high dynamic range ADCs, fast FPGAs, and more efficient processors, it is now possible to very quickly and effectively find a radar's frequency and keep up with its sampling rate [7].

2. Project Description and Goals

The ECM team constructed a radar and ECM module to demonstrate different techniques of electronically countering radar signals. The ECM module has an interactive menu to select the type of jamming technique to deploy. Users can program and edit how the radio in the ECM module will characterize signals using Python scripts. The design project required $380 to build the radar and ECM module that provides the following features:

- Interactive graphical user interface (GUI)
- Ability to detect and characterize radars
- Capability of deploying different countermeasure methods for jamming
- Costs less than $1400 to produce

3. Technical Specifications

The two major components of the system are the radar and the ECM module. Table 1 displays the performance specifications for the radar and Table 2 displays the specifications for the ECM module. The radar prototype consists of a breadboard attached to two antennas and a computer system. Table 1 shows the effective range of the radar to be 70 m, a figure attain through testing. As shown in Table 2, an output power of 100 mW ensures that the ECM module can transmit pulses back to the radar at distances of up to and greater than the effective range of the radar. The ECM module can also generate arbitrary waveforms, so it is possible to counter a wide range of radar waveforms.

Table 1. Radar Specifications

Feature	**Specification**
Dimensions	12x12x1 inches
Weight	2 pounds
Transmit Power	60 mW
Operating Frequency Range	2.4 GHz
Operation Distance	Up to 70 m
Output Waveforms	Pulse train, FM chirp
Software	MATLAB and Python
Polarization	Vertical or Left Hand Circular

Table 2. ECM Module Specifications

Feature	Specification
Dimensions	12x8x2 inches
Weight	3 pounds
Transmit Power	100 mW
Operating Frequency Range	400-4400 MHz
Operation Distance	Depends on radar sensitivity, 140 m average
Output Waveforms	Arbitrary waveform generation
Software	GNU Radio and Python
Polarization	Vertical or Left Hand Circular

4. Design Approach and Details

4.1 Design Approach

System Overview

The system demonstrated consists of two parts: a radar and an ECM module. The radar is used to collect range and velocity information from targets moving in front of it. It is used to observe the effectiveness of the ECM module. The ECM module causes the radar to display incorrect information about the targets it is illuminating by employing a variety of tactics. It has the capability to transmit and receive in the radar's frequency range and perform signal analysis to characterize the type of radar that it is encountering. Once it learns about the radar, it can switch modes and select the appropriate countermeasure and begin to jam. Figure 5 below shows the relationship between the radar and ECM module.

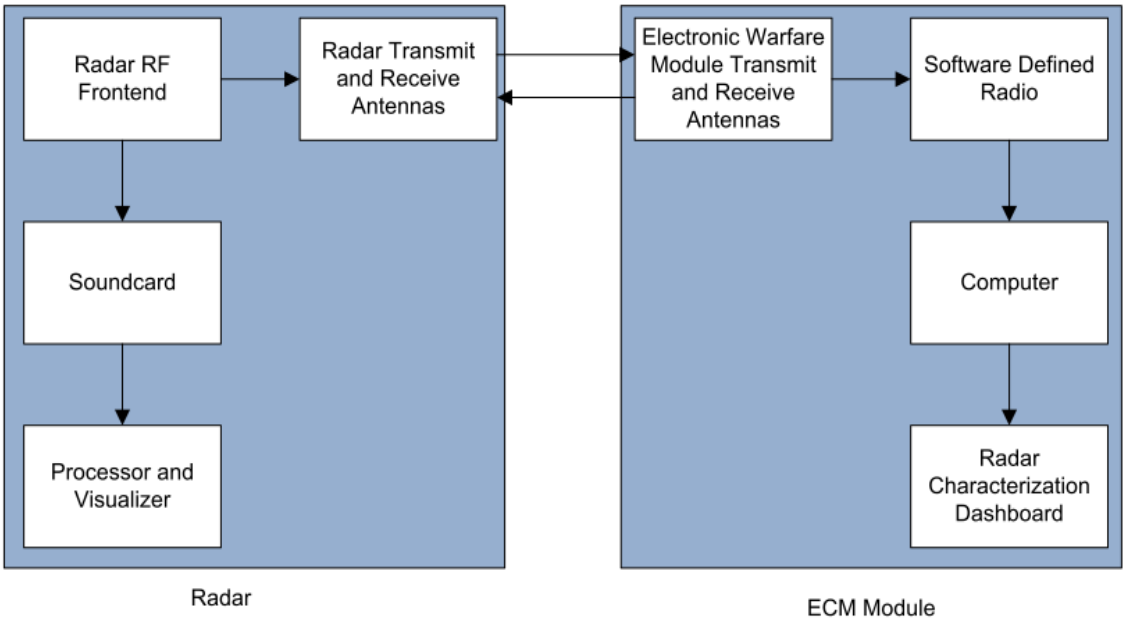


Figure 5. Block diagram of the radar and ECM module layouts.

Radar Subsystem

The radar subsystem consists of a standard radar platform originally developed at MIT. The radar system diagram can be seen in Figure 6 below.

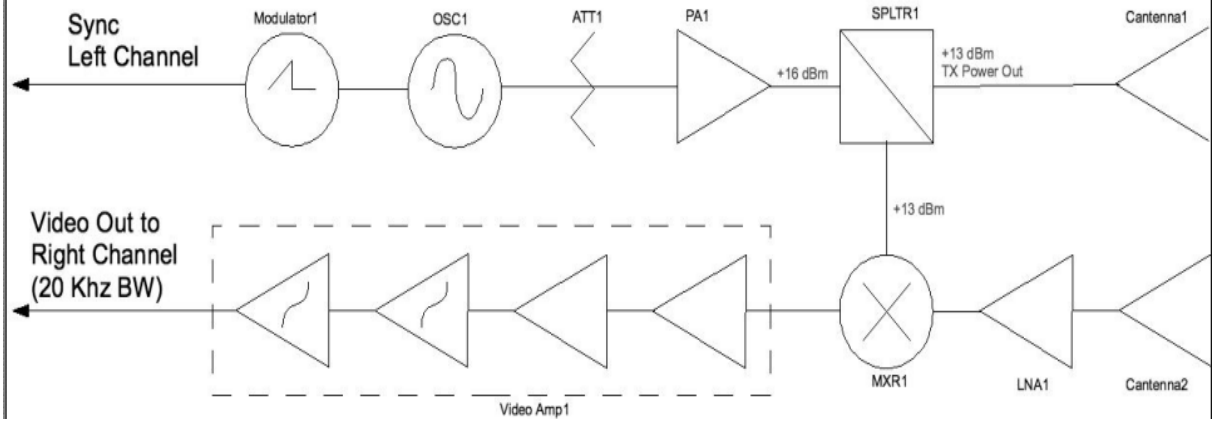


Figure 6. Block diagram of radar system components.

A voltage-controlled oscillator (VCO) generates a linear frequency modulated continuous wave at 2.4 GHz which is sent to the transmit antenna. The receive side amplifies the signal it observes

on the receive antenna and mixes it down to baseband using the VCO as the local oscillator. This signal is then adjusted to have a 20 KHz bandwidth and a correct amplitude to be suitable for capture from a PC stereo sound card. The right channel is used to input the receive data and the left channel is used to sync the VCO. The board has test points at the VCO output, the right channel output, and the 12V supply line. The VCO output and right channel output test points were probed using a spectrum analyzer to confirm correct functionality. The 12V supply line was probed with an oscilloscope to ensure the noise from the power supply was within acceptable limits. Once successful, the radar was tested using cables within inline attenuators on the TX and RX SMA connectors using a spectrum analyzer and a signal generator. Once it was determined that the radar could both transmit and receive signals correctly in the lab, directional antennas were attached and the radar was taken outside to image people walking around. The data generated in this test was sent to MATLAB to visualize range vs time. The Cartesian-temporal plot can be seen in Figure 7 below.

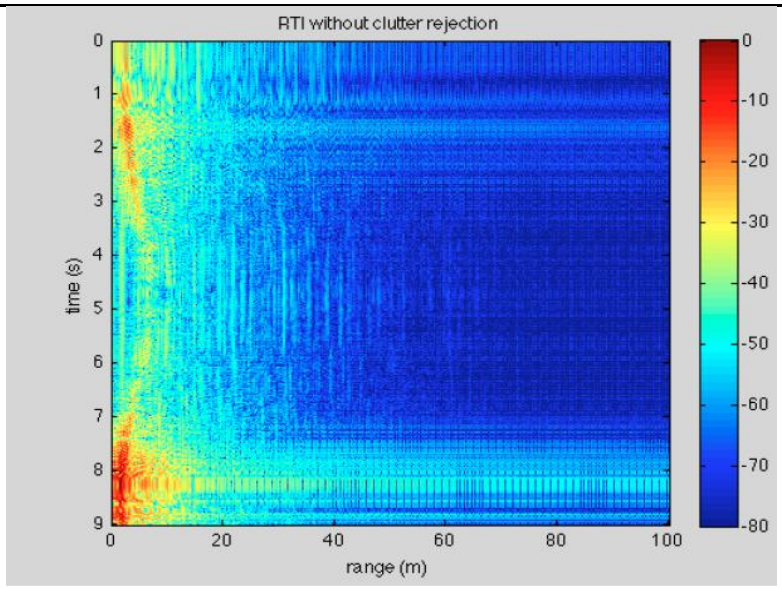


Figure 7. Cartesian-temporal plot of time vs velocity.

The data is also visible in real-time showing range and return signal strength, which is denoted with false color.

Signal Characterization

The ECM module receives incident radar waves and must determine the type of radar it is observing. This feature was implemented but never tested. It first attempts to detect certain polarizations of the wave using antenna diversity by comparing the RSSI of a vertically polarized antenna and a left hand circularly polarized antenna, both with the same gain. Table 3 shows the polarization loss experienced if the radar is transmitting with a different polarization than the ECM module is receiving on.

Table 3. Polarization Loss

Transmit Antenna Polarization	**Receive Antenna Polarization**	**Percent Lost**
Vertical	Vertical	0
Vertical	Slant (45 or 135 degrees)	50
Vertical or Horizontal	Horizontal or Vertical	100
Vertical	Circular (right or left-hand)	50
Horizontal	Horizontal	0
Horizontal	Slant (45 or 135 degrees)	50
Horizontal	Circular (right or left-hand)	50
Circular (left-handed)	Circular (left-handed)	0

Once polarization is determined, the ECM module uses the appropriate antenna to sample the wave and look for a continuous wave or a pulsed wave. Next, it looks for deliberate modulation.

It detects both of these features by decomposing the frequency components using Fourier analysis and determines the time/frequency relationship of pulses to one another using wavelet analysis. If modulation is detected, the ECM module demodulates the samples and looks for pulse compression. Once all of this critical information about the radar is acquired, the ECM module attempts to classify the type of radar it is encountering. If it has seen the radar before, it can attempt to jam the radar using known weaknesses in the implementation. Otherwise, if it has not seen the radar before, it will continue to collect information about the radar for later analysis and will alert the user to the situation. It does not attempt to jam the radar in this instance, in case target-on-jam systems are in use. The signal characterizer in the ECM module is written in Python using GNU Radio. Fourier and wavelet analysis is performed within GNU Radio to characterize the radar signature. The characterization component of this project was never tested, however preliminary code can be found in Appendix D.

Jammer Subsystem

Once the target radar is known, this subsystem actually performs the jamming. Table 4 shows the type of radar detected and the action performed by the jammer subsystem.

Table 4. Jammer Response to Radar Type

Type of Radar	**Jammer Response**
Pulsed	Identify range gate, transmit pulses back
Pulse-Doppler	Identify range gate, transmit frequency shifted pulses back
Pulse compression	Identify range gate, transmit frequency shifted chirps back
Other	Collect data, do not attempt to jam.

The jammer subsystem of the ECM consists of a universal software radio peripheral (USRP), antennas, and a computer. It is controlled using GNU Radio, which is a framework that allows the user to write Python scripts to define how the USRP transmits and receives. It also allows the user to perform signal analysis. Figure 9 below shows the architecture of the USRP.

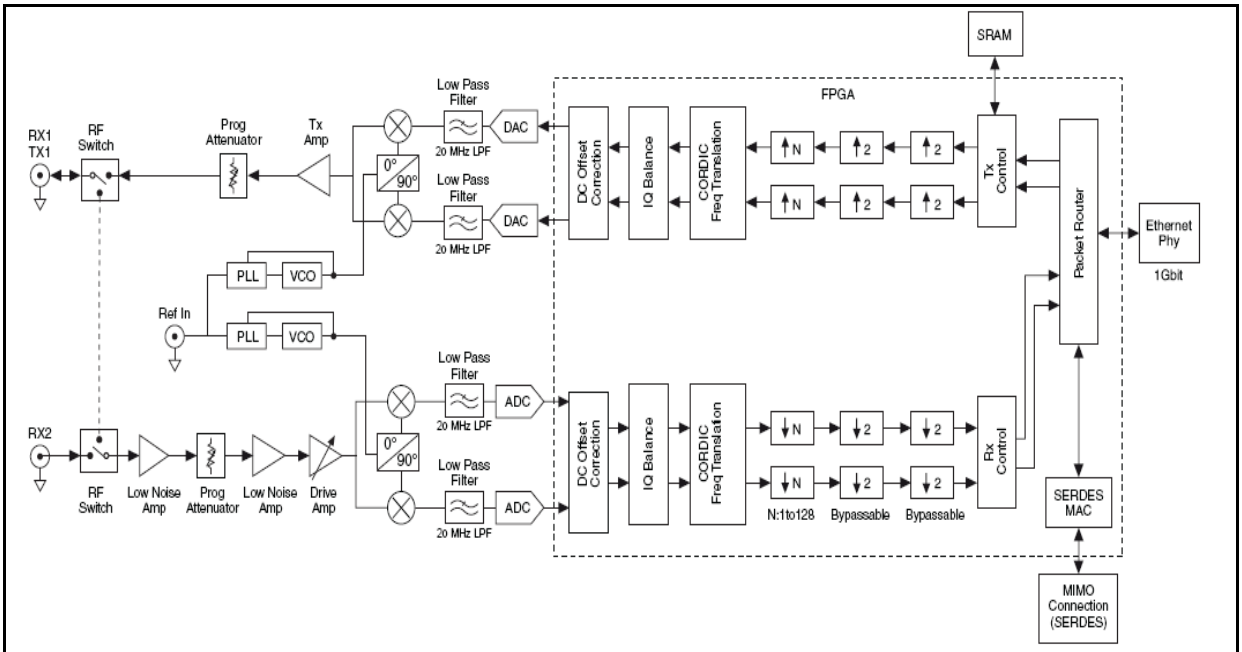


Figure 9. Block diagram of the USRP components.

Because the radar and ECM module both operate in the 2.4 GHz band, care must be taken to ensure that Wi-Fi doesn't interfere with the system and that the system doesn't interfere with Wi-Fi. The demonstration took place outdoors if possible to limit the effects of Wi-Fi, however we were able to successfully test some features inside. The frequency of both systems was shifted to an open Wi-Fi channel in the ISM band covering 2.4-2.5 GHz. The use of directional antennas on both the radar and ECM module further limited potential interference problems. Because the USRP daughterboard bandwidth was only 40 MHz, the radar was modified to sweep over 40 MHz by reducing the amplitude of the signal going into the VCO to 25% of the original amplitude.

Dashboard

To command and control the ECM module, a simple GUI was developed to let the user visualize the signal that is being analyzed as well as observe the status of the ECM and execute other countermeasure options. A screenshot of the GUI can be seen below in Figure 10.

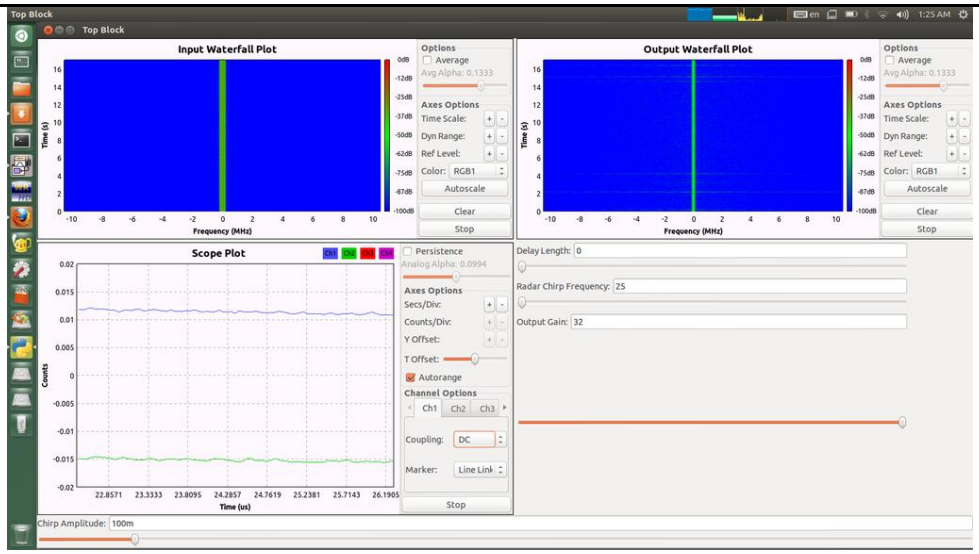


Figure 10. GUI for the ECM module.

5. Schedule Tasks and Milestones

The ECM team tested, researched, and implemented the proposed prototype and developed it over 3 months. The table in Appendix A displays all the major milestones listing their difficulty level and persons assigned to the task. Appendix B shows a Gantt chart of all major milestones. Appendix C shows a detailed Gantt Chart with a timeline of all the tasks that were performed, their difficulty and the persons performing these tasks.

6. Results and Acceptance Testing

The demonstration consisted of a portable, mountable radar connected to a computer and an ECM module connected to a computer. The radar and ECM modules were placed 10 meters

from each other. The radar was activated and started transmitting. Normal operation was demonstrated by having volunteers walk back and forth in front of the radar while the real time processor that was written displayed the correct plot. Then, the ECM sent modified return pulses back to the radar, causing the radar to report incorrect target information. The demonstration of the project took place in an open space, and the poster session demo took place at the senior design expo outdoors near the McCamish Pavillion. The results of the demonstration were viewable on the computer screens via a custom designed GUI.

As per our project specifications, we were able to demonstrate all three types of Radar jamming, namely:

1. Pure Noise Jamming

Pure noise jamming was accomplished by broadcasting a 2.4 GHz Gaussian noise signal at a high power. This code can be found in Appendix D. Noise saturates any sensitivity of the radar and makes it impossible to observe the reflections off of the real target occurring at much lower amplitudes. The disadvantage of this technique is that is requires a large amplifier to work well and disturbs other users of the spectrum. It can also be dangerous in a military scenario if the enemy is using a target-on-jam system that tracks large signal sources. Figure 11 shows the capture of the radar without any type of jamming taking place. Figure 12 shows the plot obtained when the jammer was activated.

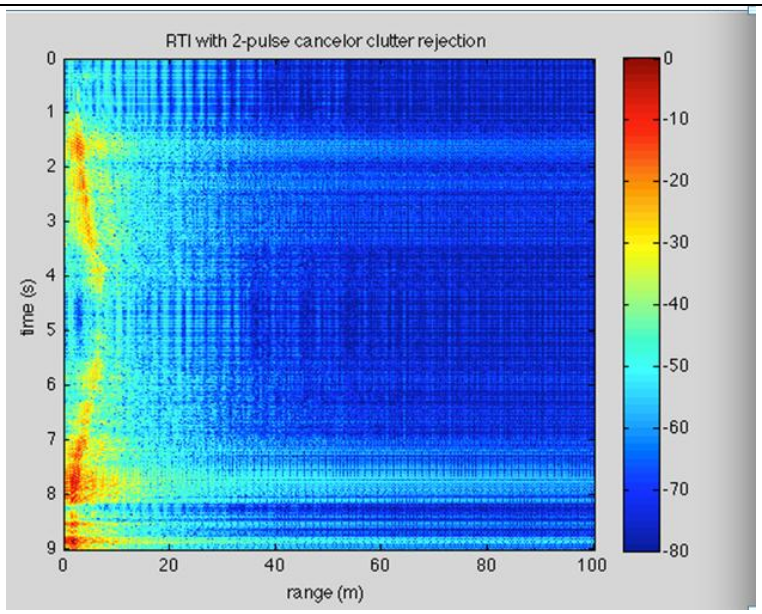


Figure 11. Original radar plot.

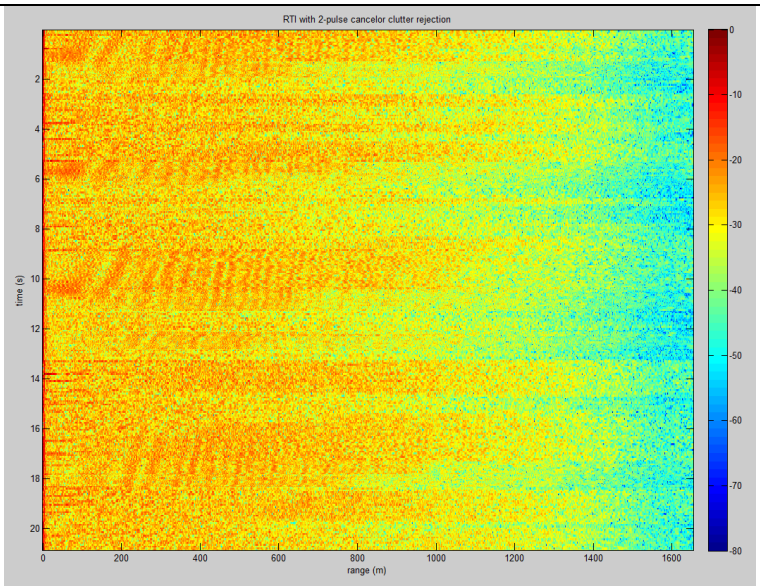


Figure 12. Result of noise jamming.

2. DRFM Jamming

DRFM jamming was the main focus of this project and was accomplished by several means. The first version synthesized a frequency modulated continuous wave onboard the USRP

and broadcasted it towards the radar. However, performance was low and it was difficult to pick out the fake reflection. The second version involved recording radar reflections directly to memory by connecting the USRP via cable to one of the radar cantennas through an amplifier and then activating the radar. Because the raw data is guaranteed to be the same thing that the radar sees, broadcasting back the recording worked much better. The second version of the code is available in Appendix D. Figure 13 is the capture by the Radar upon activating the USRP to produce DRFM Jamming.

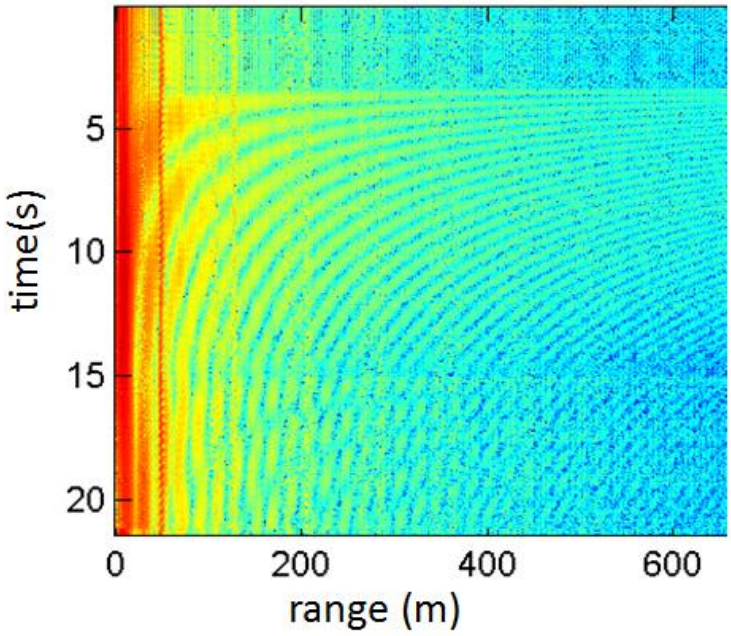


Figure 13. Result of DRFM Jamming.

3. Clutter Rejection Jamming

Clutter rejection algorithms are normally designed to filter out reflections from objects that the user is not interested in. The clutter rejection algorithm that was implemented for the MIT radar filters out objects that are not moving and does magnitude normalization. This clutter rejection jamming technique, which causes the clutter rejection algorithm to actually add noise to the plot, was discovered by accident during experimentation. Further investigation is required to

fully understand what is happening, but we believe that we are exploiting a poor choice of filter bandwidth in the clutter rejection algorithm. Figure 14 shows the radar capture with clutter rejection jamming before the clutter rejection algorithm on the PC has been run. Figure 15 shows the radar capture with clutter rejection jamming after the clutter rejection algorithm on the PC has been run. Note that the apparent noise has increased.

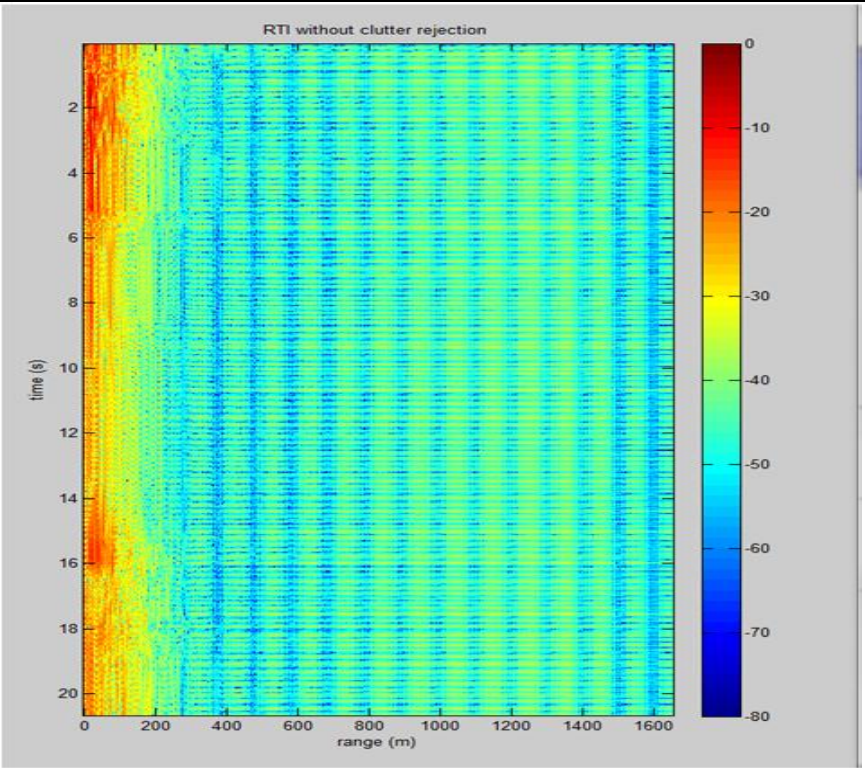


Figure 14. Radar capture during clutter rejection jamming before the clutter rejection algorithm is applied.

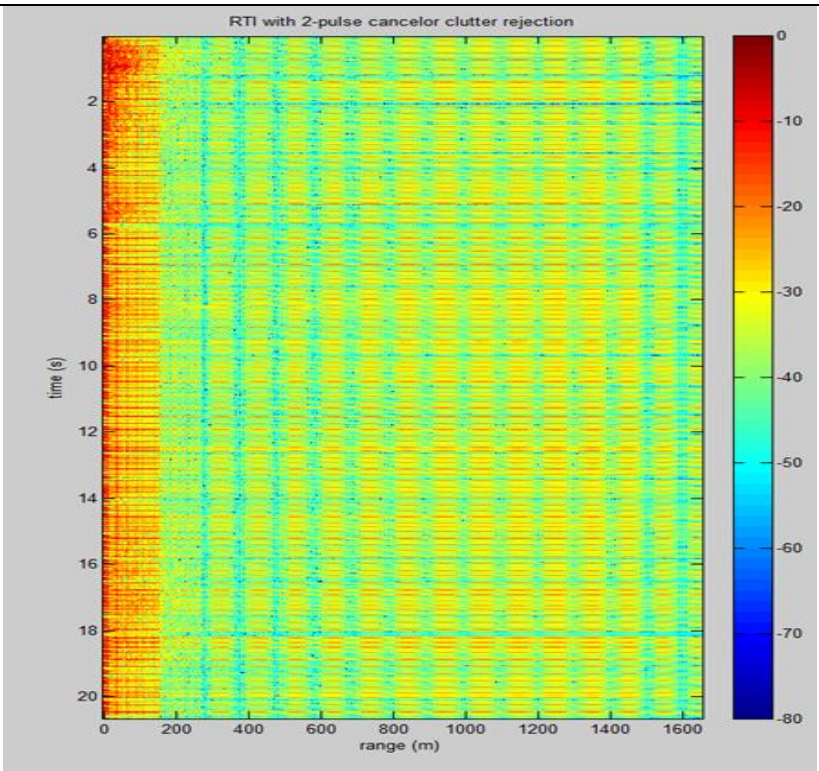


Figure 15. Radar capture during clutter rejection jamming after the clutter rejection algorithm is applied.

7. Cost Analysis

The budget for the Senior Design project was $360. The main cost associated with the production of the ECM module was the cost of the USRP. That cost was waived as a team member had the required part. The cost of the antenna was reduced because it was possible to use antennas built from coffee cans which work well for this application and cost $6. The cost of the software development required for processing and modifying incoming radio waves is negligible because it is either open source or the ECM team wrote it. Table 5 shows the breakdown of costs for parts required for the prototype.

Table 5. Component costs for prototype

Part Description	Quantity	Unit Price($)	Total Price($)
USRP B100	1	$640	$640
Antennas	4	$6	$24
USRP SBX Daughterboard	1	$475	$475
Cantenna RADAR	1	$300	$300

Three engineers were required to complete the design and development of the ECM module. A breakdown of the number of hours that were put in by each engineer during development is shown in Table 6.

Table 6. Development hours per engineer

Task	Hours
Class	20
Weekly Meetings	10
Research	20
Presentation	1
Assembly	5
Testing	5
Software Development	10
Total	71

Labor costs are calculated based on the above table and assuming an average salary of $27 per hour for each engineer. Considering the number of hours required for development of the project, labor costs amount to $5751. Assuming fringe benefits of 30% of labor and an overhead of 120% of labor and material, the total development cost for the ECM module is shown in Table 7 below.

Table 7. Total Development Costs

Component	Cost
Labor	$5751.00
Parts	$1439.00
Fringe Benefits	$1725.30
Overhead	$8479.20
Total Development cost	$17270.50

The production run will consist of an approximation of five units being sold per week over a period of four years. This amounts to 1080 units being sold over a span of four years. There will be a discount of 5% being offered to any educational organization seeking to procure more than 50 units of the product. The selling price will be marked up by 6.71% when compared to the cost price and will be sold at a price of $3200 after taking into account fringe benefits and overhead. This amounts to a profit of $216.50 per unit sold and a profit of $56.50 per unit when sold in bulk. Assuming that all units were bought individually, this amounts to a total profit of $233,820 in a span of 4 years. The marked price is substantially lower than any ECM module available in the market and the first of its kind with respect to domestic use. Assuming that our ECM module is in production, the time taken to assemble and test the product amounts to 1.5 hours, Table 8 shows a breakdown of profit per unit.

Table 8. Breakdown of profit per unit

Expense or Income Component per Unit	Cost per Unit
Parts	$1439
Assembly Labor	$14
Test Labor	$27
Fringe Benefits	$12.3
Subtotal	$1356.3
Overhead	$1627.2
Total Input Cost	$2983.5
Selling Price	$3200.00
Profit	$216.50

8. Conclusions and Future Work

The ECM team was able to build and demonstrate a radar and an accompanying electronic countermeasure module. It will allow hobbyists and educators to demonstrate and innovate methods of defeating tactical radar. Some features were not tested, and offer an area for future work. A circularly polarized antenna was designed and simulated, but never fabricated or tested. This can be found in Appendix E. Future improvements could include more sophisticated ECM techniques and better radar characterization. Because the system was designed to be open source and flexible, it lends itself extremely well to being modified, and indeed is intended to be modified by the end user. It is this flexibility that will help speed and enhance the learning process in educating the next generation of radar engineers.

9. References

[1] Katz, S.; Flynn, J.; , "Using software defined radio (SDR) to demonstrate concepts in communications and signal processing courses," *Frontiers in Education Conference, 2009. FIE '09. 39th IEEE* , vol., no., pp.1-6, 18-21 Oct. 2009

[2] Penley, Bill, *Early Radar History – an Introduction*". Cambridge, MA: MIT Press, 2002.

[3] Navy, "*Introduction to Naval Weapons Engineering"*. Washington, DC: Office of Headquarters Operations, 2005.

[4] G. R. Curry, *Pocket Radar Guide – Key Radar Facts, Equations, and Data*. Raleigh, NC: SciTech Publishing, Inc., 2010. Available online: http://www.knovel.com/web/portal/browse/display?_EXT_KNOVEL_DISPLAY_bookid=4419

[5] Sun Guoying; Li Yunjie; Gao Meiguo; , "An Improved DRFM System Based on Digital Channelized Receiver," *Image and Signal Processing, 2009. CISP '09. 2nd International Congress on* , vol., no., pp.1-5, 17-19 Oct. 2009

[6] Watson, Charles, "A Comparison of DDS and DRFM Techniques in the Generation of "Smart Noise" Jamming Waveforms", M.S. thesis, Naval Postgraduate School, Monterey, CA, 1996.

[7] SPEC, ADEP T4000. July 24, 2012. [Online]. Available: http://www.spec.com/content/view/91/1/. [Accessed: Jan. 22, 2013].

[8] Small Business Innovation Research, Navy Direct Digital Radio Frequency Conversion Digital Radio Frequency Memory. Jan. 2, 2013. [Online]. Available: http://www.dodsbir.net/sitis/display_topic.asp?Bookmark=43277. [Accessed: Jan. 22, 2013].

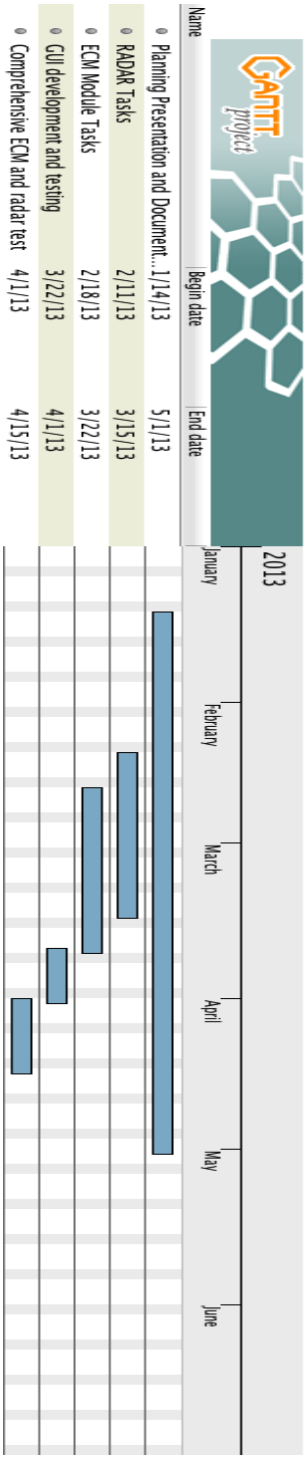
[9] Dr. Gregory Charvat's Personal Website. Nov. 6, 2011. [Online]. Available: http://www.glcharvat.com/Dr._Gregory_L._Charvat_Projects/Cantenna_Radar.html. [Accessed: Feb. 6, 2013].

[10] Ettus Research, "About Ettus Research." [Online]. Available: http://ettus.com/site/about. [Accessed: Feb. 6, 2013].

Appendix A

Task Name	Task Lead	Risk Level
Planning, Presentation and Documentation	All	Low
TRP	All	Low
Project Proposal	All	Low
Parts Ordering	HS	Low
PDR Presentation	All	Low
Final Project Presentation	All	Low
Final Project Documentation	All	Medium
Final Project Report	All	Medium
RADAR tasks	All	Medium
Testing Receiver and Transmitter	HS, OK	Medium
Testing Software Processing Capabilities	HS, TH	High
ECM Module tasks	All	Medium
Signal attenuation and characterization software	OK, TH	High
Testing Receiver and Transmitter	HS, TH	Medium
GUI development and testing	All	Medium
Comprehensive ECM and radar test	All	Medium

Appendix B


GANTT project
Name
Begin date
End date
2013
January
February
March
April
May
June
Planning Presentation and Document... 1/14/13 5/1/13
RADAR Tasks 2/11/13 3/15/13
ECM Module Tasks 2/18/13 3/22/13
GUI development and testing 3/22/13 4/1/13
Comprehensive ECM and radar test 4/1/13 4/15/13

Appendix C

GANTT project
2013
Name
Accountable
Week 3 1/13/13
Week 4 1/20/13
Week 5 1/27/13
Week 6 2/3/13
Week 7 2/10/13
Week 8 2/17/13
Week 9 2/24/13
Week 10 3/3/13
Week 11 3/10/13
Week 12 3/17/13
Week 13 3/24/13
Week 14 3/31/13
Week 15 4/7/13
Planning, Researching and Documenting All
Weekly Progress Report HS
TRP All
Radars TH
Jamming Techniques HS
Cantenna Radar OK
Project Proposal All
Introduction, Proj Description, Tech Specs TH
Design Approach and Details HS
Demonstration, Schedule, Cost Analysis OK
Ordering Parts HS
RADAR All
Build HS,OK
Test Sync Channel OK,TH
Test Video out to right channel OK, TH
Test Transmitting Antenna HS, TH
Test Recieving Antenna HS, TH
Analysing Recieved Data OK, HS
Software Analysis for various radar types OK, HS
ECM All
Receiving Antenna Test OK
Transmitting Antenna Test HS
Signal Charachterization Software All
Signal Attenuation Software Test All
Pulsed OK, HS
Pulsed Doppler TH, HS
Pulse Compression TH, HS
Random OK, TH
Transmit test for modulated waveform TH
GUI Development and Testing All
Comprehensive ECM and Radar Test All

Appendix D - Code

NoiseJammer.py

```
#!/usr/bin/env python

from gnuradio import analog
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio import uhd
from gnuradio.eng_option import eng_option
from gnuradio.gr import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class top_block(grc_wxgui.top_block_gui):

	def __init__(self):
		grc_wxgui.top_block_gui.__init__(self, title="Top Block")
		_icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
		self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

		##################################################
		# Variables
		##################################################
		self.variable_slider_2 = variable_slider_2 = .1
		self.variable_slider_1 = variable_slider_1 = 32
		self.variable_slider_0_0 = variable_slider_0_0 = 25
		self.variable_slider_0 = variable_slider_0 = 0
		self.test_freq = test_freq = variable_slider_0_0
		self.samp_rate = samp_rate = 21e6
		self.gain = gain = variable_slider_1
		self.delay_length = delay_length = variable_slider_0
		self.amplitude = amplitude = variable_slider_2

		##################################################
		# Blocks
		##################################################
		_variable_slider_0_0_sizer = wx.BoxSizer(wx.VERTICAL)
		self._variable_slider_0_0_text_box = forms.text_box(
			parent=self.GetWin(),
			sizer=_variable_slider_0_0_sizer,
			value=self.variable_slider_0_0,
			callback=self.set_variable_slider_0_0,
```

```
        label="Radar Chirp Frequency",
        converter=forms.float_converter(),
        proportion=0,
)
self._variable_slider_0_0_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_variable_slider_0_0_sizer,
        value=self.variable_slider_0_0,
        callback=self.set_variable_slider_0_0,
        minimum=25,
        maximum=100,
        num_steps=100,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
)
self.GridAdd(_variable_slider_0_0_sizer, 11, 10, 1, 9)
_variable_slider_2_sizer = wx.BoxSizer(wx.VERTICAL)
self._variable_slider_2_text_box = forms.text_box(
        parent=self.GetWin(),
        sizer=_variable_slider_2_sizer,
        value=self.variable_slider_2,
        callback=self.set_variable_slider_2,
        label="Chirp Amplitude",
        converter=forms.float_converter(),
        proportion=0,
)
self._variable_slider_2_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_variable_slider_2_sizer,
        value=self.variable_slider_2,
        callback=self.set_variable_slider_2,
        minimum=0,
        maximum=1,
        num_steps=10,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
)
self.Add(_variable_slider_2_sizer)
_variable_slider_1_sizer = wx.BoxSizer(wx.VERTICAL)
self._variable_slider_1_text_box = forms.text_box(
        parent=self.GetWin(),
        sizer=_variable_slider_1_sizer,
```

```
            value=self.variable_slider_1,
            callback=self.set_variable_slider_1,
            label="Output Gain",
            converter=forms.float_converter(),
            proportion=0,
    )
    self._variable_slider_1_slider = forms.slider(
            parent=self.GetWin(),
            sizer=_variable_slider_1_sizer,
            value=self.variable_slider_1,
            callback=self.set_variable_slider_1,
            minimum=0,
            maximum=32,
            num_steps=31,
            style=wx.SL_HORIZONTAL,
            cast=float,
            proportion=1,
    )
    self.GridAdd(_variable_slider_1_sizer, 12, 10, 1, 9)
    _variable_slider_0_sizer = wx.BoxSizer(wx.VERTICAL)
    self._variable_slider_0_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_variable_slider_0_sizer,
            value=self.variable_slider_0,
            callback=self.set_variable_slider_0,
            label="Delay Length",
            converter=forms.int_converter(),
            proportion=0,
    )
    self._variable_slider_0_slider = forms.slider(
            parent=self.GetWin(),
            sizer=_variable_slider_0_sizer,
            value=self.variable_slider_0,
            callback=self.set_variable_slider_0,
            minimum=0,
            maximum=710000,
            num_steps=1000,
            style=wx.SL_HORIZONTAL,
            cast=int,
            proportion=1,
    )
    self.GridAdd(_variable_slider_0_sizer, 10, 10, 1, 9)
    self.uhd_usrp_sink_0 = uhd.usrp_sink(
            device_addr="",
```

```
                stream_args=uhd.stream_args(
                        cpu_format="fc32",
                        channels=range(1),
                ),
        )
        self.uhd_usrp_sink_0.set_samp_rate(samp_rate)
        self.uhd_usrp_sink_0.set_center_freq(2.28e9, 0)
        self.uhd_usrp_sink_0.set_gain(10, 0)
        self.gr_vco_f_0 = gr.vco_f(samp_rate, 6.28e6, 1)
        self.gr_hilbert_fc_0 = gr.hilbert_fc(64)
        self.analog_sig_source_x_0 = analog.sig_source_f(samp_rate,
analog.GR_SAW_WAVE, variable_slider_0_0, amplitude, 0)

        ##################################################
        # Connections
        ##################################################
        self.connect((self.analog_sig_source_x_0, 0), (self.gr_vco_f_0, 0))
        self.connect((self.gr_vco_f_0, 0), (self.gr_hilbert_fc_0, 0))
        self.connect((self.gr_hilbert_fc_0, 0), (self.uhd_usrp_sink_0, 0))


    def get_variable_slider_2(self):
        return self.variable_slider_2

    def set_variable_slider_2(self, variable_slider_2):
        self.variable_slider_2 = variable_slider_2
        self._variable_slider_2_slider.set_value(self.variable_slider_2)
        self._variable_slider_2_text_box.set_value(self.variable_slider_2)
        self.set_amplitude(self.variable_slider_2)

    def get_variable_slider_1(self):
        return self.variable_slider_1

    def set_variable_slider_1(self, variable_slider_1):
        self.variable_slider_1 = variable_slider_1
        self.set_gain(self.variable_slider_1)
        self._variable_slider_1_slider.set_value(self.variable_slider_1)
        self._variable_slider_1_text_box.set_value(self.variable_slider_1)

    def get_variable_slider_0_0(self):
        return self.variable_slider_0_0

    def set_variable_slider_0_0(self, variable_slider_0_0):
        self.variable_slider_0_0 = variable_slider_0_0
```

```
        self.set_test_freq(self.variable_slider_0_0)
        self._variable_slider_0_0_slider.set_value(self.variable_slider_0_0)
        self._variable_slider_0_0_text_box.set_value(self.variable_slider_0_0)
        self.analog_sig_source_x_0.set_frequency(self.variable_slider_0_0)

def get_variable_slider_0(self):
        return self.variable_slider_0

def set_variable_slider_0(self, variable_slider_0):
        self.variable_slider_0 = variable_slider_0
        self.set_delay_length(self.variable_slider_0)
        self._variable_slider_0_slider.set_value(self.variable_slider_0)
        self._variable_slider_0_text_box.set_value(self.variable_slider_0)

def get_test_freq(self):
        return self.test_freq

def set_test_freq(self, test_freq):
        self.test_freq = test_freq

def get_samp_rate(self):
        return self.samp_rate

def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.analog_sig_source_x_0.set_sampling_freq(self.samp_rate)
        self.uhd_usrp_sink_0.set_samp_rate(self.samp_rate)

def get_gain(self):
        return self.gain

def set_gain(self, gain):
        self.gain = gain

def get_delay_length(self):
        return self.delay_length

def set_delay_length(self, delay_length):
        self.delay_length = delay_length

def get_amplitude(self):
        return self.amplitude

def set_amplitude(self, amplitude):
```

```
                self.amplitude = amplitude
                self.analog_sig_source_x_0.set_amplitude(self.amplitude)

if __name__ == '__main__':
        parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
        (options, args) = parser.parse_args()
        tb = top_block()
        tb.Run(True)
```

Clutter.py

```
#!/usr/bin/env python

from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio import uhd
from gnuradio.eng_option import eng_option
from gnuradio.gr import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class top_block(grc_wxgui.top_block_gui):

        def __init__(self):
                grc_wxgui.top_block_gui.__init__(self, title="Top Block")
                _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
                self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

                ##################################################
                # Variables
                ##################################################
                self.variable_slider_2 = variable_slider_2 = .1
                self.variable_slider_1 = variable_slider_1 = 32
                self.variable_slider_0 = variable_slider_0 = 0
                self.samp_rate = samp_rate = 21e6
                self.gain = gain = variable_slider_1
                self.delay_length = delay_length = variable_slider_0
                self.amplitude = amplitude = variable_slider_2

                ##################################################
```

```
# Blocks
##################################################
_variable_slider_2_sizer = wx.BoxSizer(wx.VERTICAL)
self._variable_slider_2_text_box = forms.text_box(
        parent=self.GetWin(),
        sizer=_variable_slider_2_sizer,
        value=self.variable_slider_2,
        callback=self.set_variable_slider_2,
        label="Chirp Amplitude",
        converter=forms.float_converter(),
        proportion=0,
)
self._variable_slider_2_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_variable_slider_2_sizer,
        value=self.variable_slider_2,
        callback=self.set_variable_slider_2,
        minimum=0,
        maximum=1,
        num_steps=10,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
)
self.Add(_variable_slider_2_sizer)
_variable_slider_1_sizer = wx.BoxSizer(wx.VERTICAL)
self._variable_slider_1_text_box = forms.text_box(
        parent=self.GetWin(),
        sizer=_variable_slider_1_sizer,
        value=self.variable_slider_1,
        callback=self.set_variable_slider_1,
        label="Output Gain",
        converter=forms.float_converter(),
        proportion=0,
)
self._variable_slider_1_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_variable_slider_1_sizer,
        value=self.variable_slider_1,
        callback=self.set_variable_slider_1,
        minimum=0,
        maximum=32,
        num_steps=31,
        style=wx.SL_HORIZONTAL,
```

```
                cast=float,
                proportion=1,
            )
            self.GridAdd(_variable_slider_1_sizer, 12, 10, 1, 9)
            _variable_slider_0_sizer = wx.BoxSizer(wx.VERTICAL)
            self._variable_slider_0_text_box = forms.text_box(
                parent=self.GetWin(),
                sizer=_variable_slider_0_sizer,
                value=self.variable_slider_0,
                callback=self.set_variable_slider_0,
                label="Delay Length",
                converter=forms.int_converter(),
                proportion=0,
            )
            self._variable_slider_0_slider = forms.slider(
                parent=self.GetWin(),
                sizer=_variable_slider_0_sizer,
                value=self.variable_slider_0,
                callback=self.set_variable_slider_0,
                minimum=0,
                maximum=710000,
                num_steps=1000,
                style=wx.SL_HORIZONTAL,
                cast=int,
                proportion=1,
            )
            self.GridAdd(_variable_slider_0_sizer, 10, 10, 1, 9)
            self.uhd_usrp_sink_0 = uhd.usrp_sink(
                device_addr="",
                stream_args=uhd.stream_args(
                    cpu_format="fc32",
                    channels=range(1),
                ),
            )
            self.uhd_usrp_sink_0.set_samp_rate(samp_rate)
            self.uhd_usrp_sink_0.set_center_freq(2.28e9, 0)
            self.uhd_usrp_sink_0.set_gain(gain, 0)
            self.blocks_file_source_0 = blocks.file_source(gr.sizeof_gr_complex*1,
"/home/ubuntu/radar2.capture", True)

            ##################################################
            # Connections
            ##################################################
            self.connect((self.blocks_file_source_0, 0), (self.uhd_usrp_sink_0, 0))
```

```
def get_variable_slider_2(self):
        return self.variable_slider_2

def set_variable_slider_2(self, variable_slider_2):
        self.variable_slider_2 = variable_slider_2
        self._variable_slider_2_slider.set_value(self.variable_slider_2)
        self._variable_slider_2_text_box.set_value(self.variable_slider_2)
        self.set_amplitude(self.variable_slider_2)

def get_variable_slider_1(self):
        return self.variable_slider_1

def set_variable_slider_1(self, variable_slider_1):
        self.variable_slider_1 = variable_slider_1
        self.set_gain(self.variable_slider_1)
        self._variable_slider_1_slider.set_value(self.variable_slider_1)
        self._variable_slider_1_text_box.set_value(self.variable_slider_1)

def get_variable_slider_0(self):
        return self.variable_slider_0

def set_variable_slider_0(self, variable_slider_0):
        self.variable_slider_0 = variable_slider_0
        self.set_delay_length(self.variable_slider_0)
        self._variable_slider_0_slider.set_value(self.variable_slider_0)
        self._variable_slider_0_text_box.set_value(self.variable_slider_0)

def get_samp_rate(self):
        return self.samp_rate

def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.uhd_usrp_sink_0.set_samp_rate(self.samp_rate)

def get_gain(self):
        return self.gain

def set_gain(self, gain):
        self.gain = gain
        self.uhd_usrp_sink_0.set_gain(self.gain, 0)

def get_delay_length(self):
```

```
                return self.delay_length

        def set_delay_length(self, delay_length):
                self.delay_length = delay_length

        def get_amplitude(self):
                return self.amplitude

        def set_amplitude(self, amplitude):
                self.amplitude = amplitude

if __name__ == '__main__':
        parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
        (options, args) = parser.parse_args()
        tb = top_block()
        tb.Run(True)
```

DRFM.py

```
#!/usr/bin/env python

from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio import uhd
from gnuradio import window
from gnuradio.eng_option import eng_option
from gnuradio.gr import firdes
from gnuradio.wxgui import forms
from gnuradio.wxgui import waterfallsink2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class top_block(grc_wxgui.top_block_gui):

        def __init__(self):
                grc_wxgui.top_block_gui.__init__(self, title="Top Block")
                _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
                self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

                ##################################################
```

```
# Variables
##################################################
self.variable_slider_1 = variable_slider_1 = 32
self.variable_slider_0 = variable_slider_0 = 0
self.samp_rate = samp_rate = 21e6
self.gain = gain = variable_slider_1
self.delay_length = delay_length = variable_slider_0

##################################################
# Blocks
##################################################
self.wxgui_waterfallsink2_0_0 = waterfallsink2.waterfall_sink_c(
        self.GetWin(),
        baseband_freq=0,
        dynamic_range=100,
        ref_level=0,
        ref_scale=2.0,
        sample_rate=samp_rate,
        fft_size=512,
        fft_rate=15,
        average=False,
        avg_alpha=None,
        title="Output Waterfall Plot",
)
self.GridAdd(self.wxgui_waterfallsink2_0_0.win, 0, 10, 10, 10)
self.wxgui_waterfallsink2_0 = waterfallsink2.waterfall_sink_c(
        self.GetWin(),
        baseband_freq=0,
        dynamic_range=100,
        ref_level=0,
        ref_scale=2.0,
        sample_rate=samp_rate,
        fft_size=512,
        fft_rate=15,
        average=False,
        avg_alpha=None,
        title="Input Waterfall Plot",
)
self.GridAdd(self.wxgui_waterfallsink2_0.win, 0, 0, 10, 10)
_variable_slider_1_sizer = wx.BoxSizer(wx.VERTICAL)
self._variable_slider_1_text_box = forms.text_box(
        parent=self.GetWin(),
        sizer=_variable_slider_1_sizer,
        value=self.variable_slider_1,
```

```
            callback=self.set_variable_slider_1,
            label="Output Gain",
            converter=forms.float_converter(),
            proportion=0,
    )
    self._variable_slider_1_slider = forms.slider(
            parent=self.GetWin(),
            sizer=_variable_slider_1_sizer,
            value=self.variable_slider_1,
            callback=self.set_variable_slider_1,
            minimum=0,
            maximum=32,
            num_steps=31,
            style=wx.SL_HORIZONTAL,
            cast=float,
            proportion=1,
    )
    self.GridAdd(_variable_slider_1_sizer, 12, 10, 1, 9)
    _variable_slider_0_sizer = wx.BoxSizer(wx.VERTICAL)
    self._variable_slider_0_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_variable_slider_0_sizer,
            value=self.variable_slider_0,
            callback=self.set_variable_slider_0,
            label="Delay Length",
            converter=forms.int_converter(),
            proportion=0,
    )
    self._variable_slider_0_slider = forms.slider(
            parent=self.GetWin(),
            sizer=_variable_slider_0_sizer,
            value=self.variable_slider_0,
            callback=self.set_variable_slider_0,
            minimum=0,
            maximum=710000,
            num_steps=1000,
            style=wx.SL_HORIZONTAL,
            cast=int,
            proportion=1,
    )
    self.GridAdd(_variable_slider_0_sizer, 10, 10, 1, 9)
    self.uhd_usrp_source_0 = uhd.usrp_source(
            device_addr="",
            stream_args=uhd.stream_args(
```

```
                        cpu_format="fc32",
                        channels=range(1),
                ),
        )
        self.uhd_usrp_source_0.set_samp_rate(samp_rate)
        self.uhd_usrp_source_0.set_center_freq(2.28e9, 0)
        self.uhd_usrp_source_0.set_gain(0, 0)
        self.uhd_usrp_sink_0 = uhd.usrp_sink(
                device_addr="",
                stream_args=uhd.stream_args(
                        cpu_format="fc32",
                        channels=range(1),
                ),
        )
        self.uhd_usrp_sink_0.set_samp_rate(samp_rate)
        self.uhd_usrp_sink_0.set_center_freq(2.28e9, 0)
        self.uhd_usrp_sink_0.set_gain(gain, 0)
        self.gr_file_source_0_0 = gr.file_source(gr.sizeof_gr_complex*1,
"/home/ubuntu/radar-rx3.capture", True)
        self.gr_file_source_0 = gr.file_source(gr.sizeof_gr_complex*1,
"/home/ubuntu/radar-rx3.capture", True)
        self.gr_delay_0_0 = gr.delay(gr.sizeof_gr_complex*1, delay_length)
        self.blocks_multiply_xx_0 = blocks.multiply_vcc(1)

        ##################################################
        # Connections
        ##################################################
        self.connect((self.uhd_usrp_source_0, 0), (self.wxgui_waterfallsink2_0, 0))
        self.connect((self.gr_file_source_0_0, 0), (self.gr_delay_0_0, 0))
        self.connect((self.gr_file_source_0, 0), (self.blocks_multiply_xx_0, 0))
        self.connect((self.gr_delay_0_0, 0), (self.blocks_multiply_xx_0, 1))
        self.connect((self.blocks_multiply_xx_0, 0), (self.uhd_usrp_sink_0, 0))
        self.connect((self.blocks_multiply_xx_0, 0), (self.wxgui_waterfallsink2_0_0, 0))


    def get_variable_slider_1(self):
        return self.variable_slider_1

    def set_variable_slider_1(self, variable_slider_1):
        self.variable_slider_1 = variable_slider_1
        self.set_gain(self.variable_slider_1)
        self._variable_slider_1_slider.set_value(self.variable_slider_1)
        self._variable_slider_1_text_box.set_value(self.variable_slider_1)
```

```
    def get_variable_slider_0(self):
        return self.variable_slider_0

    def set_variable_slider_0(self, variable_slider_0):
        self.variable_slider_0 = variable_slider_0
        self.set_delay_length(self.variable_slider_0)
        self._variable_slider_0_slider.set_value(self.variable_slider_0)
        self._variable_slider_0_text_box.set_value(self.variable_slider_0)

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.wxgui_waterfallsink2_0.set_sample_rate(self.samp_rate)
        self.wxgui_waterfallsink2_0_0.set_sample_rate(self.samp_rate)
        self.uhd_usrp_sink_0.set_samp_rate(self.samp_rate)
        self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)

    def get_gain(self):
        return self.gain

    def set_gain(self, gain):
        self.gain = gain
        self.uhd_usrp_sink_0.set_gain(self.gain, 0)

    def get_delay_length(self):
        return self.delay_length

    def set_delay_length(self, delay_length):
        self.delay_length = delay_length
        self.gr_delay_0_0.set_delay(self.delay_length)

if __name__ == '__main__':
    parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
    (options, args) = parser.parse_args()
    tb = top_block()
    tb.Run(True)
```

Radar.m

```
function userinput

sprintf('How would you like to process the data?')
prompt = '1 for RealTime Audio Processing 2 for First acquiring data and then processing';
result = input(prompt);

if result == 1
   final_workingrealtime;
userinput;
elseif result == 2
   notrealtime;
userinput;
else
   sprintf('Undefined user input')
end


%MIT IAP Radar Course 2011
%Resource: Build a Small Radar System Capable of Sensing Range, Doppler,
%and Synthetic Aperture Radar Imaging
%
%Gregory L. Charvat
%Om Kapoor
%Hunter Scott

%Process Range vs. Time Intensity (RTI) plot

%NOTE: set up-ramp sweep from 2-3.2V to stay within ISM band
%change fstart and fstop bellow when in ISM band
function final_workingrealtime
clear all;
close all;
pause on
%read the raw data .wave file here
Yt =  audiorecorder(16000,16,2);
% Y = wavrecord(44100*5,44100,2);
FS = 16000;
NBITS = 16;

%constants
c = 3E8; %(m/s) speed of light

%radar parameters
```

```
Tp = 20E-3; %(s) pulse time
N = Tp*FS; %# of samples per pulse
fstart = 2260E6; %(Hz) LFM start frequency for example
fstop = 2590E6; %(Hz) LFM stop frequency for example
%fstart = 2402E6; %(Hz) LFM start frequency for ISM band
%fstop = 2495E6; %(Hz) LFM stop frequency for ISM band
BW = fstop-fstart; %(Hz) transmti bandwidth
f = linspace(fstart, fstop, N/2); %instantaneous transmit frequency
x = 0;
%range resolution
rr = c/(2*BW);
max_range = rr*N/2;
record(Yt);
pause(8)
n = 1;


%%%%
%Find the size of



%%%
while(1)
Ybar = getaudiodata(Yt, 'single');
sz = size(Ybar);
%the input appears to be inverted
%Method 2
if x == 0
   Y = Ybar;
   x = 1;
else
   Y = [Y(size(Ybar,1)-5000:size(Y,1),:);Ybar(5000:size(Ybar,1),:)];
end
trig = -1*Y(:,1);
s = -1*Y(:,2);



%parse the data here by triggering off rising edge of sync pulse
count = 0;
thresh = 0;
start = (trig > thresh);
for ii = 100:(size(start,1)-N)
```

```
    if start(ii) == 1 & mean(start(ii-11:ii-1)) == 0
        %start2(ii) = 1;
        count = count + 1;
        sif(count,:) = s(ii:ii+N-1);
        time(count) = ii*1/FS;
    end
end
%subtract the average
ave = mean(sif,1);
for ii = 1:size(sif,1);
    sif(ii,:) = sif(ii,:) - ave;
end

zpad = 8*N/2;

figure(20);
sif2 = sif(2:size(sif,1),:)-sif(1:size(sif,1)-1,:);
v = ifft(sif2,zpad,2);
S=v;
R = linspace(0,max_range,zpad);
for ii = 1:size(S,1)
    %S(ii,:) = S(ii,:).*R.^(3/2); %Optional: magnitude scale to range
end
S = dbv(S(:,1:size(v,2)/2));
m = max(max(S));
imagesc(R,time,S-m,[-80, 0]);
colorbar;
ylabel('time (s)');
xlabel('range (m)');
title('RTI with 2-pulse cancelor clutter rejection');

record(Yt)
pause(2)
stop(Yt)
end
end

function notrealtime
%read the raw data .wave file here
Y =  wavrecord(16000*10,16000,2);
% Y = wavrecord(44100*5,44100,2);
FS = 16000;
NBITS = 16;
```

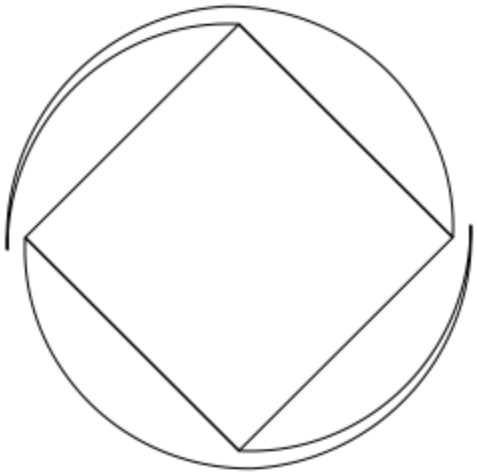
```
%constants
c = 3E8; %(m/s) speed of light
%radar parameters
Tp = 20E-3; %(s) pulse time
N = Tp*FS; %# of samples per pulse
fstart = 2260E6; %(Hz) LFM start frequency for example
fstop = 2590E6; %(Hz) LFM stop frequency for example
%fstart = 2402E6; %(Hz) LFM start frequency for ISM band
%fstop = 2495E6; %(Hz) LFM stop frequency for ISM band
BW = fstop-fstart; %(Hz) transmti bandwidth
f = linspace(fstart, fstop, N/2); %instantaneous transmit frequency
x = 0;
%range resolution
rr = c/(2*BW);
max_range = rr*N/2;
trig = -1*Y(:,1);
s = -1*Y(:,2);
%parse the data here by triggering off rising edge of sync pulse
count = 0;
thresh = 0;
start = (trig > thresh);
for ii = 100:(size(start,1)-N)
    if start(ii) == 1 & mean(start(ii-11:ii-1)) == 0
        %start2(ii) = 1;
        count = count + 1;
        sif(count,:) = s(ii:ii+N-1);
        time(count) = ii*1/FS;
    end
end
%subtract the average
ave = mean(sif,1);
for ii = 1:size(sif,1);
    sif(ii,:) = sif(ii,:) - ave;
end

zpad = 8*N/2;

figure(20);
sif2 = sif(2:size(sif,1),:)-sif(1:size(sif,1)-1,:);
v = ifft(sif2,zpad,2);
S=v;
R = linspace(0,max_range,zpad);
for ii = 1:size(S,1)
    %S(ii,:) = S(ii,:).*R.^(3/2); %Optional: magnitude scale to range
```

```
end
S = dbv(S(:,1:size(v,2)/2));
m = max(max(S));
imagesc(R,time,S-m,[-80, 0]);
colorbar;
ylabel('time (s)');
xlabel('range (m)');
title('RTI with 2-pulse cancelor clutter rejection');
end
end
```

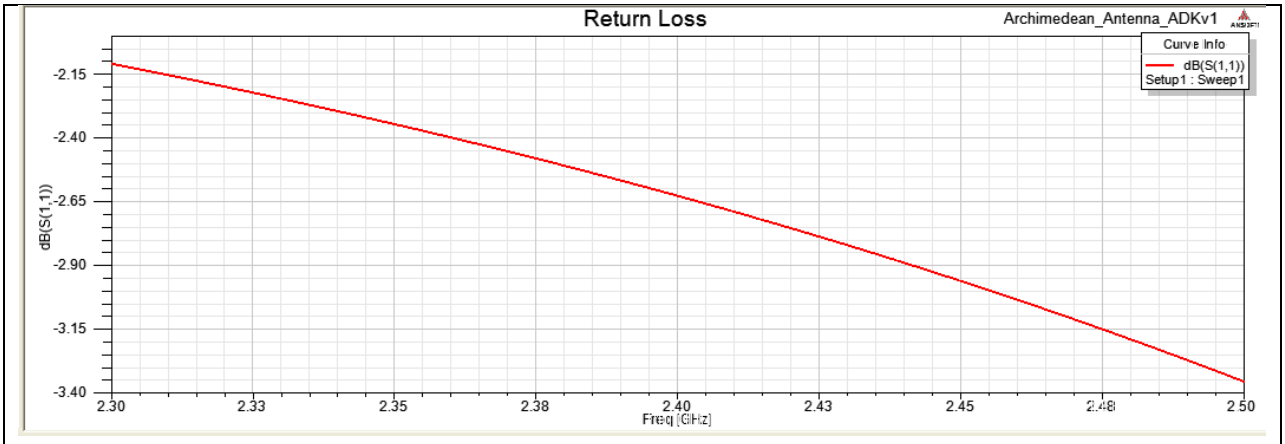
radartracker.m

```
img = imread('plot.png');
%search the intensity of the red layer of the array (im[0,:,:])
imshow(img);
imgred = img;
imshow(imgred);
imgred(:,:,2) = 0;
imshow(imgred);
imgred(:,:,3) = 0;
imshow(imgred)
imgred = rgb2gray(imgred);

top = 15;% pixel
bottom =25;%pixels from bottom
margin_left = 40;%pixel from left to right
margin_right = 5;%pixel from right to left

J =  imgred(top:size(imgred,1) - bottom , margin_left:size(imgred,2) - margin_right);
imshow(J)
vi = (J > 70);
imshow(vi)
```

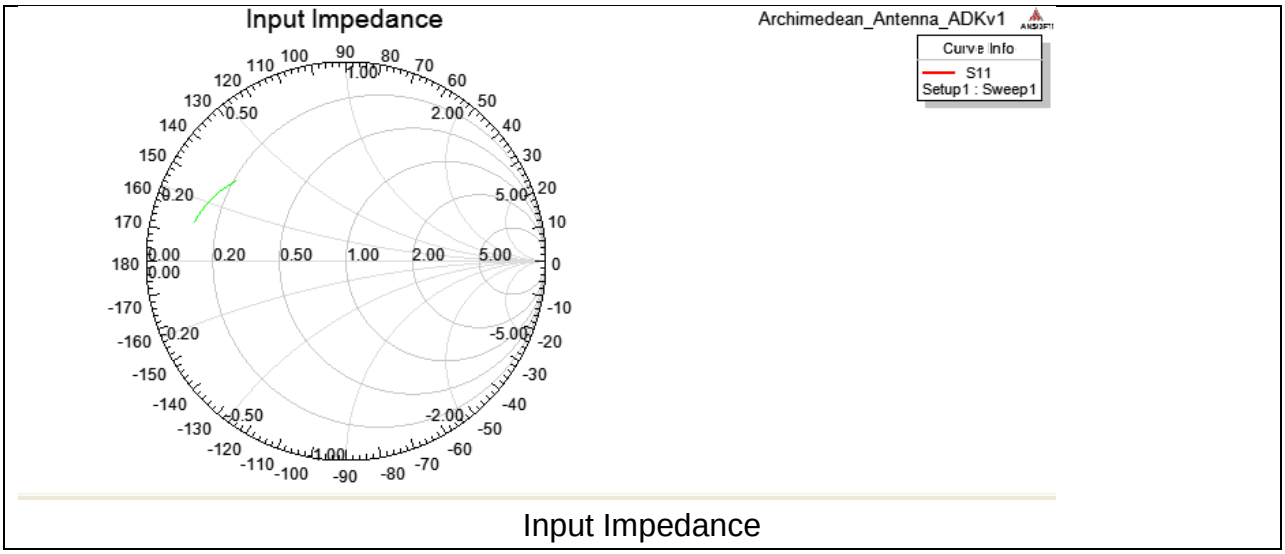
Appendix E – Circular Antenna Design



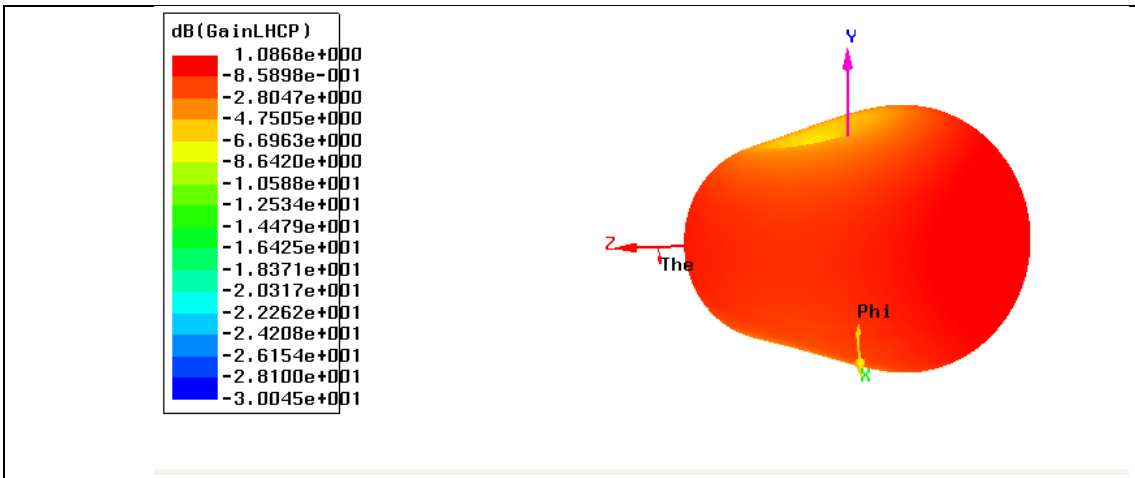
Antenna Geometry, scaled by 2.5



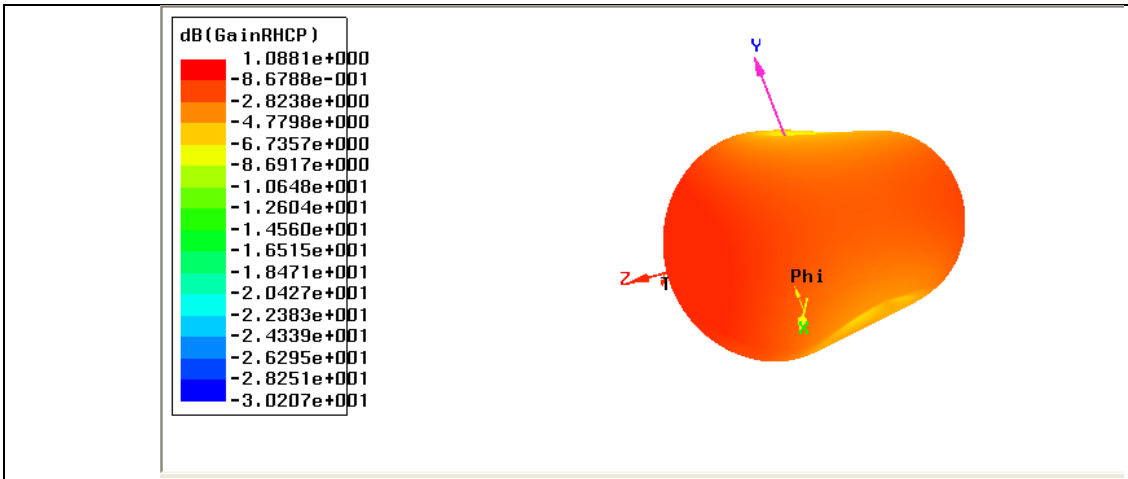
Return Loss



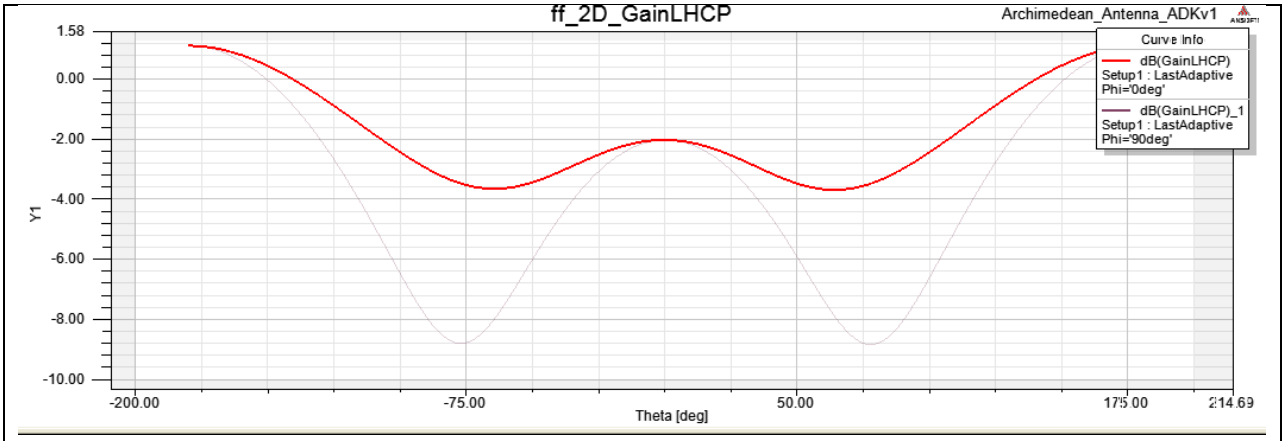
Input Impedance



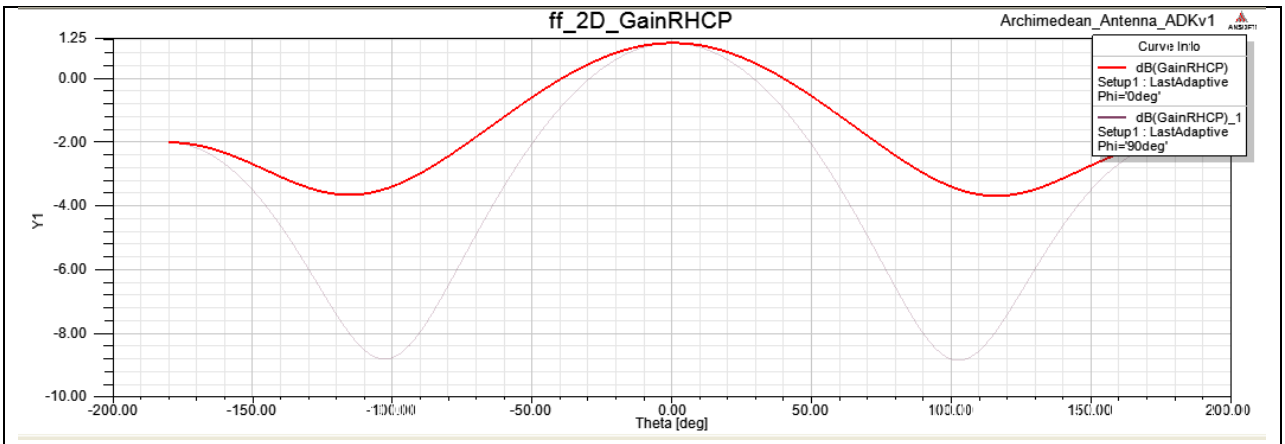
3D Left Hand Radiation Pattern



3D Right Hand Radiation Pattern



2D Left Hand Gain



2D Right Hand Gain