بسم الله الرحمن الرحيم

Sudan University of Science and Technology

College of Postgraduate Studies

Active Cancellation Algorithm for Radar Cross Section Reduction

خوارزمية الغاء نشط لتخفيض المقطع الرادري

Thesis Submitted for the Partial Fulfillment of the Requirement for the Award of the Degree of Ph. D. in Computer Engineering

Prepared by:

Isam Abdel Nabi Osman Mohammed Ali

Supervised by:

Associate prof. Dr. Abdelrasol Jabar Alzobidy

March 2014

بسم الله الرحمن الرحيم

قال تعالى :

 إقرأ باسم ربك الذي خلق خلق الإنسان من علق أقرأ وربك الأكرم الذي علم بالقلم علم الإنسان مالم يعلم

صدق الله العظيم

سورة العلق الايات 1-5

ACKNOWLEDGEMENT

I like to acknowledge the contributions of my supervisor Dr. Abdelrasol Jabar Alzobidy who helps brilliant creativity

support, and have guided this project from its conception to its completion, and I would like to send my special thanks to him.

I thank the teaching staff of the department of electronic engineering, who gave me their considerations and were always behind me helping and encouraging.

Special thanks for my family for their usual supports without limitations, to accomplish this work and standing behind me.

DEDICATION

Thank you god, for giving me the courage, ability and strength to accomplish this work.

I dedicate this deepest love and affection to my parents, my wife and my sons. Their love, vulnerability and wisdom have inspired me to do the best I can.

To my friends, thanks for the endearing friendship, for the joy I have always shared, for the days filled with humor and laughter and for bond of endless understanding.

To all the people who helped me to accomplish this work.

With my greater love

Isam ...

ABSTRACT

Modern components for signal processing and high speed computers make it possible to achieve radar visibility reduction (Stealth Technology), that requires reduce the radar cross section (RCS) of an aircraft or the fighting's systems because it seems to be on the enemy's radar detection capabilities. To achieve this goal, this research proposed an Active cancellation algorithm for radar cross section reduction using MATLAB, digital radio-frequency memory (DRFM), and phased array technology to generate the desired signal to cancel the reflected radar returns. The algorithm depends on a pre calculation approach in which an omni direction RCS, clutter, and noise databases generated in advance. Signal processing system function analysis parameter of the measured radar signal, and then find the corresponding echo data (amplitude and phase etc …) in the target RCS database through real-time amendment to generate out of phase echo. Through the establishment of a target scattering field with the abolition of a coherent signal in the direction of the radar system detection, the radar receiver stays in empty pattern synthesis. The result achieved by the proposed method improves visibility reduction by 50%.

مستخلص

المكونات الحديثة لمعالجة الاشارات وحاسبات السرعة العالية تجعل بالامكان تحقيق خفض الروية للرادار (تكنولوجيا التخفي) والتي تطلب إنقاص مقطع الرادار العرضي (RCS) للطائرة او الانظمة القتالية لانها تمثل مقدرة رادار العدو من الكشف. لتحقيق هذا الهدف، اقترح هذ البحث خوارزمية الالغاء النشط لتخفيض مقطع الرادار العرضي باستخدام الماتلاب ، ذواكر الاشارة الراديوية الرقمية (DRFM) وتقنية الصف المرحلية لتوليد الاشارة المطلوبة لالغاء عائدات الرادار المنعكسة. الخوارزمية تعتمد علي مبدأ تكوين قواعدبيانات المقطع العرضى متعدد الاتجاه، البيئة المحيطة، والضجيج مسبقاً وتقوم وحدة معالجة الاشارة بتحليل اشارة الرادار المستقبلة ومن ثم ايجاد معاملات الارتداد (المطال والطور الخ...) المطابقة لها في قاعدة بيانات الهدف من خلال الضبط والتعديل الفوري يتم توليد اشارة معاكسة الطور وارسالها في اتجاه الرادر المكتشف ليظل شكل الاشارة لمستقبل الرادار خالياً. النتيجة المحققة بواسطة الخوارزمية المقترحة تحسن خفض الروية بـ 50%.

TABLE OF CONTENTS

1. LIST OF ABBREVIATIONS

RCS	Radar Cross Section
DRFM	Digital Radio Frequency Memory
GO	Geometric Optics
PO	Physical Optics
GTD	Geometric Theory of Diffraction
PTD	Physical Theory of Diffraction
RAM	Radar Absorbent Material
RADAR	RAdio Detection And Ranging
IF	Intermediate Frequency
CW	Continous Wave
MTI	Moving Target Indication
CFAR	Constant False Alarm Rate
ADT	Automatic Detector and Tracker
PR	Pulsed Radar
PRF	Pulse Repition Frequency
PRI	Pulse Repition Interval
PP	Principle Polarizaton
OP	Orthognal Polarizaton
HFA	High Frequency Asymptic
FMM	Fast Multiple Method
FTDT	Finite – Difference Time Domain
MoM	Method of Moments
PEC	Perfectly Electric Conductor
EFIE	Electric Field Integral Equation
RCSR	Radar Cross Section Reduction
RF	Radio Frequency
LPF	Low Pass Filter
BW	Band Width
GUI	Graphical User Interface
SPC	Signal Processing and Control
BVDE	Blind Velocity and Doppler Effect
PSBF	Power Syntheis and Beam Forming
PRT	Pulse Repition Time

LIST OF FIGURES

Chapter One

2. CHAPTER ONE: INTRODUCTION

2.1 Overview:

Radar cross section (RCS) is the measure of a target's ability to reflect radar signals in the direction of the radar receiver, i.e. it is a measure of the ratio of backscatter power per unit solid angle in the direction of the radar (from the target) to the power density that is intercepted by the target. The RCS of a target can be viewed as a comparison of the strength of the reflected signal from a target to the reflected signal from a perfectly smooth sphere of cross sectional area of 1 m^2 as shown in Figure 1.1[1].

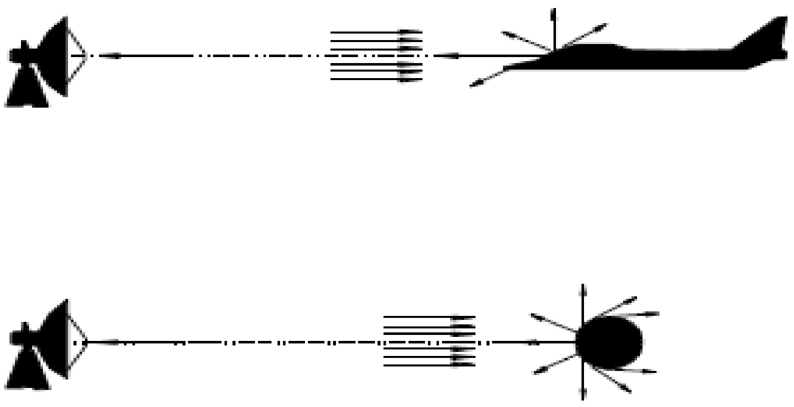


Figure 1.1: Concept of Radar Cross Section

RCS, denoted by the Greek letter σ and measured in m², is defined as:

$$\sigma = 4\Pi \frac{P_s}{P_i} \qquad \ldots. \qquad 1.1$$

P_i: = power density, or intensity, of a plane wave striking the target.

P_s: = power per unit solid angle reflected by the target.

RCS has a wide spread ranging from 10^{-5} for small insects to 10^{6} for large ships. Hence, RCS is often stated in the logarithmic decibel scale:

$$\sigma_{db} = 10 \log\left(\frac{\sigma}{1.m^2}\right) \qquad \ldots. \qquad 1.2$$

2.1.1 RCS is a function of:

- Position of transmitter/receiver relative to target.
- Target geometry and material composition.
- Angular orientation of target relative to transmitter/receiver.

- Frequency or wavelength.
- Antenna polarization.

The conceptual definition of RCS includes the fact that not all of the radiated energy falls on the target. A target's RCS (σ) is most easily visualized as the product of three factors [5]:

σ = **Projected cross section** x **Reflectivity** x **Directivity.**

RCS (σ) is used above for an equation representing power reradiated from the target.

- **Reflectivity:**

  The re-radiated fraction of intesrcepted power.

- **Directivity:**

  The ratio ratio of the maximum intensity of the radiator to the intensity of an isotropic source.

Figure 1.2 and 1.3 shows that RCS does not equal geometric area. For a sphere, the RCS, $\sigma = \Pi r^2$, where r is the radius of the sphere.

The RCS of a sphere is independent of frequency if operating at sufficiently high frequencies where $\lambda \ll$Range, and $\lambda \ll$ radius (r). Experimentally, radar return reflected from a target is compared to the Radar return reflected from a sphere which has a frontal or projected area of one square meter. Using the spherical shape aids in field or laboratory measurements since orientation or positioning of the sphere will not affect radar reflection intensity measurements as a flat plate would. If calibrated, other sources (cylinder, flat plate, or corner reflector, etc.) could be used for comparative measurements [1],[5].

:

l

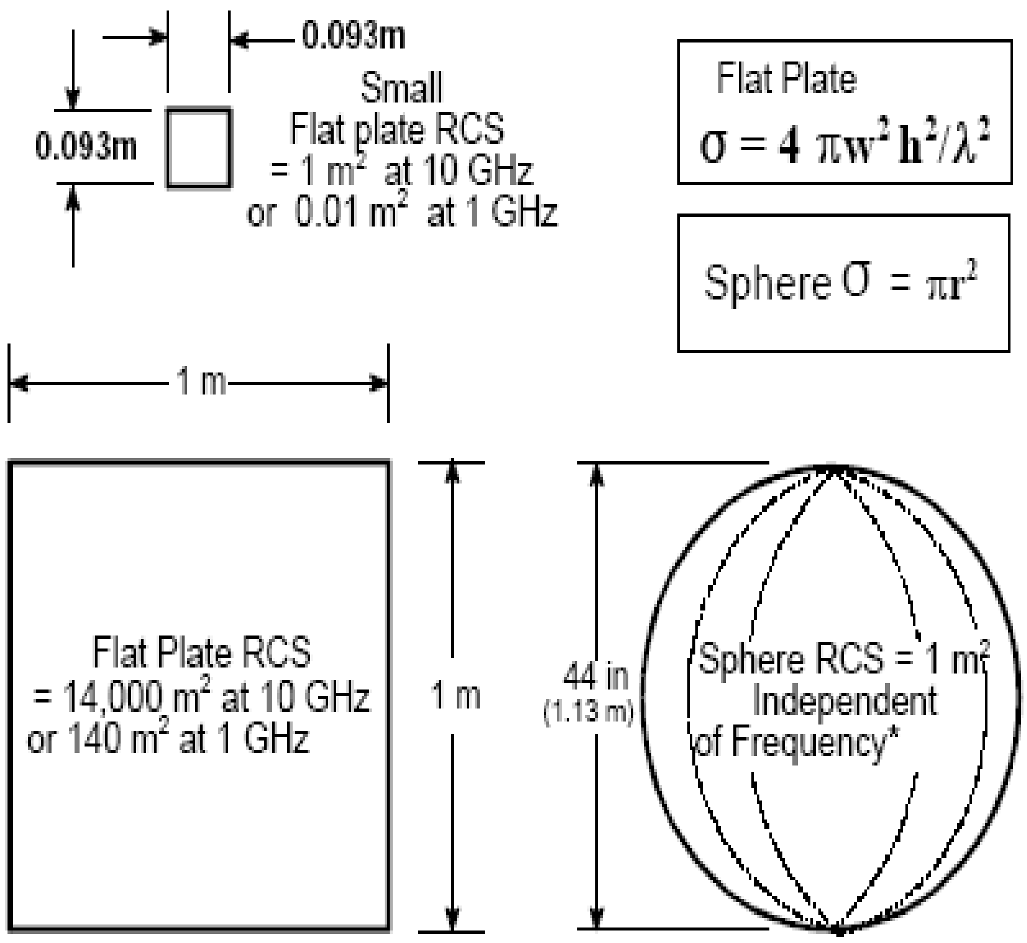


Figure 1.2 RCS versus Physical Geometry

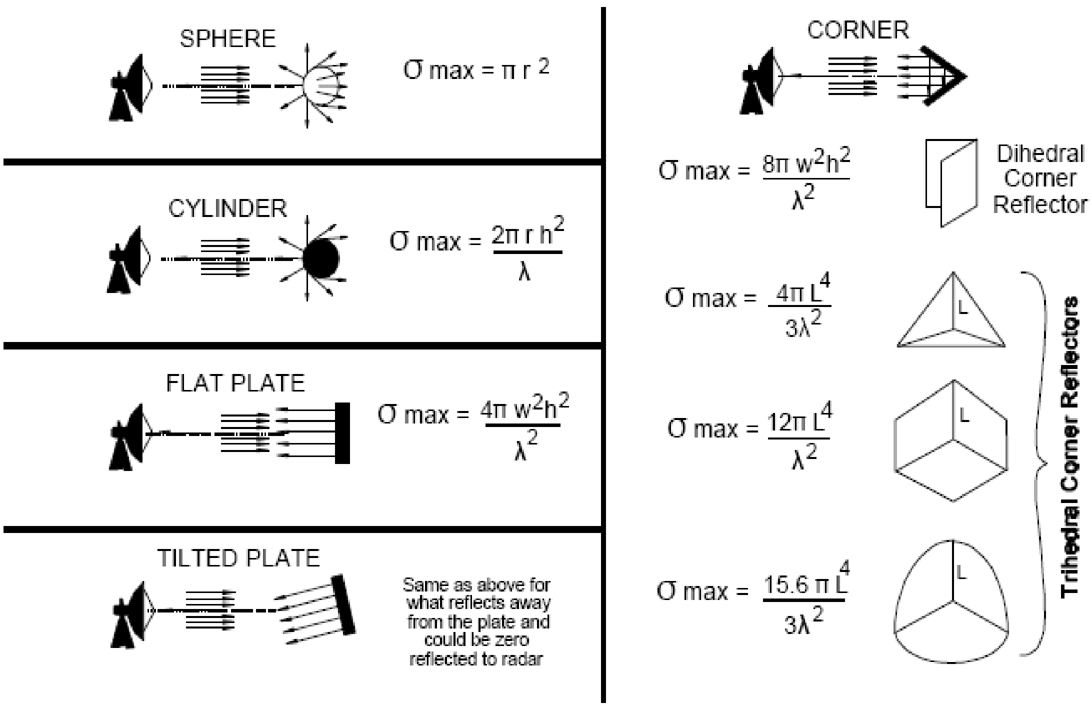


Figure 1.3: Backscatter from Shapes

In Figure 1.4, RCS patterns are shown as objects are rotated about their vertical axes (the arrows indicate the direction of the radar reflections).

The sphere is essentially the same in all directions. The flat plate has almost no RCS except when aligned directly toward the radar.

The corner reflector has an RCS almost as high as the flat plate but over a wider angle, i.e., over ±60E. The return from a corner reflector is analogous to that of a flat plate always being perpendicular to your collocated transmitter and receiver. Targets such as ships and aircraft often have many effective corners. Corners are sometimes used as calibration targets or as decoys, i.e. corner reflectors. An aircraft target is very complex. It has a great many reflecting elements and shapes. The RCS of real aircraft must be measured. It varies significantly depending upon the direction of the illuminating radar [5].

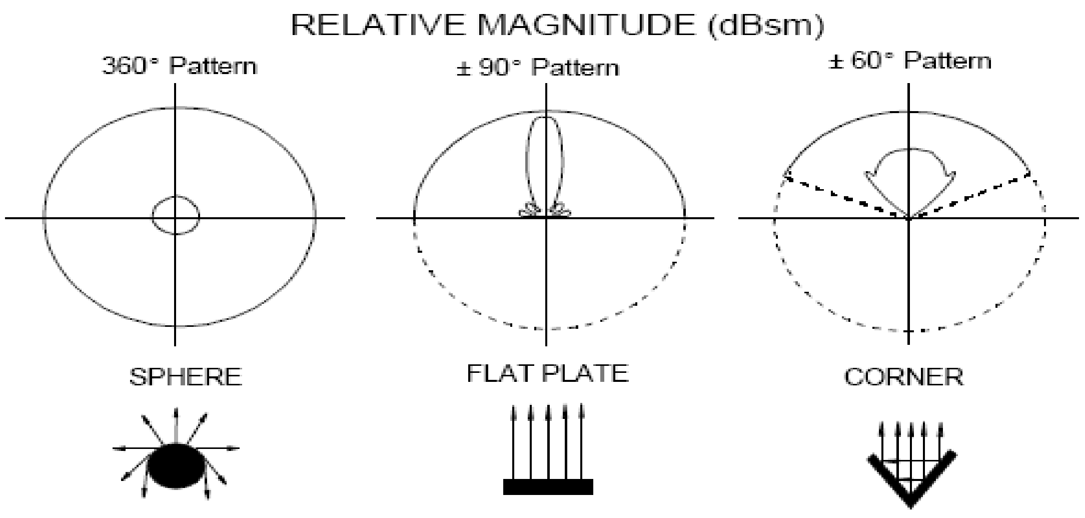


Figure 1.4: RCS Patterns

2.1.2 RCS Prediction Techniques:

Most radar systems use RCS as a means of discrimination. Therefore, accurate prediction of target RCS is critical in order to design and develop strong discrimination algorithms. Additionally, measuring and identifying the scattering centers (sources) for a given target aid in developing RCS reduction techniques. Another reason of lesser importance is that RCS calculations require broad and extensive technical knowledge, thus many scientists and scholars find the subject challenging and intellectually motivating there are two categories of RCS prediction methods are available [2]:

- Exact prediction methods.

- Approximate prediction methods.

Exact methods of RCS prediction are very complex even for simple shape objects. This is because they require solving either differential or integral equations that describe the scattered waves from an object under the proper set of boundary conditions. Such boundary conditions are governed by Maxwell's equations. Even when exact solutions are achievable, they are often difficult to interpret and to program using digital computers. Due to the difficulties associated with the exact RCS prediction, approximate methods become the viable alternative. The majority of the approximate methods are valid in the optical region, and each has its own strengths and limitations. Most approximate methods can predict RCS within few decibels of the truth. In general, such a variation is quite acceptable by radar engineers and designers. Approximate methods are usually the main source for predicting RCS of complex and extended targets such as aircrafts, ships, and missiles. When experimental results are available, they can be used to validate and verify the approximations. Some of the most commonly used prediction methods are Geometrical Optics (GO), Physical Optics (PO), Geometrical Theory of Diffraction (GTD), Physical Theory of Diffraction (PTD).

2.1.3 RCS Reduction Techniques:

In many military and civilian applications it is desirable to reduce the RCS of a target. In general, there are four basic ways to reduce the RCS [5]:

1. **Target shaping:**

   Target shaping refers to the special selection of target surface shapes to minimize the amount of energy scattered back to the radar, typically at the cost of redirection of the scattered energy from the direction of interest to another region of little or no interest. Intake cavities make it impossible to reduce the radar reflections from the object inside of the cavity as shown in Figure 1.5.

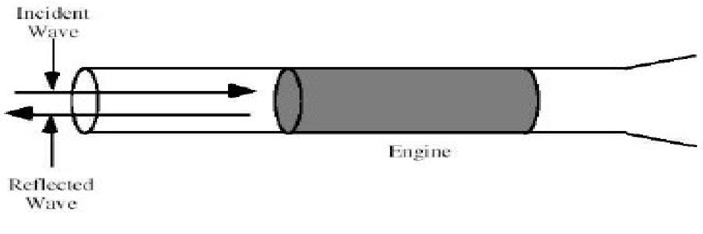


Figure 1.5: Straight Cavity

   With respect to the engine, these cavities make highly reflective turbine blades visible to radar which causes a significant increase in the RCS of the aircraft. Often, engine intakes incorporate an S-shaped duct as seen in Figure 1.6 so that the turbine blades are not visible to radar.

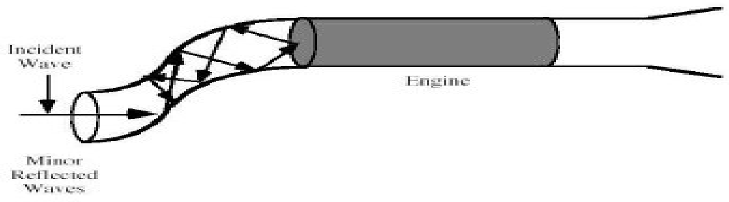


Figure 1.6: S-Shaped Cavity

2. **Radar Absorbent Material**

    Radar absorbent material (RAM) is probably the most common technology used to reduce an aircraft's RCS. An equivalent optical example would be black paint. An object that is painted black absorbs all the light that hits it (i.e., black is the absence of reflected light hitting your eyes). The idea behind RAM paint is to absorb the energy of the radio waves transmitted by the radar antenna. RAM contains carbonyl iron ferrite as the active ingredient. When radar waves hit the RAM coating, a magnetic field is produced in the metallic elements of the coating. The magnetic field has alternating polarity and dissipates the energy of the signal

3. **Passive Cancellation (Discrete loading)**

    Passive cancellation methods require the loading of the target with discrete antenna-like elements (e.g., slots or dipoles) to reduce the overall cross section. These techniques are rather difficult to implement in practice, are severely limited in bandwidth, require the control of possible self-oscillations, and in some cases may even revert to reinforcement when some parameters differ from the nominal values for which the loads were designed.

4. **Active Cancellation System**

    Active cancellation is a way of creating a new waveform which will cancel the original radar signal reflected from the airframe. This method is very similar to active jamming techniques. However, active cancellation methods require transmission of very low power levels compared to conventional EW jamming techniques. Here, the main purpose is reducing the RCS while canceling the reflected radar signal by a process of modifying and retransmitting the incident radar waves, rather than having the radar jammed. The method is also called “active loading”. The active cancellation platform must radiate counter signals, which have the same amplitude but reversed phase from the radar. Moreover the radiation must be towards the same reflected direction to cancel the bounced signal.

    Using such a process requires some data, including angle of arrival, intensity, waveform and the frequency of the received signal. Obviously, the emission must coincide in time with the incident pulse. Thus, this kind of a replication is very complex; moreover, it requires smart systems to calculate the airframe’s own echo characteristics in order to generate the exact pulses for cancellation.

2.2 Problem statement:

There are two active cancellation levels; fully active and semi active. Fully active systems are those that receive, amplify the threat signal and retransmit the required cancellation signal which is out of phase. Here, the transmitted wave parameters, such as signal intensity, phase, frequency and polarization, must be carefully adjusted to compensate for the changes. The second one is semi active cancellation, by which limited changes in threat signal parameters are met to compensate.

The challenge of this study is to design a fully active RCS reduction algorithm using MATLAB R2013a with aid of hardware controlled by onboard computer.

2.3 Objectives:

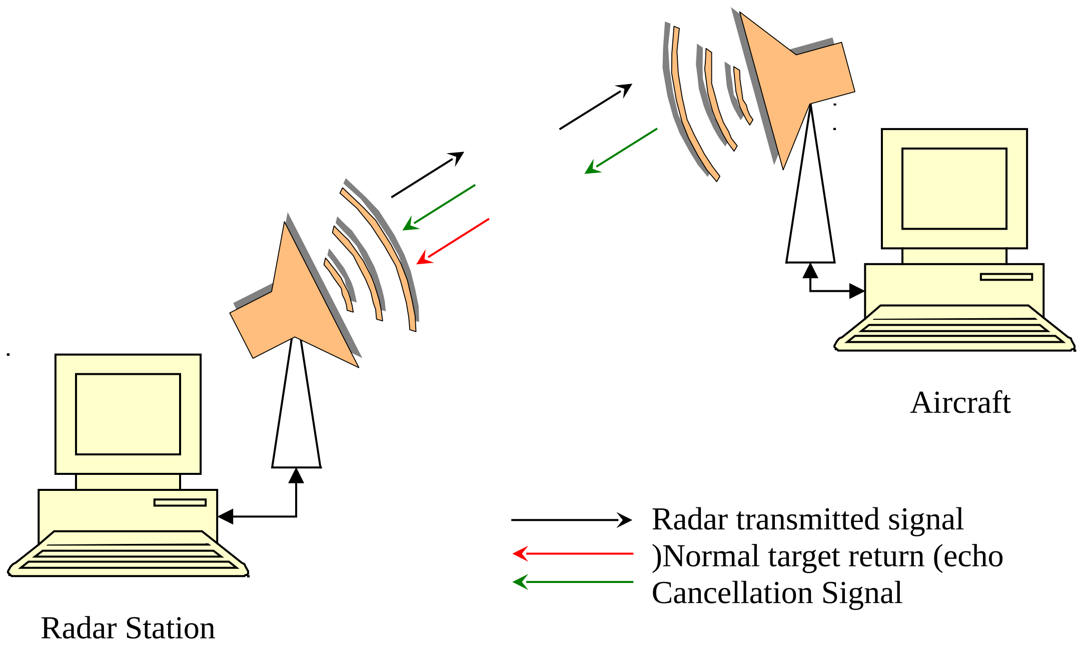
The main objective is to design a fully active RCS reduction algorithm that does the flowing tasks:

- Receive a radar signal and analyzes it to extract signal information like Frequency, amplitude, phase etc…
- Search in database for corresponding echo signal.
- Generate out of phase signal with the extracted information.
- Calculate the Blind velocity to avoid aircraft movement detection.
- Transmit the generated out of phase signal in the direction of the received one.

Due to the cost and security problems, the second objective is to design a radar and aircraft simulation to evaluate the proposed algorithm.

2.4 Methodology:

The methodology used in the research has been started with literature survey for the radar cross section and the technologies used to reduce it. Analyze the requirement to achieve an active cancellation for RCS and design an active cancellation algorithm for radar cross section reduction to analysis the radar received signal and generate equi out of phase of the normal return echo. The proposed algorithm has been evaluated by creating a simulation for radar and aircraft using tow computers running MTLAB as shown in figure 1.7.



.Figure 1.7: Shows the implementation of the algorithm

2.5 Organization of the Thesis

This thesis consists of eight chapters including a conclusion. These chapters are organized into the following: An introductory chapter that contains an overview about RCS and its prediction and reduction techniques in addition to the objectives, problem statement, methodology, and organization of the thesis.

The second chapter is about Radar, radar operation, radar classification and the definition of radar cross section and its specifications.

The third chapter is about the challenges in the area of RCS computation or measurement of large targets and the approximate and exact methods used for RCS prediction or calculation.

Chapter four is to examine methods of controlling RCS and the tradeoffs involved in implementing these methods. Radar cross section reduction techniques and their categories (Target shaping, Materials selection and coatings, Passive cancellation and Active cancellation).

Chapter five is about the simulation that used to evaluate the proposed algorithm.

Chapter six illustrates the novel algorithms proposed by this thesis for radar cross section reduction.

Chapter seven discusses the results of the thesis which gained from simulations.

Chapter eight displays the conclusion and future work.

Chapter Two

3. CHAPTER TWO: RADAR AND RADAR CROSS SECTION (RCS)

3.1 Introduction*:*

Radar was originally developed to satisfy the needs of the military for surveillance and weapon control. Military applications have funded much of the development of its technology. However, radar has seen significant civil applications for the safe travel of aircraft, ships, and spacecraft; the remote sensing of the environment, especially the weather; and law enforcement and many other applications.

3.2 Radar:

Radar is an electromagnetic sensor for the detection and location of reflecting object. The word radar is an abbreviation for ***RAdio Detection And Ranging***. In general, radar systems use modulated waveforms and directive antennas to transmit electromagnetic energy into a specific volume in space to search for targets. Objects (targets) within a search volume will reflect portions of this energy (radar returns or echoes) back to the radar. These echoes are then processed by the radar receiver to extract target information such as range, velocity, angular position, and other target identifying characteristics[2] [3]. Radar operates by radiating electromagnetic energy and detecting the echo returned from reflecting objects (targets). The nature of the echo signal provides information about the target.

The range, or distance, to the target is found from the time it takes for the radiated energy to travel to the target and back. The angular location of the target is found with a directive antenna (one with a narrow beam width) to sense the angle of arrival of the echo signal. If the target is moving, radar can derive its track, or trajectory, and predict the future location. The shift in frequency of the received echo signal due to the Doppler Effect caused by a moving target allows radar to separate desired moving targets (such as aircraft) from undesired stationary targets (such as land and sea clutter). With sufficiently high resolution, radar can discern something about the nature of a target's size and shape. Radar resolution may be obtained in range or angle, or both. Range resolution requires large bandwidth. Angle resolution requires (electrically) large antennas.

Radar is an active device in that it carries its own transmitter and does not depend on ambient radiation, as do most optical and infrared sensors. Radar can detect relatively small targets at near or far distances and can measure their range with precision in all weather, which is its chief advantage when compared with other sensors.

The principle of radar has been applied from frequencies of a few megahertz (HF, or high-frequency region of the electromagnetic spectrum) to well beyond the optical region (laser radar).

3.3 Radar Block Diagram

The basic parts of a radar system are illustrated in the simple block diagram of Figure 2.1 [6].

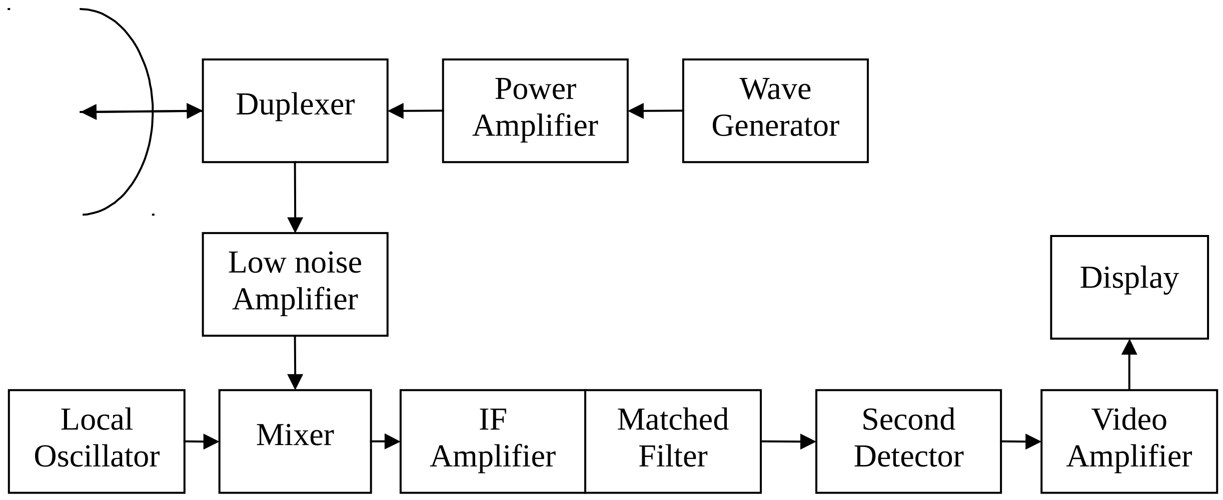


Figure 2.1 Block diagram of simple radar with power amplifier and superheterodyne

3.3.1 Transmitter:

The transmitter is shown as a power amplifier and power oscillator to generate a high power with stable waveforms, and must often operate over a wide bandwidth, with high efficiency.

3.3.2 Duplexer:

The duplexer acts as a rapid switch to protect the receiver from damage when the high power transmitter is on. On reception, with the transmitter off, the duplexer directs the weak received signal to the receiver rather than to the transmitter.

3.3.3 Antenna:

The transmitter power is radiated into space by a directive antenna which concentrates the energy into a narrow beam. Mechanically steered parabolic reflector antennas and planar phased arrays both find wide application in radar. The narrow, directive beam that is characteristic of most radar antennas not only concentrates the energy on target but also permits a measurement of the direction to the target.

3.3.4 Receiver:

The signal collected by the antenna is sent to the receiver, which is almost super heterodyne type. The receiver serves to:

- separate the desired signal from the ever-present noise and other interfering signals
- Amplify the signal sufficiently to actuate a display, such as a cathode ray tube, or to allow automatic processing by some form of digital device.

The mixer of the super heterodyne receiver translates the receiver RF signal to an intermediate frequency. The gain of the intermediate-frequency (IF) amplifier results in an increase of the receiver signal level. The IF amplifier also includes the function of the matched filter: one which maximizes the output signal-to-noise ratio. Maximizing the signal-to-noise ratio at the output of the IF maximizes the detectability of the signal. Almost all radars have a receiver which closely approximates the matched filter. The second detector in the receiver is an envelope detector which eliminates the IF carrier and passes the modulation envelope. When Doppler processing is employed, as it is in CW (continuous-wave), MTI, and pulse Doppler radars, the envelope detector is replaced by a phase detector which extracts the doppler frequency by comparison with a reference signal at the transmitted frequency. There must also be included filters for rejecting the stationary clutter and passing the doppler-frequency-shifted signals from moving targets.

3.3.5 Video amplifier:

The video amplifier raises the signal power to a level where it is convenient to display the information it contains. As long as the video bandwidth is not less than half of the IF bandwidth, there is no adverse effect on signal detectability. A threshold is established at the output of the video amplifier to allow the detection decision to be made. If the receiver output crosses the threshold, a target is said to be present. The decision may be made by an operator, or it might be done with an automatic detector without operator intervention.

3.3.6 Signal Processing:

There has not always been general agreement as to what constitutes the signal-processing portion of the radar, but it is usually considered to be the processing whose purpose is to reject undesired signals (such as clutter) and pass desired signals due to targets. It is performed prior to the threshold detector where the detection decision is made. Signal processing includes the matched filter and the doppler filters in MTI and pulse doppler radar. Pulse compression, which is performed before the detection decision is made, is sometimes considered to be signal processing, although it does not fit the definition precisely.

3.3.7 Data Processing:

This is the processing done after the detection decision has been made. Automatic tracking is the chief example of data processing. Target recognition is another example. It is best to use automatic tracking with good radar that eliminates most of the unwanted signals so that the automatic tracker only has to deal with desired target detections and not undesired clutter. When a radar cannot eliminate all nuisance echoes, a means to maintain a constant false-alarm rate (CFAR) at the input to the tracker is necessary.

3.3.8 Displays:

The display for a surveillance radar is usually a cathode-ray tube with a PPI (plan position indicator) format. A PPI is an intensity-modulated, map-like presentation that provides the target's location in polar coordinates (range and angle). Older radars presented the video output of the receiver (called raw video) directly to the display, but more modern radars generally display processed video, that is, after processing by the automatic detector or the automatic detector and tracker (ADT). These are sometimes called cleaned-up displays since the noise and background clutter is removed.

3.4 Radar Operation:

Radar operation can be summarized as, the radar signal, usually a repetitive train of short pulses, is generated by the transmitter and radiated into space by the antenna. The duplexer permits a single antenna to be time-shared for both transmission and reception. Reflecting objects (targets) intercept and reradiate a portion of the radar signal, a small amount of which is returned in the direction of the radar. The returned echo signal is collected by the radar antenna and amplified by the receiver. If the output of the radar receiver is sufficiently large, detection of a target is said to occur. Radar generally determines the location of a target in range and angle, but the echo signal also can provide information about the nature of the target. The output of the receiver may be presented on a display to an operator who makes the decision as to whether or not a target is present, or the receiver output can be processed by electronic means to automatically recognize the presence of a target and to establish a track of the target from detections made over a period of time. With automatic detection and track (ADT) the operator usually is presented with the processed target track rather than the raw radar detections [13].

3.5 Radar Classifications

Radars can be classified as ground based, airborne, or ship based radar systems. They can also be classified into numerous categories based on the specific radar characteristics, such as the frequency band, antenna type, and waveforms utilized. Another classification is concerned with the mission and/or the functionality of the radar. This includes: weather, acquisition and search, tracking, track-while-scan, fire control, early warning.

Radars are most often classified by the types of waveforms they use, or by their operating frequency. Considering the waveforms first, radars can be Continuous Wave (CW) or Pulsed Radars (PR). CW radars are those that continuously emit electromagnetic energy, and use separate transmit and receive antennas. None modulated CW radars can accurately measure target radial velocity (Doppler shift) and angular position. Target range information cannot be extracted without utilizing some form of modulation. The primary use of non modulated CW radars is in target velocity search and track, and in missile guidance. Pulsed radars use a train of pulsed waveforms (mainly with modulation). In this category, radar systems can be classified on the basis of the Pulse Repetition Frequency (PRF), as low PRF, medium PRF, and high PRF radars. Low PRF radars are primarily used for ranging where target velocity (Doppler shift) is not of interest. High PRF radars are mainly used to measure target velocity. Continuous wave as well as pulsed radars can measure both target range and radial velocity by utilizing different modulation schemes [2].

3.6 Radar Equation:

The single most useful description of the factors influencing radar performance is the radar equation which gives the range of a radar in terms of the radar characteristics. One form of this equation gives the received signal power P_r as:

$$P_r = \frac{P_t \times G_t}{4\pi R^2} \times \frac{\sigma}{4\pi R^2} \times A_e \qquad \ldots \qquad 2.1$$

The right-hand side has been written as the product of three factors to represent the physical processes taking place. The first factor is the power density at a distance *R* meters from a radar that radiates a power of P_t watts from an antenna of gain G_t. The numerator of the second factor is the target cross section σ in square meters. The denominator accounts for the divergence on the return path of the electromagnetic radiation with range and is the same as the denominator of the first factor, which accounts for the divergence on the outward path. The product of the first two terms represents the power per square meter returned to the radar. The antenna of effective aperture area A_e intercepts a portion of this power in an amount given by the product of the three factors. If the maximum radar range R_{max} is defined as that which results in the received power P_r being equal to the receiver minimum detectable signal S_{min}, the radar equation may be written:

$$R_{\max}^4 = \frac{P_t G_t A_e \sigma}{(4\pi)^2 S_{\min}} \quad \ldots \quad 2.2$$

When the same antenna is used for both transmitting and receiving, the transmitting gain G_t and the effective receiving aperture A_e are related by $G_t = 4\pi A_e/\lambda^2$, where λ is the wavelength of the radar electromagnetic energy. Substituting into Eq. (2.2) gives two other forms of the radar equation:

$$R_{mx}^4 = \frac{P_t G_t^2 \lambda^2 \sigma}{(4\pi)^3 S_{\min}} \quad \ldots \quad 2.3$$

The minimum detectable signal S_{min}, which appears in the radar equation, is a statistical quantity and must be described in terms of the probability of detection and the probability of a false alarm. This signal to be reliably detected it must be larger than noise (generally by 10 to 20 dB) at the point in the receiver where the detection decision is made. The minimum detectable signal can be expressed as the signal-to-noise ratio *(S/N)* required for reliable detection times the receiver noise. The receiver noise is expressed relative to the thermal noise that would be produced by an ideal receiver. The thermal noise is equal to *kTB,* where *k* is Boltzmann's constant, *T* is the temperature, and *B* is the receiver bandwidth. The receiver noise is the thermal noise multiplied by the factor F_n, the receiver noise figure. The receiver noise figure is measured relative to a reference temperature T_0 = 290 K (approximately room temperature), and the factor kT_0 becomes 4x 10^{-21} W/Hz. The minimum detectable signal in the radar equation can be written [13]:

$$S_{\min} = kT_0 B F_n \frac{S}{N} \quad \ldots \quad 2.4$$

3.7 Radar Cross Section (RCS):

Radar detects or tracks a target, and sometimes can identify it, only because there is an echo signal. It is therefore critical in the design and operation of radars to be able to quantify or otherwise describe the echo, especially in terms of such target characteristics as size, shape, and orientation. For that purpose the target is ascribed an effective area called the *radar cross section (RCS).* It is the projected area of a metal sphere which would return the same echo signal as the target had the sphere been substituted for the target [4].

3.8 RCS Definition

Electromagnetic waves, with any specified polarization, are normally diffracted or scattered in all directions when incident on a target. These scattered waves are broken down into two parts. The first part is made of waves that have the same polarization as the receiving antenna. The other portion of the scattered waves will have a different polarization to which the receiving antenna does not respond. The two polarizations are orthogonal and are referred to as the Principle Polarization (PP) and Orthogonal Polarization (OP), respectively. The intensity of the *backscattered* energy that has the same polarization as the radar's receiving antenna is used to define the target radar cross section (RCS).

Radar cross section (RCS) is the measure of a target's ability to reflect radar signals in the direction of the radar receiver, i.e. it is a measure of the ratio of backscatter power per unit solid angle in the direction of the radar (from the target) to the power density that is intercepted by the target. RCS, denoted by the Greek letter σ and measured in m², is defined as [5]:

$$\sigma = 4\pi \frac{P_S}{P_i} \qquad \ldots \qquad 2.5$$

Where P_i: power density, or intensity, of a plane wave striking the target. P_s: power per unit solid angle reflected by the target.

3.9 RCS is a function of:

- Position of transmitter/receiver relative to target.
- Target geometry and material composition.
- Angular orientation of target relative to transmitter/receiver.
- Frequency or wavelength.
- Antenna polarization.

Assume the power density of a wave incident on a target located at range R away from the radar is P_{Di} . The amount of reflected power from the target is [5]:

$$P_r = \sigma P_{Di} \qquad \ldots \qquad 2.6$$

σ denotes the target cross section. Define P_{Dr} as the power density of the scattered waves at the receiving antenna. It follows that:

$$P_{Dr} = \frac{P_r}{4\pi R^2} \qquad \ldots \qquad 2.7$$

Equating Eqs. (2.6) and (2.7) yields:

$$\sigma = 4\pi R^2 \left(\frac{P_{Dr}}{P_{Di}} \right) \qquad \ldots \qquad 2.8$$

and in order to ensure that the radar receiving antenna is in the far field (i.e., scattered waves received by the antenna are planar), Eq. (2.8) is modified:

$$\sigma = 4\pi R^2 \lim_{R \to \infty} \left(\frac{P_{Dr}}{P_{Di}} \right) \qquad \ldots \qquad 2.9$$

The RCS defined by Eq. (2.9) is often referred to as either the monostatic RCS, the backscattered RCS, or simply target RCS. The backscattered RCS is measured from all waves scattered in the direction of the radar and has the same polarization as the receiving antenna. It represents a portion of the total scattered target RCS σ_t, where $\sigma_t > \sigma$. Assuming spherical coordinate system defined by (ρ,θ,φ), then at range ρ the target scattered cross section is a function of (θ,φ). Let the angles (θ_i,φ_i) define the direction of propagation of the incident waves. Also, let the angles (θ_s,φ_s) define the direction of propagation of the scattered waves. The special case, when $\theta_s = \theta_i$ and $\varphi_s = \varphi_i$, defines the monostatic RCS. The RCS measured by the radar at angles $\theta_s \neq \theta_i$ and $\varphi_s \neq \varphi_i$ is called the bistatic RCS. The total target scattered RCS is given by:

$$\sigma_t = \frac{1}{4\pi} \int_{\varphi_S=0}^{2\pi} \int_{\theta_S=0}^{\pi} \sigma(\theta_S, \varphi_S) \sin\theta_S \; d\theta \, d\varphi \qquad \ldots \qquad 2.10$$

The amount of backscattered waves from a target is proportional to the ratio of the target extent (size) to the wavelength, λ, of the incident waves. In fact, a radar will not be able to detect targets much smaller than its operating wavelength. For example, if weather radars use L-band frequency, rain drops become nearly invisible to the radar since they are much smaller than the wavelength. RCS measurements in the frequency region, where the target extent and the wavelength are comparable, are referred to as the Rayleigh region. Alternatively, the frequency region where the target extent is much larger than the radar operating wavelength is referred to as the optical region. In practice, the majority of radar applications fall within the optical region. Figures 2.2 and 2.3 show that RCS does not equal geometric area. For a sphere, the RCS, σ = πr2, where r is the radius of the sphere. The RCS of a sphere is independent of frequency if operating at sufficiently high frequencies where λ<<Range, and λ << radius (r). Experimentally, radar return reflected from a target is compared to the Radar return reflected from a sphere which has a frontal or projected area of one square meter. Using the spherical shape aids in field or laboratory measurements since orientation or positioning of the sphere will not affect radar reflection intensity measurements as a flat plate would. If calibrated, other sources (cylinder, flat plate, or corner reflector, etc.) could be used for comparative measurements [1],[5].

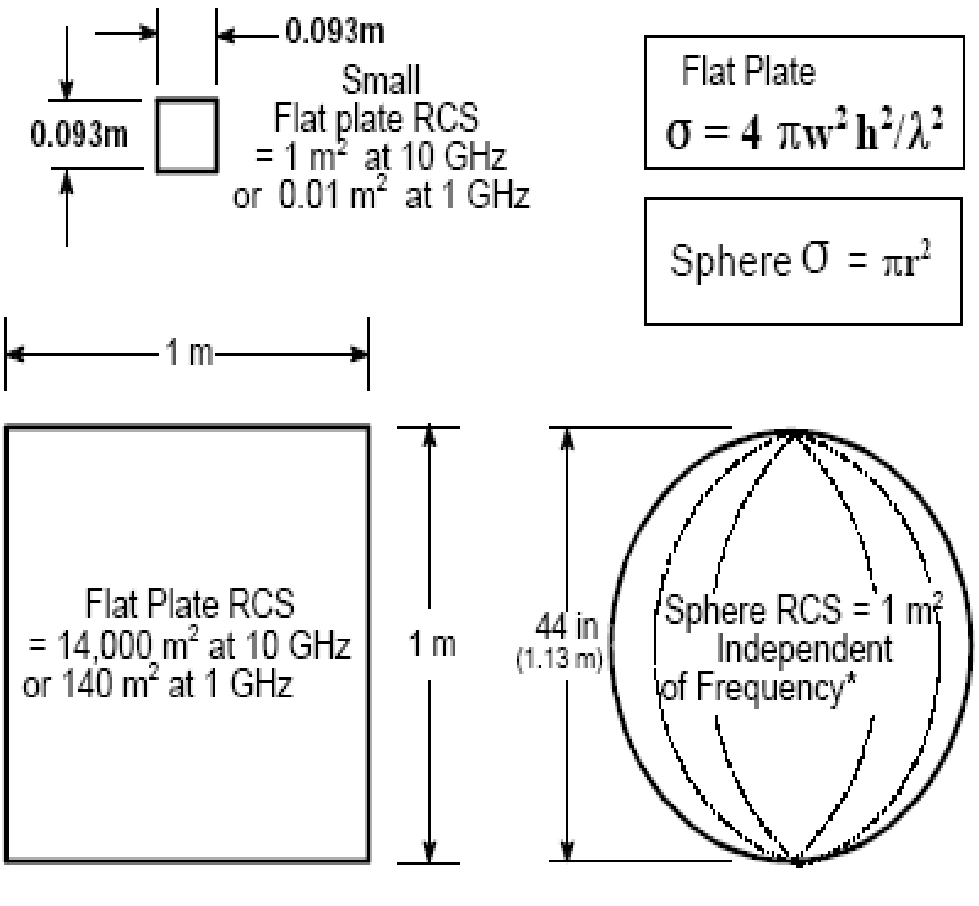


Figure 2.2: RCS versus Physical Geometry

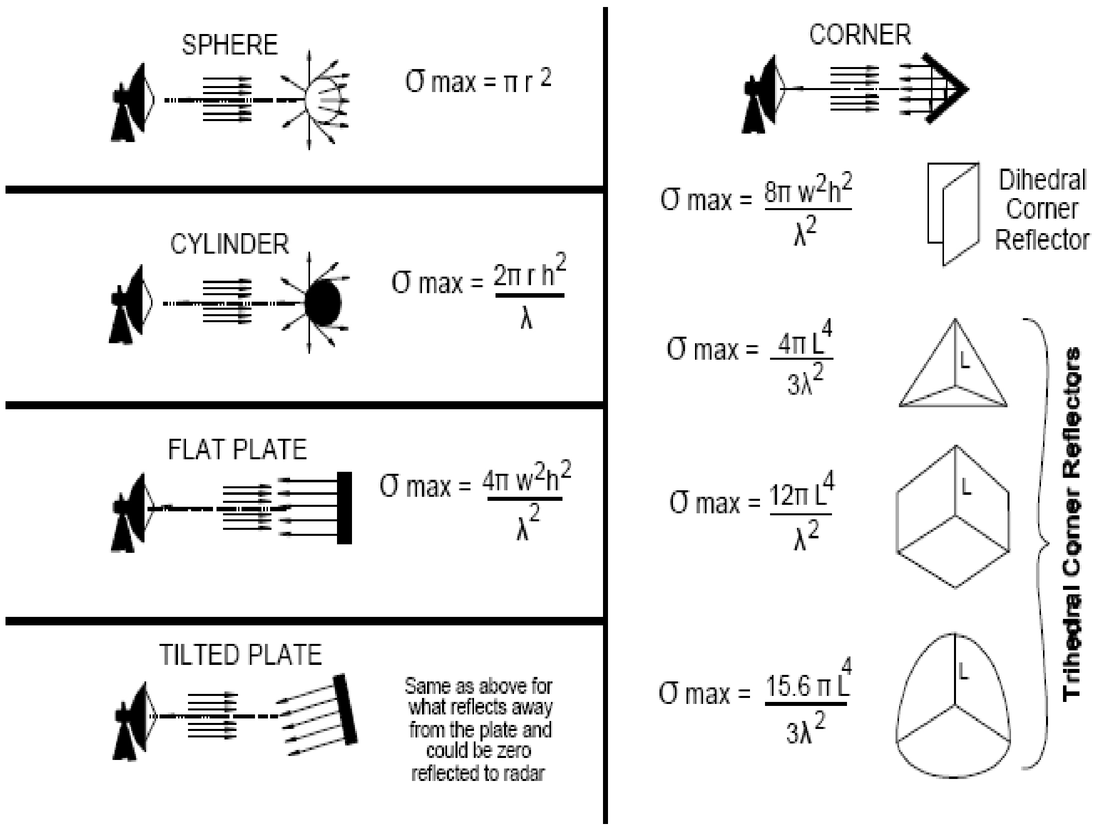

SPHERE
$\sigma \max = \pi r^2$
CYLINDER
$\sigma \max = \frac{2\pi r h^2}{\lambda}$
FLAT PLATE
$\sigma \max = \frac{4\pi w^2 h^2}{\lambda^2}$
TILTED PLATE
Same as above for what reflects away from the plate and could be zero reflected to radar
CORNER
$\sigma \max = \frac{8\pi w^2 h^2}{\lambda^2}$
Dihedral Corner Reflector
$\sigma \max = \frac{4\pi L^4}{3\lambda^2}$
$\sigma \max = \frac{12\pi L^4}{\lambda^2}$
$\sigma \max = \frac{15.6\, \pi L^4}{3\lambda^2}$
L
L
L
Trihedral Corner Reflectors

Figure 2.3: Backscatter from Shapes

Chapter Three

4. CHAPTER THREE: RADAR CROSS SECTION PREDICTION TECHNIQUES

4.1 Introduction:

The challenges in the area of RCS computation or measurement of large targets, such as, aircrafts, tankers, and tanks of modern design are obvious. The Rayleigh region is of no interest in terms of RCS modeling since every target acts like a point scatterer and the RCS contribution is almost negligible. High frequency asymptotic (HFA) techniques, such as *Geometric Optics* (GO), *Physical Optics* (PO), *Geometric Theory of Diffraction* (GTD) and *Physical Theory of Diffraction* (PTD) have been successfully applied to RCS modeling in optical region. Pure numerical methods, such as *Fast Multipole Method* (FMM), *Finite-Difference Time Domain* (FDTD) and *Method of Moments* (MoM), techniques are powerful RCS simulators applicable in resonance region where the target contributes its RCS as a whole [1].

4.2 Geometric Optics (GO):

The theory of geometric optics was used for many years by astronomers and lens makers in designing and building optical systems. It accounts not only for the way light rays are reflected from smooth surfaces, but for the change in the angle of the transmitted ray when light passes from one medium to another. This was of considerable importance in the design of lenses, because the bending of the transmitted ray depends not only on the wavelength of the light, but on the refractive index of the lens materials, which itself varies with the wavelength. When used to predict the scattering of radar waves from objects of practical interest, the body surfaces are usually assumed to be perfectly conducting, although this is not necessarily a restriction.

Geometric optics is a ray-tracing procedure in which the wavelength is allowed to become infinitesimally small. This being the case, energy propagates along slender tubes according to the formula.

$$u = Pe^{ikS} \qquad \ldots \qquad 3.1$$

where the amplitude *P* represents the intensity of either the magnetic or electric field transverse to the direction of propagation. The amplitude *P* and the phase factor *S* are both functions of position in space and may be complex numbers. Propagation is in the direction given by AS, hence surfaces of constant *S* are surfaces of constant phase. Eq.3.1 is a solution of the wave equation in the limit of vanishing wavelength, and because the field components are transverse to the direction of propagation, the solution is not valid near discontinuities such as edges.

When a ray strikes a smooth flat surface separating two media of different refractive index, part of the energy is reflected and part is transmitted across the boundary into the second medium. When the electromagnetic boundary conditions are invoked, we find that the transmitted ray propagates in a direction different from that of the incident ray (refraction), and the angle of the reflected ray, as measured from the surface normal, is equal to the angle of the incident ray. The effect is known as Snell's law, and the amplitude and phase of both the reflected and transmitted rays can be calculated.

The reflection coefficient for a perfectly conducting surface is - 1, implying a 180° phase shift and no reduction of the intensity of the reflected wave. This is true only at the point of reflection (the specular point), and if the ray is due to a point source of energy some finite distance from the surface, the reflected ray decays in intensity as it travels away from the specular point. The decay in intensity is due, of course, to the spreading of energy, and the effect can be accentuated if the reflecting surface is curved. It is also possible for the energy in a ray bundle to increase, which is precisely the effect desired of focused mirrors. In this case, a caustic may be formed when an infinity of rays converge at a point or a line.

The decay or increase in energy can be calculated by invoking the principle of conservation of energy along a ray tube. By demanding that all the energy entering the tube at one end be transmitted to the other, we will find that the ratio of the power density at the output to that at the input is

$$\frac{|A(s)|^2}{|A(o)|^2} = \frac{p_1 p_2}{(s+p_1)(s+p_2)} \qquad \ldots \qquad 3.2$$

The curvature of the reflected wavefront is given by p_1 and p_2 and that of the reflecting surface is a_1 and a_2 In general, the planes containing a_1 and a_2 are neither parallel nor perpendicular to the plane of incidence. The radii *p1* and *p2* are measured at the caustics (here below the actual surface) formed by extending the reflected ray tube backward until the rays meet. where *A(o)* and *A(s)* are the field intensities at the input and output, respectively, *s* is the distance along the tube between the two ends, and p_1 and p_2 are the principal radii of curvature of the wavefront at the output of the tube.

If the body is illuminated by a spherical wave due to a point source located a finite distance away, we can find the image of the source in the surface by extending the reflected ray tube backward until the sides of the tube intersect. Because of the differences in the surface radii of curvature, the two sides of the tube will intersect along a line that does not necessarily coincide with the intersection of the top and bottom of the tube, hence the image of a point source in a curved surface does not generally yield another point. The effect is called *astigmatism,* and the result is a blurred image.

The radii of curvature of the reflected wavefront at the specular point can be related to the radii of curvature of the incident wavefront and the radii of curvature of the body there. The relationship is not a simple one, and it involves not only the angle of arrival of the incident ray, but also the angle by which the principal planes of the body curvature are rotated out of the plane of Incidence. This relationship may be inserted in Eq. 3.2, and the distance *s* may be measured from the specular point to the point of observation. If the direction of observation is now taken to be back toward the source, *s* becomes the distance *R*. If *R* is forced to infinity, as required by the formula, the angular dependence on the local angle of arrival of the incident ray disappears. In addition, the angular rotation of the principal planes of the body radii of curvature out of the plane of incidence also drops out of the expression. The results of the calculation are simply [1]:

$$\sigma = \pi a_1 a_2 \qquad \ldots \qquad 3.3$$

where a_1 and a_2 are the principal radii of curvature of the body at the specular point.

4.3 Physical Optics (PO):

The theory of physical optics overcomes the catastrophe of the infinities of flat and singly curved surfaces by approximating the induced surface fields and integrating them to obtain the scattered field. Because the induced fields remain finite, the scattered fields are finite as well. The beginning point is the Stratton-Chu integral equations. These expressions hold for a closed scattering surface, and Stratton has demonstrated that, if the surface is not closed, additional terms must be added (line integrals around the edge bounding the open surface) to account for the edge discontinuity. Two simplifications can be made immediately in the integrals. One is the far-field approximation, in which the distance *R* from an origin in or near the obstacle to the far-field observation point is much larger than any obstacle dimension. This allows the gradient of the Green's function to be well approximated by [1]:

$$\nabla\psi \approx ik\hat{s}\Psi \qquad \ldots \qquad 3.4$$

where $\hat{s}$ is a unit vector aligned along the scattering direction. Under far-field conditions, the line integrals can be represented as surface integrals, and when combined with the other terms, another simplification results. It will be found that there can be no component of the surface field distribution along the scattering direction, whence the Stratton-Chu integrals can be written as:

$$\overline{E}_s = ik\psi_0 \int_s \hat{s} \times \left[\hat{n} \times \overline{E} - Z_0 \hat{s} \times \left(\hat{n} \times \overline{H}\right)\right] e^{ik\bar{r}\left(\hat{i}-\hat{s}\right)}\, ds \qquad \ldots \qquad 3.4$$

$$\overline{H}_s = ik\psi_0 \int_s \hat{s} \times \left[\hat{n} \times \overline{E} + Y_0 \hat{s} \times \left(\hat{n} \times \overline{E}\right)\right] e^{ik\bar{r}\left(\hat{i}-\hat{s}\right)}\, ds \qquad \ldots \qquad 3.5$$

where $\hat{\imath}$ is a unit vector along the direction of incidence, $\hat{r}$ is now the position vector from the local origin to the surface patch dS, $Y_0 = 1/Z_0$ is the admittance of free space, and $\Psi = e^{(ikR)/4\pi R}$ is the farfield Green's function. Note that the scattered fields are represented in terms of the tangential components of the total fields on the surface; hence the desired scattered fields appear on both sides of the equation. Either of the two equations can be used to calculate the far scattered field because of the relationship:

$$\overline{H}_s = Y_0 \times \overline{E}_s \qquad \ldots \qquad 3.6$$

We can approximate the total fields within the integrals by making the *tangent plane approximation.* That is, we assign the surface fields the values they would have had if the body had been perfectly smooth and flat at the surface patch of integration dS. This approximation can be made for any body material, but we shall assume the body to be perfectly conducting. In this case, the tangential components of the total fields are:

$$\begin{aligned} \hat{n} \times \overline{E} &= 0 \\ \hat{n} \times \overline{H} &= 2\hat{n} \times \overline{H}_i \end{aligned} \qquad \ldots \qquad 3.7$$

where $\overline{H}_i$ is the incident magnetic field strength at the surface path. If the incident wave propagates in a direction given by the unit vector $\hat{\imath}$, with a magnetic intensity H_0 and a magnetic polarization along the unit vector $\hat{h}$, (3.4) becomes the physical optics integral:

$$\overline{E}_s = -i2kZ_0 H_0 \psi_0 \int_s [\hat{s} \times (\hat{n} \times \hat{h}_i)] e^{ik\hat{r}(\hat{\imath}-\hat{s})}\, ds \qquad \ldots \qquad 3.8$$

where the surface *S* is now the illuminated portion of the body. In other words, the tangential fields on shaded portions of the body are assumed to be precisely zero. Despite the fact that implicitly contains phase information, we would like to eliminate the range dependence in the phase factor as well as in the amplitude we can fined [1]:

$$\sqrt{\sigma} = \lim_{R\to\infty} 2\sqrt{\pi}\, R \frac{\hat{E}_s . \hat{e}_r}{E_0} e^{ikR} \qquad \ldots \qquad 3.9$$

When Eq. 3.8 is substituted into Eq. 3.9, we have the physical optics expression for the square root of the RCS:

$$\sqrt{\sigma} = -i \frac{k}{\sqrt{\pi}} \int_s \hat{n} . \hat{e} \times \hat{h}_i\, e^{ik\hat{r}(\hat{\imath}-\hat{s})}\, ds \qquad \ldots \qquad 3.10$$

4.4 Geometric Theory of Diffraction (GTD):

Keller was well aware of the wide-angle failure of physical optics and introduced his classic geometrical theory of diffraction as a way of computing scattered fields well away from the specular directions. It was shown earlier that when a ray or electromagnetic wave impinges on a flat infinite surface, part of the wave or ray is reflected and part is transmitted through the surface. Whether the surface is dielectric or perfectly conducting, the reflected ray can propagate in only one direction in space; that direction is in the plane of incidence containing the incident ray and the surface normal, and it subtends the same angle with respect to the surface normal as does the incident ray. This unique specular direction of the reflected ray is a consequence of the doubly infinite size of the surface [1].

If the doubly infinite surface is now halved, thereby generating an edge, we may think of the reflected (or scattered) fields as arising from two sources, one being a surface contribution, as before, plus an edge contribution. Ignoring for the moment the surface contribution, let us consider the edge and the components of an incident ray parallel and perpendicular to it. We would expect reflected ray components along the edge to be constrained to a unique direction because that dimension of the edge is infinite. However, we would expect the transverse components of reflected rays to propagate in all directions perpendicular to the edge because the transverse dimension of the edge is zero. Thus, in contrast to the single, unique direction taken by the ray reflected by a surface, an edge-reflected ray can lie anywhere along a forward cone whose half-angle is precisely that subtended by the edge and the incident ray.

In Sommerfeld's two-dimensional problem, the incident ray impinges on the edge at right angles, and Sommerfeld represented the diffracted rays as a spectrum of plane waves. His solution amounts to finding the coefficients of each of the elementary plane waves diffracted by the edge. Hence, for any scattering direction, the strength of the diffracted ray can be calculated. The Sommerfeld two-dimensional solution for diffracted rays can be applied to Keller's three-dimensional problem by adjusting Sommerfeld's propagation constants to match the transverse propagation components of Keller's problem. The direction-dependent amplitudes of Sommerfeld's solution then make it possible to estimate the amplitude and phase of a diffracted ray anywhere on the Keller cone. This yielded the key ingredient of the theory: the *diffraction coefficients,* which depend on the polarization of the incident ray. The other ingredients of Keller's GTD include the decay in field intensity away from the diffracting edge and the phase of the ray along its propagation path. The intensity of the diffracted ray thus has the form [1]:

$$u = \frac{D e^{iks}}{\left[s \left(1 + {}^{s}\!/_{p_1} \right) \right]^{1/2}} A e^{ik\Psi} \qquad \ldots \qquad 3.11$$

where D is a diffraction coefficient depending on the polarization and angle of arrival of the incident ray and the direction of the scattered ray, s is the distance along the ray from the edge element to a far-field observation point, A and Ψ are the amplitude and phase of the incident ray, respectively, and p_i is the distance from the edge element to a caustic of the diffracted ray. When Eq. 3.11 is applied to the components of the far diffracted electric field, the diffracted field can be expressed as:

$$\bar{E}_d = -\frac{\Gamma e^{iks}}{\sin^3\beta}\left[(\hat{t}.\bar{E}_i)(X-Y)\hat{s}\times(\hat{s}\times\hat{t}) + Z_0(\hat{t}.\bar{H}_i)(X+Y)\hat{s}\times\hat{t}\right] \quad \ldots \quad 3.12$$

where Γ is a divergence factor accounting for the nature of the edge excitation (plane wave, spherical wave, …) and the spreading of energy away from the edge, and β is the angle subtended by the edge and the incident ray. In this expression, $\hat{t}$ is a unit vector aligned along the edge, and the diffraction coefficients are given by:

$$X = \frac{\left(1/n\right)\sin\left(\pi/n\right)}{\cos\left(\pi/n\right) - \cos\left[\Psi_s - \Psi i/n\right]} \quad \ldots \quad 3.13$$

$$Y = \frac{\left(1/n\right)\sin\left(\pi/n\right)}{\cos\left(\pi/n\right) - \cos\left[\Psi_s + \Psi i/n\right]} \quad \ldots \quad 3.14$$

where *n* is the exterior wedge angle normalized with respect to π, and Ψ_i and Ψ_s are the angles of the transverse components of the incident and diffracted directions with respect to one of the surfaces meeting at the edge. It should be noted that (3.12) and (3.13) are the form of the diffraction coefficients as suggested by Ufimtsev, although Ufimtsev did not specifically label them *X* and *Y.* We use the difference in the coefficients for those components of the incident magnetic polarizations along the edge. To obtain the RCS due to an edge, we need only substitute the field of Eq. 3.12 into Eq. 3.9.

4.5 Physical Theory of Diffraction (PTD):

Like Keller, Ufimtsev sought a more accurate representation of the scattered fields than yielded by physical optics and he relied on the canonical solution of the scattering by a wedge for his diffraction coefficients, but, unlike Keller, he retained in his solution the approximate physical optics result and sought instead a correction by which to improve the physical optics approximation. He therefore represented the scattered field as the sum of the physical optics contribution and an edge contribution, using the exact solution of the two-dimensional wedge problem to extract the latter. That is, if we have in hand the exact solution and subtracts from it the physical optics (surface) contribution, what remains must be the contribution from the edge itself, there being no other scattering features present except the surface and the edge [1].

In developing his PTD, Ufimtsev claims to have considered the "non-uniform" induced edge currents in addition to the "uniform" induced surface currents of physical optics, but nowhere in his work will the reader find an explicit representation of the edge currents. He considered instead the scattered fields, not the surface currents, of the exact solution for a wedge.

Nevertheless, we are seldom interested in the surface field, except as a means by which to compute the scattered field, hence his approach is defensible.

As in most two-dimensional problems, Ufimtsev recognized two distinct cases depending on whether the incident field is polarized parallel or perpendicular to the edge. Arbitrary polarizations, of course, can be handled as linear combinations of the two cases. He represented the total fields (incident plus scattered fields)

$$E_z = E_{oz}\left[u\left(r,\Psi_s - \Psi_i\right) - u\left(r,\Psi_s + \Psi_i\right)\right] \quad \ldots \quad 3.15$$

$$H_z = H_{oz}\left[u\left(r,\Psi_s - \Psi_i\right) + u\left(r,\Psi_s + \Psi_i\right)\right] \quad \ldots \quad 3.16$$

Where

$$u\left(r,\Psi\right) = \frac{1}{2\alpha}\int_c \frac{e^{-ikr\cos\beta}}{1-\exp[i\pi(\beta+\psi)/\alpha]}d\beta \quad \ldots \quad 3.17$$

where *r* is the distance from the edge to the point of observation, Ψ_s is the angular coordinate of that point above one face of the wedge, Ψ_s is the direction of arrival of the incident wave, similarly measured, *α* is the external wedge angle, and *C* is the Sommerfeld contour in the complex plane [1].

4.6 Fast Multipole Method (FFM):

The fast multipole method (FMM) is a mathematical technique that was developed to speed up the calculation of long-ranged forces in the n-body problem. It does this by expanding the system Green's function using a multipole expansion, which allows one to group sources that lie close together and treat them as if they are a single source.

The FMM has also been applied in accelerating the iterative solver in the method of moments (MOM) as applied to computational electromagnetics problems. The FMM was first introduced in this manner by Greengard and Rokhlin and is based on the multipole expansion of the vector Helmholtz equation. By treating the interactions between far-away basis functions using the FMM, the corresponding matrix elements do not need to be explicitly stored, resulting in a significant reduction in required memory. If the FMM is then applied in a hierarchical manner, it can improve the complexity of matrix-vector products in an iterative solver from O(N^2) to O(N). This has expanded the area of applicability of the MOM to far greater problems than were previously possible.

4.7 Finite-Difference Time Domain (FDTD):

Finite-difference time-domain (FDTD) is a numerical analysis technique used for modeling computational electrodynamics (finding approximate solutions to the associated system of differential equations). Since it is a time-domain method, FDTD solutions can cover a wide frequency range with a single simulation run, and treat nonlinear material properties in a natural way. The FDTD method belongs in the general class of grid-based differential numerical modeling methods (finite difference methods). The time-dependent Maxwell's equations (in partial differential form) are discretized using central-difference approximations to the space and time partial derivatives. The resulting finite-difference equations are solved in either software or hardware in a leapfrog manner: the electric field vector components in a volume of space are solved at a given instant in time; then the magnetic field vector components in the same spatial volume are solved at the next instant in time; and the process is repeated over and over again until the desired transient or steady-state electromagnetic field behavior is fully evolved [1].

4.8 Method of Moments (MoM):

The Method of moments (MoM) is an integral based algorithm used for estimation of RCS of a Perfectly Electric Conductor (PEC) by using the Electric Field Integral Equation (EFIE). Consider an arbitrarily shaped perfect electric conductor shown in Figure 3.1, the PEC surface has divided into triangular sub-domains using surface batch model. The simultaneous equations have been generated over the sub-domains and added together to form a global matrix equation. The solution of the matrix gives electric current distribution in the surface of the PEC objects.

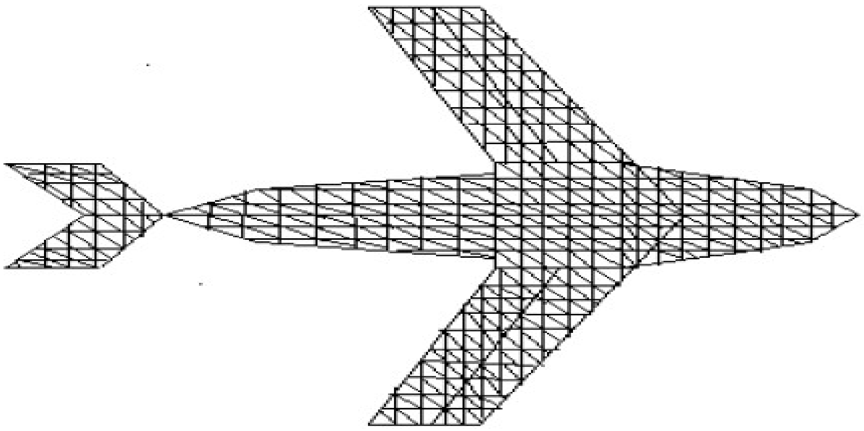


Figure 3.1: Aircraft surface modeled by triangular patches

Let S denotes the surface of an open or closed perfectly conducting object with unit normal $\hat{n}$. An electric field, defined to be the field due to an impressed source in the absence of the object, is incident on and induces surface currents J on S.

The scattering electric field E_s can be computed from the surface current by [7]

$$1\,E_s = -j\omega A - V\Phi \qquad \ldots \qquad (3.18)$$

With the magnetic vector potential defined as,

$$A(r) = \frac{\mu}{4\Pi}\int_S J\,\frac{e^{-jkR}}{R}\,dS' \qquad \ldots \qquad (3.19)$$

And the electric scalar potential as,

$$\Phi(r) = \frac{1}{4\Pi\varepsilon}\int_S \sigma\frac{e^{-jkR}}{R}\,dS' \qquad \ldots \qquad (3.20)$$

Where, k is wave number and R = |r – r`| is the distance between an arbitrarily located observation point r and a source point r` on S. Both r and r` are defined with respect to a global coordinate origin O. The surface charge density σ is related to surface divergence of J through the equation of continuity,

$$\nabla_s . J = -j\omega\sigma \qquad \ldots \qquad (3.21)$$

An integro-differential equation for J has been derived by enforcing the boundary condition,

$$\begin{aligned}
&\hat{n}\times\left(E^s + E^i\right) = 0 \text{ On S, obtain}\\
&\hat{n}\times E^s = -\hat{n}\times E^i = \hat{n}\times(-j\omega A - \nabla\Phi),\ r\ on\ S \qquad \ldots \qquad (3.22)\\
&-E^i_{\tan} = (-j\omega A - \nabla\Phi)_{\tan},\ r\ on\ S
\end{aligned}$$

The above equation is the so-called electric field equation [8].

The radiated magnetic and electric fields of an infinitesimal dipole located at the origin is expressed at a point r in terms of vector notations as,

$$\begin{aligned}
&H(r) = \frac{jk}{4\Pi}(m\times r)Ce^{-jkr}\ ,\ C = \frac{1}{r^2}\left[1+\frac{1}{jkr}\right]\\
&E(r) = \frac{\eta}{4\Pi}\left(\left(M - m\right)\left[\frac{jk}{r} + C\right] + 2MC\right)e^{-jkr}\\
&where, \qquad \ldots \qquad (3.23)\\
&M = \frac{(r.m)r}{r^2} \quad r = |r|\\
&\eta = \sqrt{\mu/\varepsilon} = 377\Omega\ \ is\ the\ \ free\ space\ impedance
\end{aligned}$$

The total electric and magnetic field at a point r are obtained as a sum over all edge elements.

$$\begin{aligned}
&E(r) = \sum_{m=1}^{M} E_m - \frac{1}{2}\left(r_m^{c+} + r_m^{c-}\right)\\
&H(r) = \sum_{m=1}^{M} H_m - \frac{1}{2}\left(r_m^{c+} + r_m^{c-}\right)
\end{aligned} \qquad \ldots \qquad (3.24)$$

From the incident electric and the estimated electric fields, the Radar Cross Section can be calculated as [9].

Chapter Four

5. CHAPTER FOUR: RADAR CROSS SECTION REDUCTION (RCSR)

5.1 Introduction:

It has been apparent for some time that the development of increasingly sophisticated detection systems threatens to reduce the mission effectiveness of many types of weapons platforms. Strong attention is now being given to methods of increasing survivability by reducing detectability. Because the specific configuration of any platform is determined by many factors involved in its mission, the final design represents a compromise between conflicting requirements. The purpose of this chapter is to survey some of the *radar cross section reduction* (RCSR) options available.

5.2 The four method of RCSR

There are only four basic techniques for reducing radar cross section:

- Shaping.
- Radar absorbing materials.
- Passive cancellation.
- Active cancellation.

Each method has advantages and disadvantages. The remainder of this section provides a brief overview of the four techniques. The two most practical and most often applied RCSR techniques are listed first, shaping and radar absorbing materials. In current RCS designs, shaping techniques are first employed to create a planform design with inherently low RCS in the primary threat sectors. Radar absorbing materials are then used to treat areas whose shape could not be optimized or to reduce the effects of creeping waves or traveling waves on the signature [1].

5.2.1 Shaping

Target shaping refers to the special selection of target surface shapes to minimize the amount of energy scattered back to the radar, typically at the cost of redirection of the scattered energy from the direction of interest to another region of little or no interest. Intake cavities make it impossible to reduce the radar reflections from the object inside of the cavity as shown in Figure 4.1.

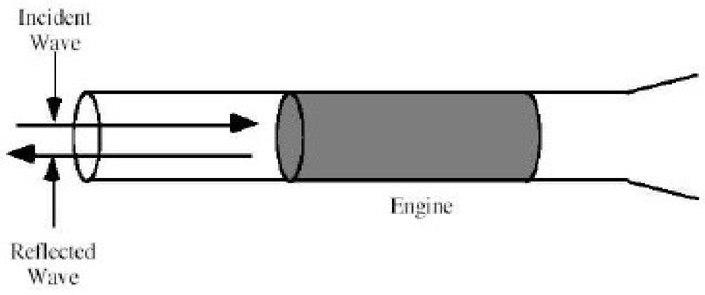


Figure 4.1: Straight Cavity

With respect to the engine, these cavities make highly reflective turbine blades visible to radar which causes a significant increase in the RCS of the aircraft. Often, engine intakes incorporate an S-shaped duct as seen in Figure 4.2 so that the turbine blades are not visible to radar.

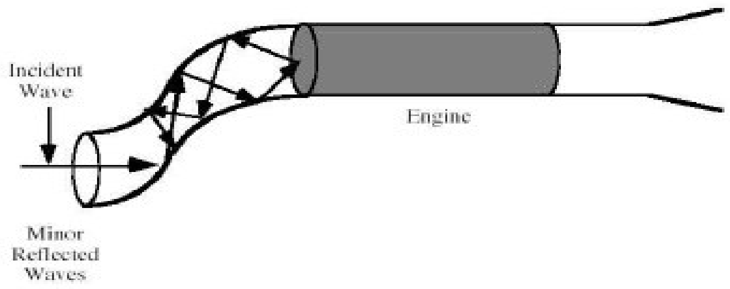


Figure 4.2: S-Shaped Cavity

The objective of shaping is to orient the target surfaces and edges to deflect the scattered energy in directions away from the radar. This cannot be done for all viewing angles within the entire sphere of solid angles because there will always be viewing angles at which surfaces are seen at normal incidence, and there the echoes will be high. The success of shaping depends on the existence of angular sectors over which low radar cross section is less important than over others.

Typically, a forward cone of angles is of primary interest for RCSR, hence, we normally want to "shift" large cross sections out of the forward sector and toward broadside. This can be accomplished by sweeping airfoils back at sharper angles, for example. The forward sector includes the elevation plane as well as the azimuth plane, and if a target is hardly ever seen from above, echo sources, such as engine intakes, can be placed on the top side of the target where they may be hidden by the forward portion of the body when viewed from below. Similarly, for a low flyer whose major threats might be look-down radars, engine inlets might instead be placed on the underside of the fuselage.

For more "boxy" structures, such as ships and ground vehicles, internal dihedral and trihedral corner and "top hat" (right circular cylinder with its axis perpendicular to a flat plate) returns are the major RCS contributors, and those can be avoided by bringing intersecting surfaces together at acute or obtuse angles. Because of the presence of the sea surface, vertical bulkheads and masts on ships, in particular, form efficient corners, and the effect can be reduced by tilting the bulkheads away from the vertical. However, this is virtually impossible to do with existing vessels and therefore is a real consideration only in new designs. Even in the case of a new design, the amount of bulkhead tilt is a trade-off between RCSR performance and cost.

Shaping Examples:

All modern military air and surface platform incorporate some aspect of shaping. Obvious examples are F-117A fighter and B-2 bomber, shown in figure 4.3 and 4.4 respectively. In plan view, it is clear that edges lie at a few common angles. Many stealth aircraft shapes are aerodynamic unstable; they achieve their high performance and stability through computer control. Shaping principles have been applied to the ships as well [1].

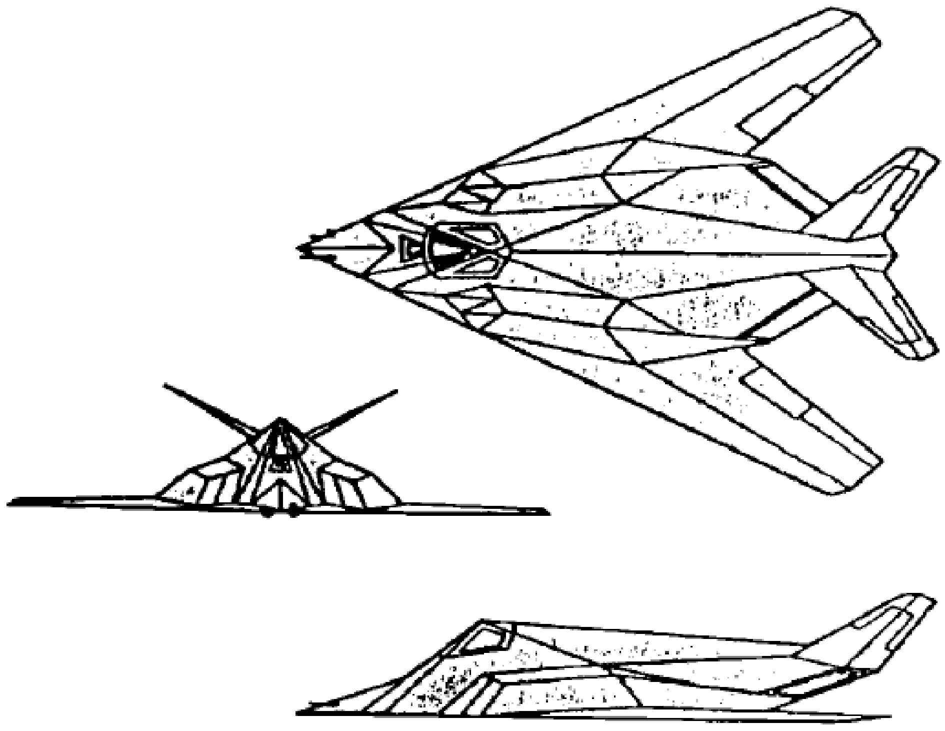


Figure 4.3: Lockheed Martin F-117

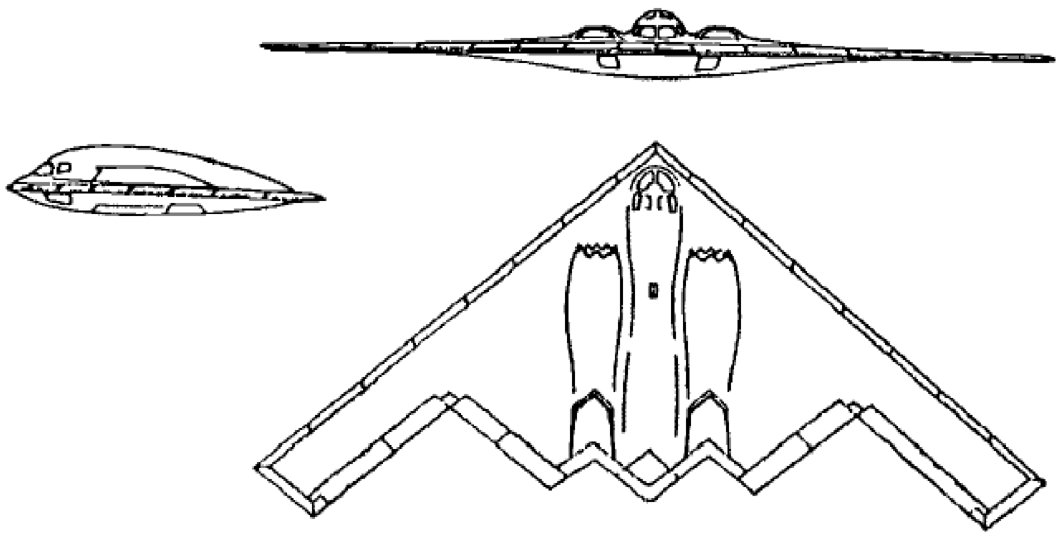


Figure 4.4**:** Northrop Grumman B-2

5.2.2 Radar Absorbing Materials:

As the name implies, radar absorbing materials reduce the energy reflected back to the radar by means of absorption. Radar energy is absorbed through one or more of several loss mechanisms, which may involve the dielectric or magnetic properties of the material. The loss is actually the conversion of radio frequency energy into heat, and although most absorbers do not dissipate enough energy to become even detectably warm when illuminated by radar, this is nevertheless the mechanism by which they operate.

When radar waves hit the RAM coating, a magnetic field is produced in the metallic elements of the coating. The magnetic field has alternating polarity and dissipates the energy of the signal. The energy that is not dissipated by the individual carbonyl iron ferrite elements is reflected to other elements as shown in Figure 4.5.

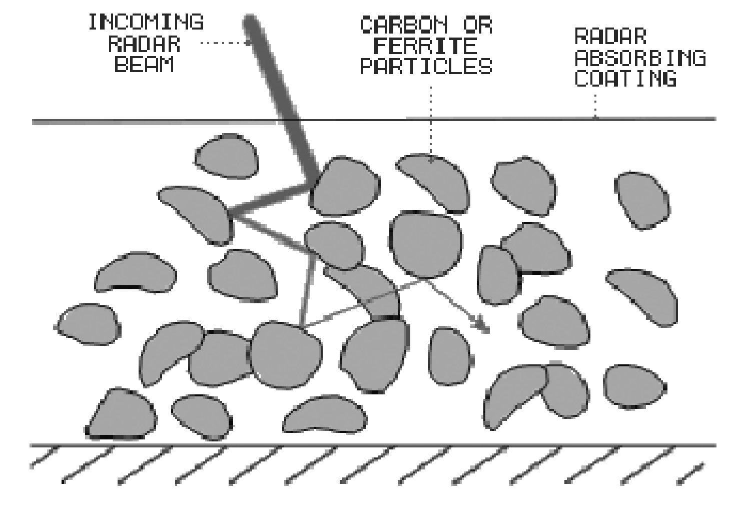


Figure 4.5: Incident Radar Beam Dissipated in RAM Coating

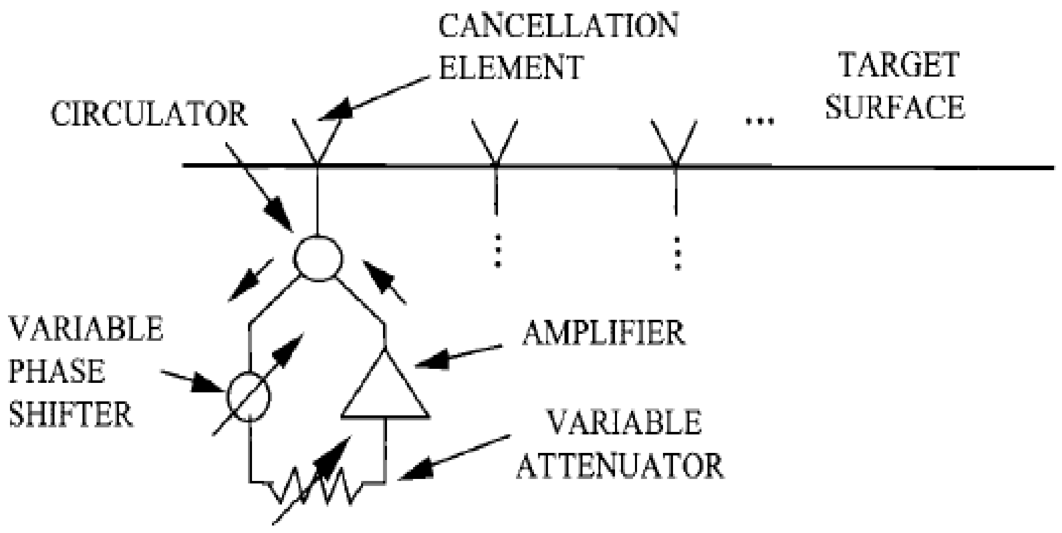
Underlying the operation of RAM is the fact that substances either exist or can be fabricated whose indices of refraction are complex numbers. In the index of refraction, which includes magnetic as well as electrical effects, the imaginary part accounts for the loss? At microwave frequencies, the loss is due to the finite conductivity of the material, as well as a kind of molecular friction experienced by molecules in attempting to follow the alternating fields of an impressed wave. It is customary to lump the effects of all loss mechanisms into the permittivity and permeability of the material because the engineer is usually interested in only the cumulative effect. Carbon was the basic material used in the fabrication of early absorbers because of its imperfect conductivity, and it continues to be important today. In fact, many commercial carbon-based materials now being marketed have designs that have not changed substantially for more than 20 years. Most are intended for experimental and diagnostic work, including the construction of indoor microwave anechoic chambers, but these materials are not easily applied to operational weapons platforms. They are usually too bulky and fragile in operational environments. Instead, magnetic absorbers are used more widely for operational systems [1].

The loss mechanism is primarily due to a magnetic dipole moment, and compounds of iron are the basic ingredients. Carbonyl iron has been used extensively, as have oxides of iron (ferrites). Magnetic materials offer the advantage of compactness because they are typically a fraction of the thickness of dielectric absorbers. However, magnetic absorbers are heavy because of their iron content and are inherently more narrowband than their dielectric counterparts. The basic lossy material is usually embedded in a matrix or binder such that the composite structure has the electromagnetic characteristics appropriate to a given range of frequencies.

5.2.3 Passive Cancellation

Passive cancellation, one form of which is known as impedance loading, received a great deal of attention in the 1960s, but the method is severely limited. The basic concept is to introduce an echo source whose amplitude and phase can be adjusted to cancel another echo source. This can be accomplished for relatively simple objects, provided that a loading point can be identified on the body. A port can be machined in the body, and the size and shape of the interior cavity can be designed to present optimum impedance at the aperture. Unfortunately, even for simple bodies, it is extremely difficult to generate the required frequency dependence for this built-in impedance, and the reduction obtained for one frequency in the spectrum rapidly disappears as the frequency changes.

Furthermore, typical weapons platforms are hundreds of wavelengths in size and have dozens, if not hundreds, of echo sources. Clearly, it is not practical to devise a passive cancellation treatment for each of these sources. In addition, the cancellation can revert to reinforcement with a small change in frequency or viewing angle. Consequently, passive cancellation has for the most part been discarded as a useful RCS reduction technique as shown in figure 4.6 [1].



.Figure 4.6: Passive Cancellation system

5.2.4 Active Cancellation

Also known as *active loading,* active cancellation is even more ambitious than passive loading. In essence, the target must emit radiation in time coincidence with the incoming pulse whose amplitude and phase cancel the reflected energy. This implies that the target must be "smart" enough to sense the angle of arrival, intensity, frequency, and waveform of the incident wave. It must also be smart enough to know its own echo characteristics for that particular wave length and angle of arrival rapidly enough to generate the proper waveform and frequency. Such a system must also be versatile enough to adjust and radiate a pulse of the proper amplitude and phase at the proper time. Clearly, the relative difficulty of active cancellation increases with increasing frequency, as scattering centers go in and out of phase with smaller aspect changes and where scattering patterns become more complex. Thus, if it has a place at all, active cancellation appears most suitable for low-frequency RCSR, where use of absorber and shaping become very difficult and scattering patterns xhibit broader lobes. Research on the technique is likely to continue because other practical means of RCSR are also difficult to apply for low frequencies. Figure 4.7 illustrate the active cancellation model [1].

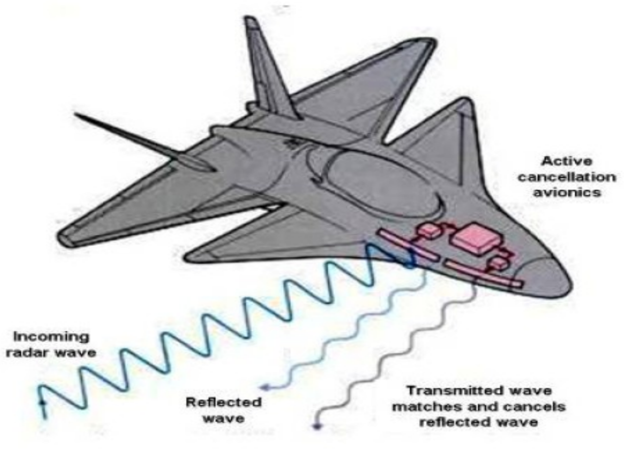

Active cancellation avionics
Incoming radar wave
Reflected wave
Transmitted wave matches and cancels reflected wave

Figure 44. Active Cancellation System (From [7])

.Figure 4.7: Active Cancellation system

5.3 The Penalties of RCSR

Most of the time, the requirement for reduced radar echo conflicts with conventional or traditional requirements for structures. As a result, the final system design is a compromise that inevitably increases the cost of the overall system, from initial engineering through production. Cost is only one penalty of RCSR; others are [1]

- Reduced payload;
- Reduced range;
- Added weight;
- Increased maintenance.

The relative importance of each factor depends on the mission of the particular platform involved, of course, and these factors change from one system to another. Not surprisingly, radar cross section reduction cannot always be justified, at least in terms of improved detection ranges. In one study, for example, Georgia Tech calculated the detection range for a hypothetical sea target ingressing against shore-based radars. The detection range was decreased less than 10% despite drastic changes in the target to reduce its radar echo. One reason for this was that the assumed threats were very sensitive and the target was detected as it came over the horizon, treated or not. On the other hand, although unsupportable on the basis of detection, RCSR for that platform may well have been justified when considered in concert with electronic countermeasures. In any event, the radar signature is simply one part of a platform specification. Trade-offs will always have to be made with respect to a large number of operational characteristics. Only when all of the characteristics are jointly considered in light of the platform mission can a truly optimum solution to maximize survivability and mission effectiveness be devised [1].

5.4 Limitations

- **Aerodynamic Performance**

  Stealth designs sacrifice parts of aircrafts' aerodynamic performance for stealth. Extreme examples such as F-117 and B-2 are unstable on 3 axes, poor in maneuvability and unable to perform supersonic flight.

- **Costs**

  High costs incur in designing, manufacturing and operating. Stealth aircrafts' designs are much more complicated than normal aircrafts. Heavy financing is required to developing computer programs to analyze RCS and aerodynamic performance. The special materials used small quantity of final products and attempts to keep secrecy dramatically rise the price of each aircraft. For example, 21 B-2 bombers, in total, cost the United States about 45 billion US dollars. Radar stealth aircrafts also have high maintenance requirement for their specially coated skin. Not mentioning the budget spent on short-lasting RAM, the specially built and air-conditioned hangars, and the maintenance team alone need a large amount of money to operate.

- **Intensifying Other Aspects of Detection**

  Purpose shaping usually aims to reduce RCS in one direction. It directs radar waves to other directions and intensifies radar echoes in those directions. For example, F-117 has an intensified radar echo in the direction of the cone above the horizontal plane, which makes it more observable in that direction. Non-resonant RAM transfer electromagnetic energy into heat, which increase the surface temperature and make aircrafts more vulnerable to infra-red detectors.

Chapter Five

6. CHAPTER FIVE: SYSTEM MODEL SIMULATION

6.1 RADAR Simulation

6.1.1 Introduction

It is a simulation of an early warning radar. The simulation works at the Intermediate Frequency (IF) level and calculates the return pulses to extract target information (coordinates, velocity, acceleration and Radar Cross Section (RCS)).
For each radar pulse the simulation calculates the returns that the radar will receive from all the targets according to the radar formula (taking into account transmission power, antenna gain, targets distance and targets RCS). The simulation calculates the amplitude and phase of the return signal (according to the IF frequency).

The radar builds a vector of its samples as complex signals (representing amplitude and phase delay) and adds it to complex random RF noise. The vector goes through a LPF representing the receiver BW and then I add another complex random noise representing the radar's thermal noise (in the digitizer).

The radar saves several reception periods to a buffer and then processes the entire buffer. The radar can perform a match filter over the received signal. The radar analyzes each range cell in the buffer in search of a target.
Detection threshold can be fixed or dynamic according to the buffer statistics (a sort of Constant False Alarm Rate (CFAR)). A simple CFAR mechanism was added.

In case Moving Target Indicator (MTI) is used the target detection is done in the frequency plane of the complex signal. Massive profiler work was done to improve real-time performance (in 50 kHz sampling rate and no analog feedback the simulation time is correlated to real-time).

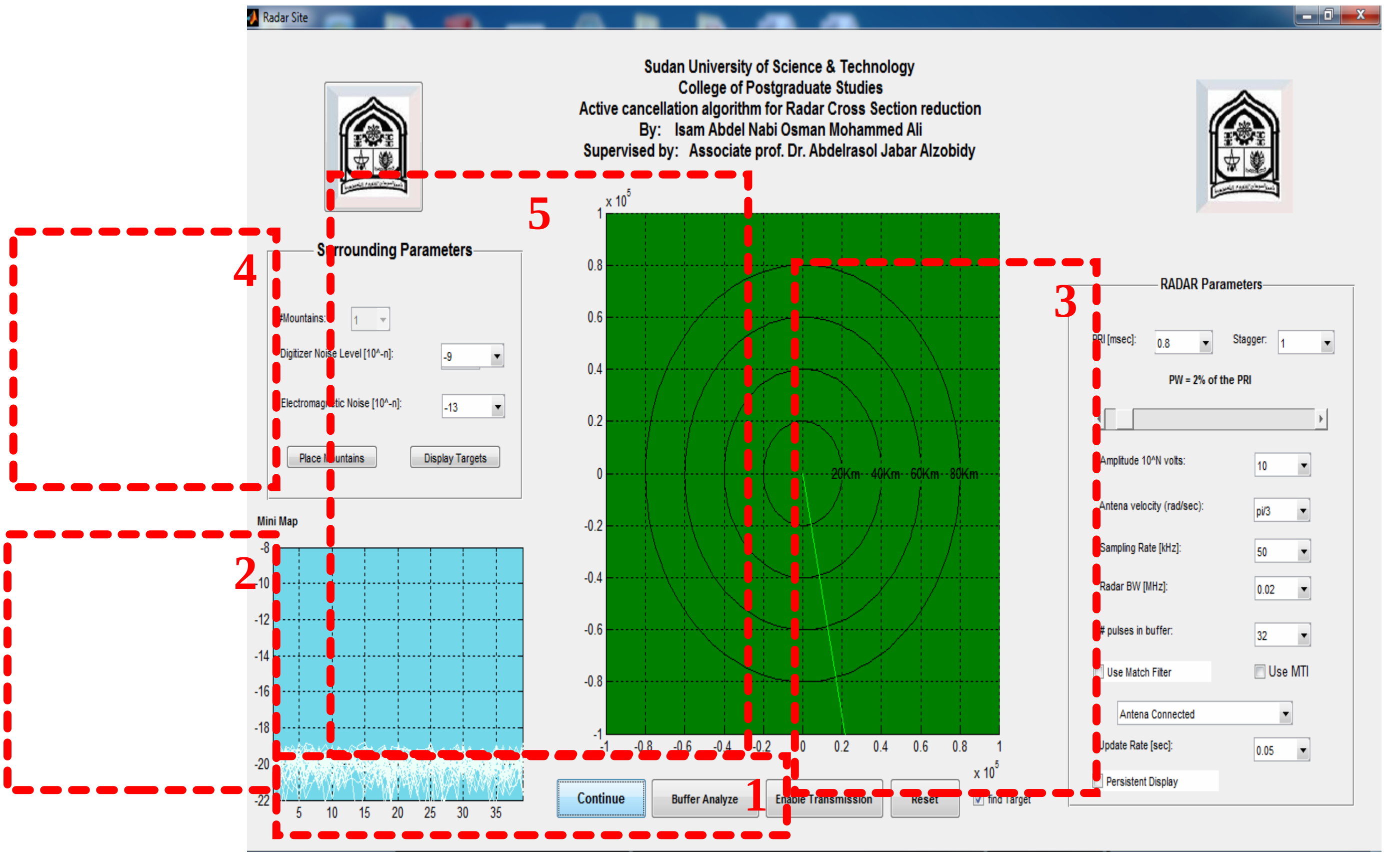
Each detected target is plotted on the main radar display. In case MTI is used a representation of the target’s velocity is also plotted (stationery targets are plotted as mountains).

6.1.2 Main Files and Functions

- **radarSimulation.m –** GUIDE file, handles all the GUI callbacks. The simulation is handled in “run_Callback” function.
- **runRadarSim.m –** Main simulation handling function. Basically this function is a while loop which keep transmitting pulses and processing them. The function main parts are transmitting, adding noise and processing the radar buffer.

6.1.3 The Radar Simulation Graphical User Interface

1. **Main Simulation Controls**
   a. Start\Pause\Continue – Starts and stops the simulation
   b. Buffer Analyze – Collects pulses into the radar buffer and then stops the simulation to display the buffer content in separate graphs. In case the “find target” is checked the simulation will only stop when it will find a target in the buffer.
   c. Enable Transmission – Configure the Network interface card to work as radar antenna to enable wave propagation.
   d. Reset – Resets all the simulation parameters (sets time back to zero and deletes all existing targets, new simulation targets are created on “Start”).
2. Mini Display – Displays real position of the simulation targets (for reference).
3. The main radar controls (see bellow).
4. The main simulation controls (see bellow).
5. The main radar display (see bellow).


Radar Site

Sudan University of Science & Technology
College of Postgraduate Studies
Active cancellation algorithm for Radar Cross Section reduction
By: Isam Abdel Nabi Osman Mohammed Ali
Supervised by: Associate prof. Dr. Abdelrasol Jabar Alzobidy

Surrounding Parameters
#Mountains: 1
Digitizer Noise Level [10^-n]: -9
Electromagnetic Noise [10^-n]: -13

RADAR Parameters
PRI [msec]: 0.8
Stagger: 1
PW = 2% of the PRI
Amplitude 10^N volts: 10
Antena velocity (rad/sec): pi/3
Sampling Rate [kHz]: 50
Radar BW [MHz]: 0.02
# pulses in buffer: 32
Use Match Filter
Use MTI
Antena Connected
Update Rate [sec]: 0.05
Persistent Display

Figure 5.1: Radar Simulation graphical user interface

6.1.4 Main RADAR display

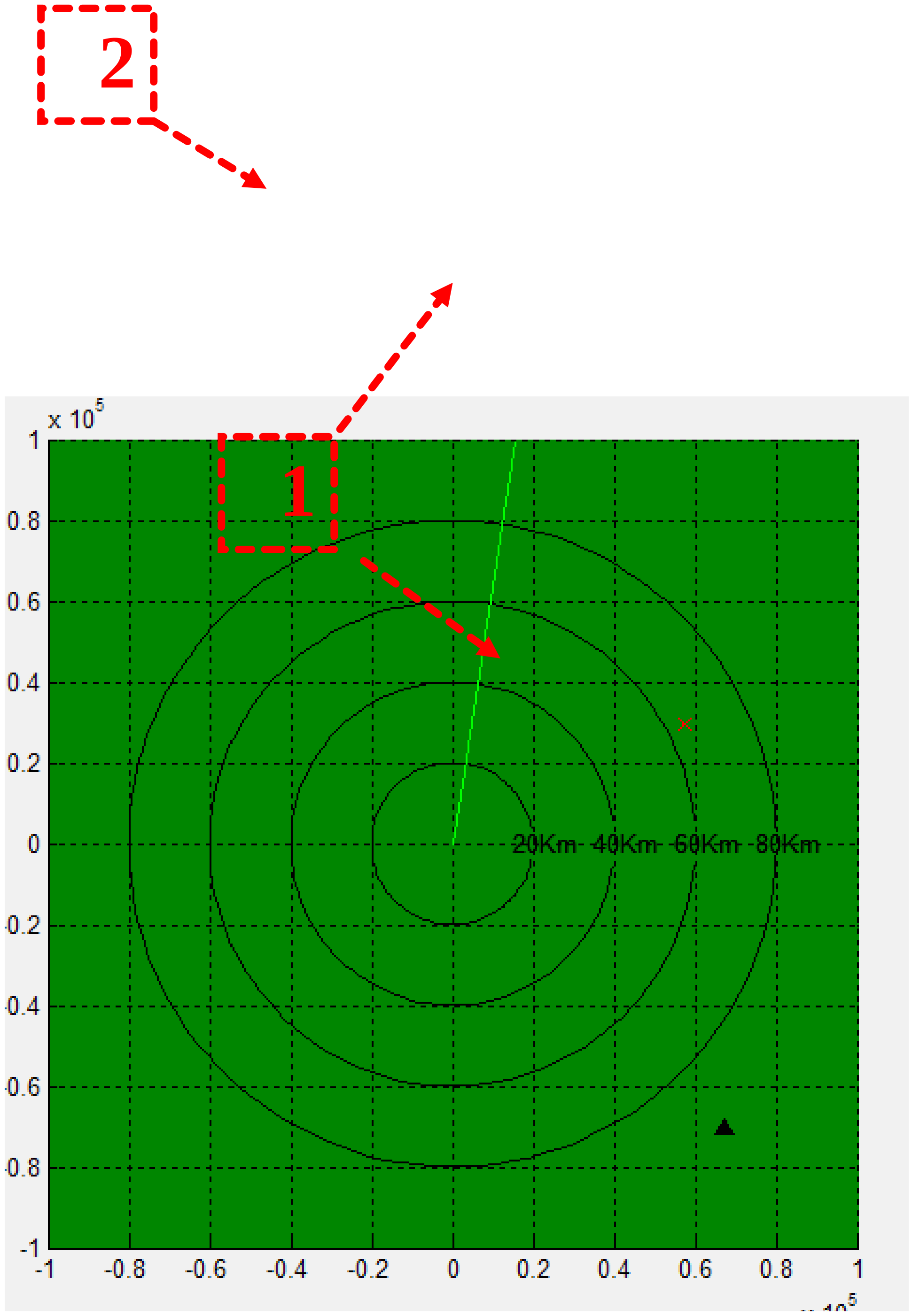


Figure 5.2: Main Radar display

1. The radar found targets.
    a. Red X - target with no velocity estimation (MTI is off)
    b. Red Arrow – a moving target with velocity estimation.
    c. Black Triangle – a stationary target (clutter).
2. The antenna point of direction.

6.1.5 Radar Controls

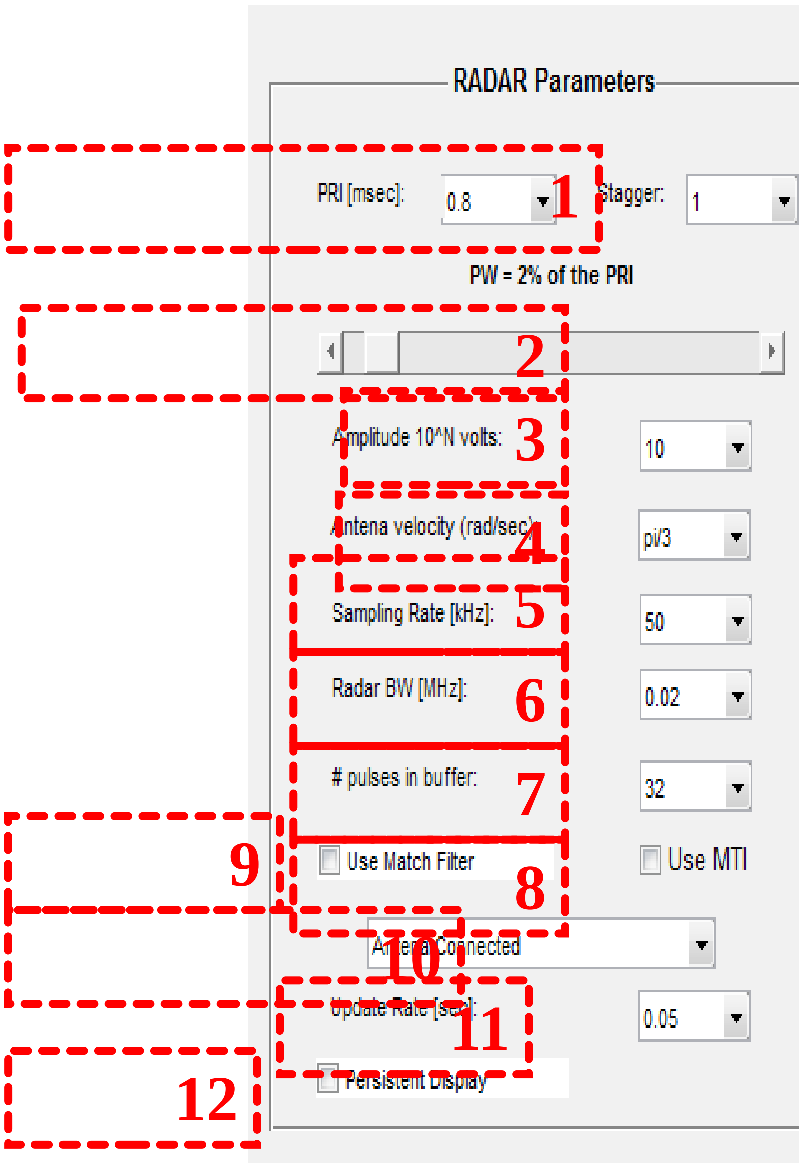


Figure 5.3: Radar Controls

1. Pulse Repetition Interval and the number of different intervals used (stagger). The PRI is the average time spent between each radar pulse transmission. The PRI defines the radar maximum range and the overall energy hitting a target.
2. Pulse Width – defined as a fraction of the PRI and sets the transmission duty cycle.
3. Transmitted Amplitude – The radar transmission power.
4. Antenna Velocity – The antenna turn velocity in radians.
5. The radar sampling rate – Defines the radar range cell size and directly affects the simulation real-time performance.
6. Radar Band Width – The reception Band Width, determines the shape of the received signal and the amount of electromagnetic noise in the receiver.
7. Number of pulses in the radar's buffer, the radar analyzes reception every number of pulses.
8. Moving Target Indicator – Activate phase analysis in frequency plane to distinguish moving targets from stationary ones (and estimate its relative velocity to the radar).
9. Use a Match filter on received signal.
10. Which type of antenna is connected (disconnect the antenna to estimate the digitizer noise).
11. The radar displays update time.
12. When using Persistent display previous detected targets are not erased from the radar display (helps tracking the planes path).

6.1.6 Simulation Controls

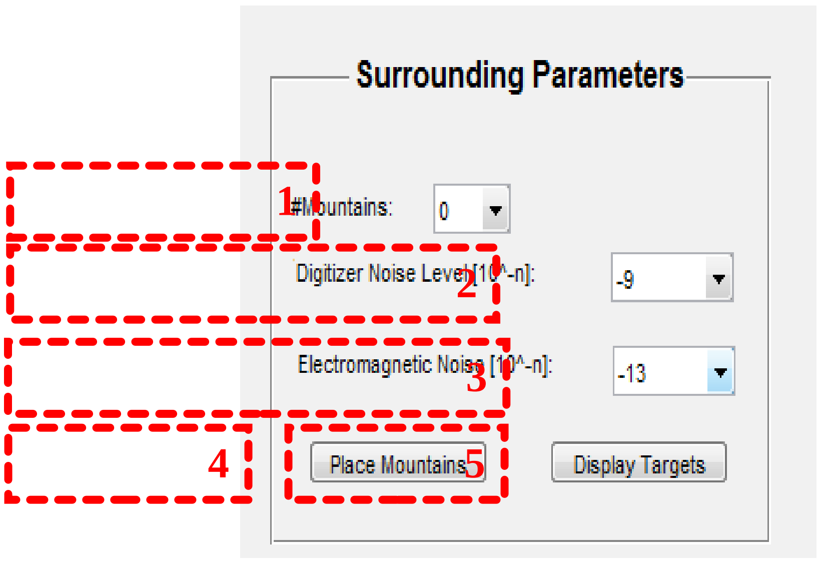


Figure 5.4: Radar Simulation Controls

1. Controlling the simulation culture. The mountains are created randomly on “Start” according to number of mountains.
2. The noise level simulated in the system. This is a normal distribution noise. The RF noise is multiplied by the radar BW and added before the applying of the BW. Digitizer noise is added after the BW.
3. Place Mountains – Enables you to place mountains in the simulation, click on the main radar display to place the mountains, when finished repress the “Place Mountains”. These mountains will be added to the simulation targets.
4. Display Targets – Opens a separate figure displaying the real position of the targets.

6.2 Aircraft Simulation

6.2.1 Introduction

1 It is a simulation of an active cancellation algorithm for radar cross section reduction which consists of two elements:

1A. Hardware components:

- Receiving antenna, used to receive a radar signal.
- Transmitting antenna, used to transmit a cancellation signal.
- Reconnaissance receiver used to stores a precision copy a received signal.

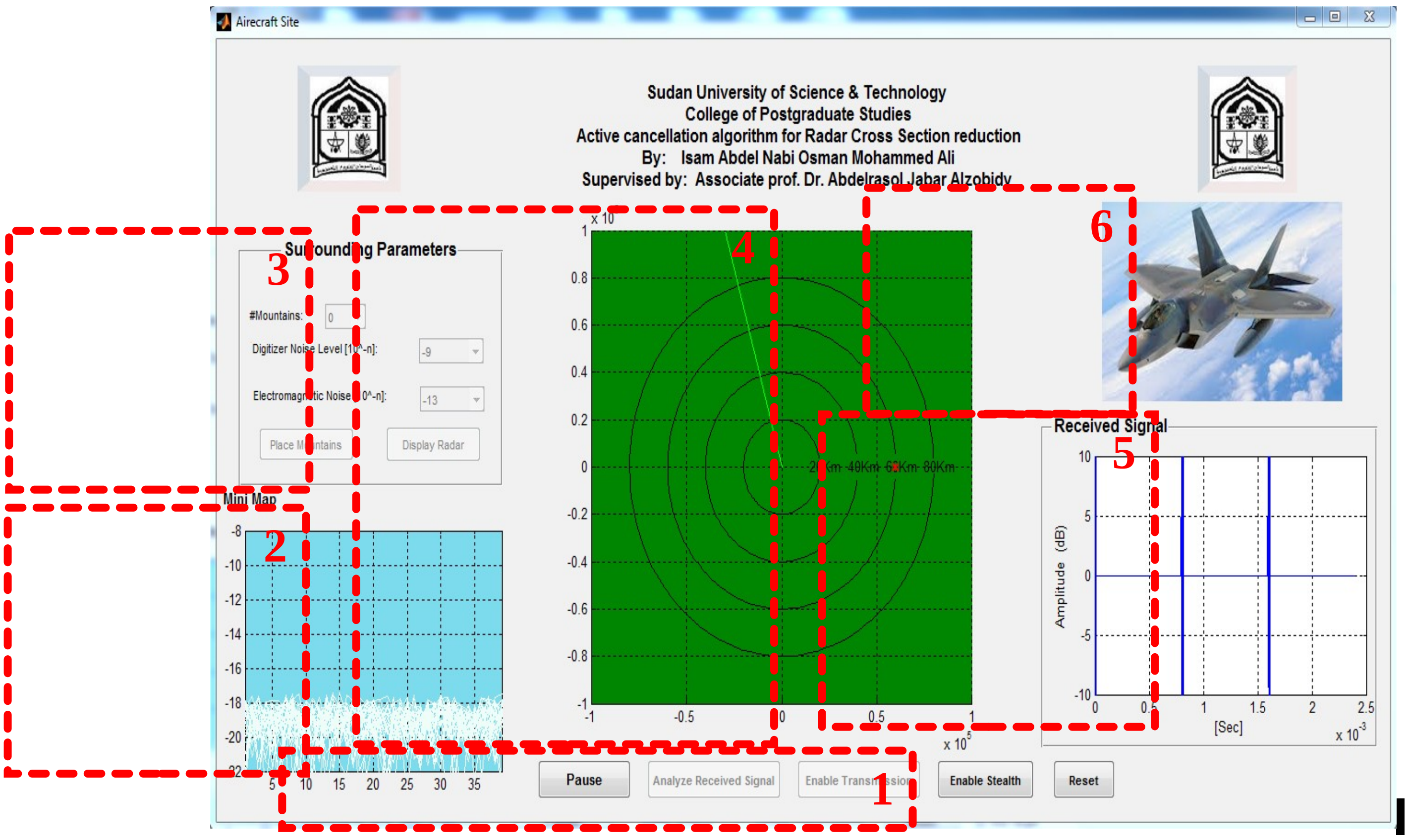
Network Card and Memory

1B. Software(MATLAB/C functions and databases):

- Signal processing and control function (SPC), used to storing a received signal, database searches, signal analysis, processing, and control of other elements.
- Blind Velocity and Doppler Effect function (BVDE), used to calculate the blind velocity according to the radar PRF and Doppler Effect.
- Power synthesis and beam forming function (PSBF), used to form the modified beam to transmit.
- Target RCS database, is related to frequency, direction, and polarization, power, for incident signal.
- Noise database, is related to the effective noise temperature, input noise power, etc.
- Clutter database, is related to aircraft speed, airborne, carrier frequency, radar point, altitude radar, distance to target and radar pulse repetition frequency (PRF).

6.2.2 Main Files and Functions

- **AircraftSimulation.m –** GUIDE file, handles all the GUI callbacks. The simulation is handled in “run_Callback” function.
- **runAircraftSim.m –** Main simulation handling function. Basically this function is a while loop which keep receiving pulses and processing them to extract phase, amplitude and frequency of the received pulses. The function main parts are transmitting an anti phase pulses, adding noise to that pulses.


Aircraft Site

Sudan University of Science & Technology
College of Postgraduate Studies
Active cancellation algorithm for Radar Cross Section reduction
By: Isam Abdel Nabi Osman Mohammed Ali
Supervised by: Associate prof. Dr. Abdelrasol Jabar Alzobidy

Surrounding Parameters

#Mountains: 0

Digitizer Noise Level [10^-n]: -9

Electromagnetic Noise [10^-n]: -13

Figure 5.5: Aircraft Simulation graphical user interface

6.2.3 The Aircraft Simulation GUI

1. Main Simulation Controls
   a. Start\Pause\Continue – Starts and stops the simulation
   b. Analyze Received signal – Collects pulses into the buffer, processing them and then stops the simulation to display the buffer content in separate graphs.
   c. Enable Transmission – Configure the Network interface card to work as radar antenna to enable wave propagation.
   d. Enable/Disable Stealth – Transmit an anti phase pulses which generated according to received pulses.
   e. Reset – Resets all the simulation parameters (sets time back to zero and deletes all existing targets, new simulation targets are created on “Start”).
2. Mini Display – Displays real position of the radar (for reference).
3. The main simulation controls (see bellow).
4. The main aircraft display (see bellow).
5. Received Signal – Display one received pulse.
6. Stealth State – Shows whiter the aircraft in visible mode or not.

6.2.4 Main Aircraft display

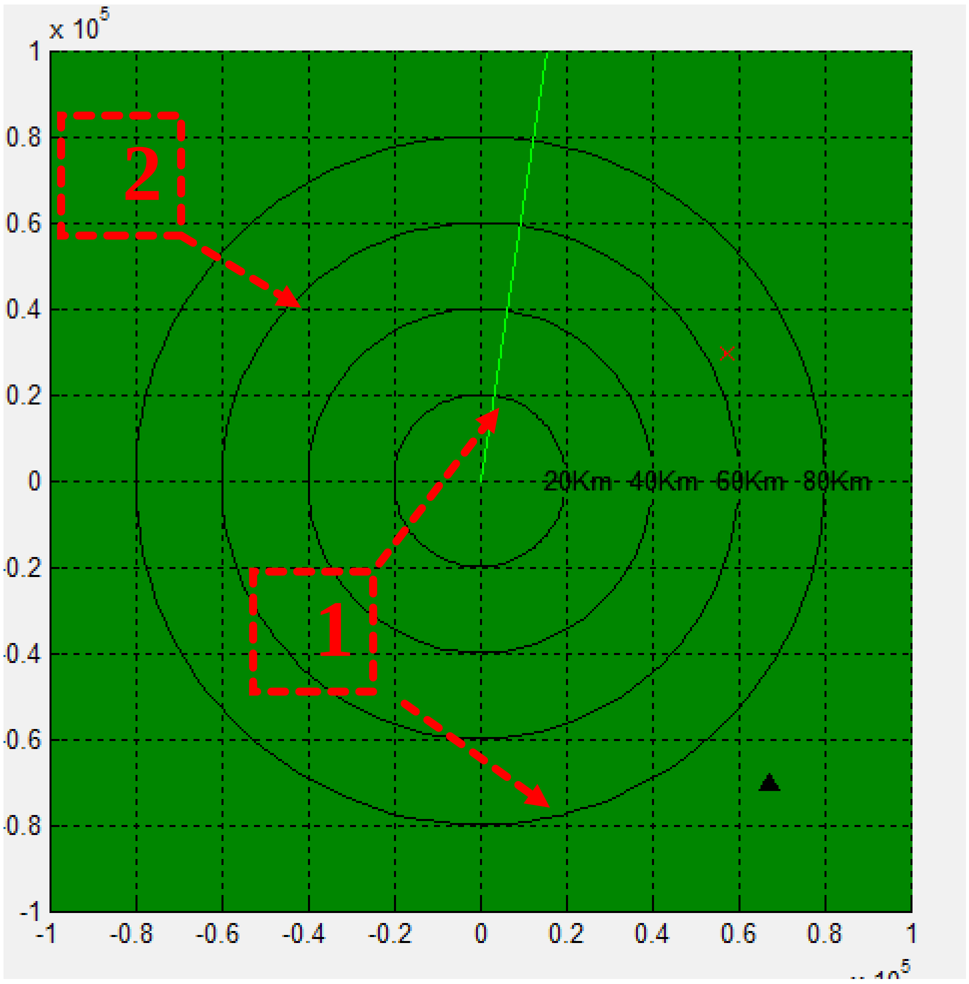
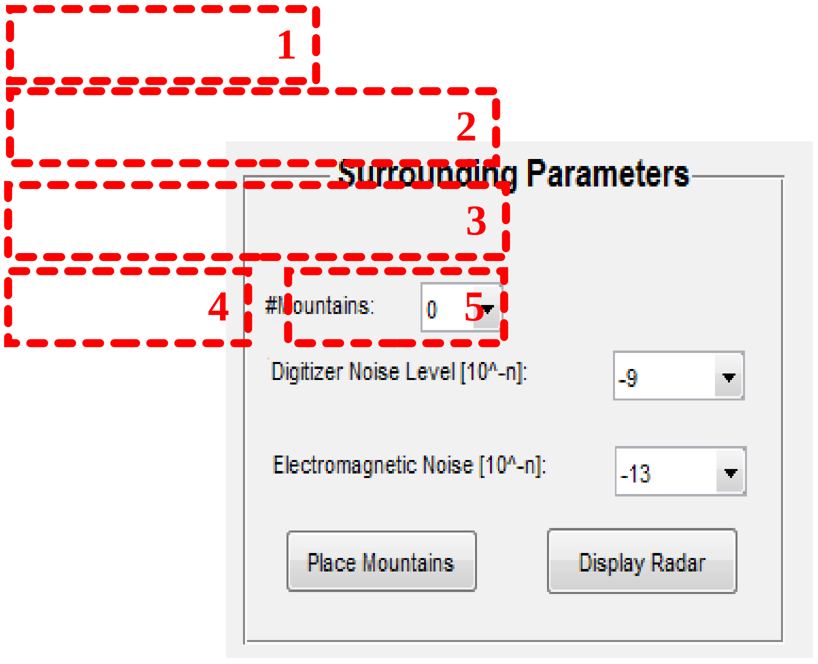


Figure 5.6: Main Aircraft display

1. The radar found Radar.
    a. Red X – radar.
    b. Black Triangle – a stationary target (clutter).
2. The antenna point of direction.

6.2.5 Simulation Controls



#Mountains: 0

Digitizer Noise Level [10^-n]: -9

Electromagnetic Noise [10^-n]: -13

Figure 5.7: Aircraft Simulation Controls

1. Controlling the simulation culture. The mountains are created randomly on “Start” according to number of mountains.
2. The noise level simulated in the system. This is a normal distribution noise. The RF noise is multiplied by the radar BW and added before the applying of the BW. Digitizer noise is added after the BW.
3. Place Mountains – Enables you to place mountains in the simulation, click on the main radar display to place the mountains, when finished repress the “Place Mountains”. These mountains will be added to the simulation targets.
4. Display Radar – Opens a separate figure displaying the real position of the radar.

Chapter Six

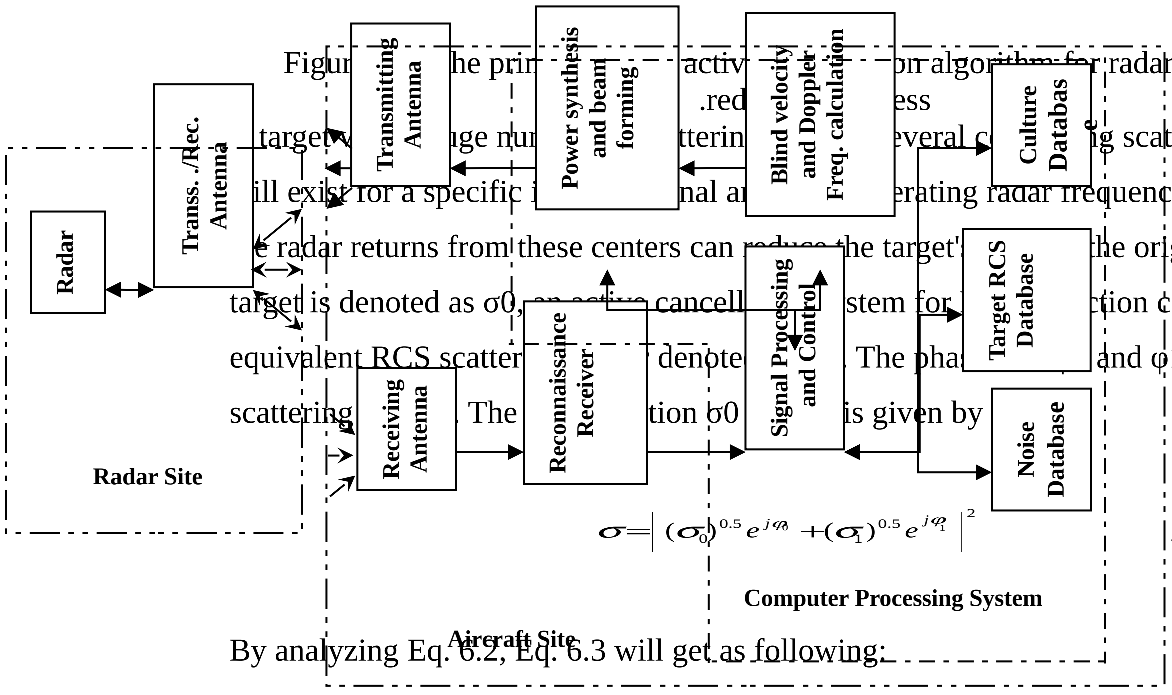
7. CHAPTER SIX: METHODOLGY

7.1 Introduction:

The algorithm based on generation an anti phase electromagnetic signal to a target's scattered signal. The effectiveness of this algorithm depends on the knowledge of its real-time characteristics, the measurement precision of the radar signal, and the accuracy of the generated cancellation signal, among other factors. Figures 6.1 and 6.2 respectively show the principle and flowchart of an active cancellation algorithm for radar cross section reduction. The incident radar phase, amplitude, frequency, polarization, radar space position and waveform characteristics are accurately and quickly measured on the Aircraft by SPC function, and a reconnaissance receiver. The characteristics of target reflection that correspond to the incident radar waveform will be extracted from the target's RCS database controlled by the computer processing system. By generating a signal (waveform) with the appropriate parameters, including phase, intensity, polarization and frequency, the target's echo can be cancelled when the wave returns to the radar receiving antenna. If we can solve the target to separate N scattering centers, then a radar return on a specific frequency is Eq.6.1:

$$\sigma = \left| \sum_{n=1}^{N} (\sigma_n)^{0.5} . e^{j\varphi_n} \right|^2 \quad \ldots \quad 6.1$$

Where σ_n is the Nth scatter RCS and φ_n is the phase due to the physical location of scatterer's.



Figur [illegible] he prin [illegible] activ [illegible] on algorithm for radar cross section .red [illegible] ess

[illegible] target w [illegible] ge nu [illegible] tterin [illegible] everal c [illegible] g scattering centers [illegible] ill exist for a specific i [illegible] nal ar [illegible] erating radar frequency. Reduction of [illegible] e radar returns from these centers can [illegible] he target' [illegible] he original RCS of the target is denoted as σ0, [illegible] cancell [illegible] stem for [illegible] ction can produced an equivalent RCS scatter [illegible] denote [illegible] The phas [illegible] and φ1, respectively of scattering [illegible] The [illegible] tion σ0 [illegible] is given by

$$\sigma = \left| (\sigma_0)^{0.5} e^{j\varphi_0} + (\sigma_1)^{0.5} e^{j\varphi_1} \right|^2 \quad \ldots \quad 6.2$$

By analyzing Eq. 6.2, Eq. 6.3 will get as following:

$$\sigma = \sigma_0 \left| 1 + \sigma_1 / \sigma_0 + 2(\sigma_1 / \sigma_0)^{0.5} [\cos(\varphi_1 - \varphi_0)] \right| \quad \ldots \quad 6.3$$

Controlling σ_1 and φ_1 can optimize those parameters to get Eq. 6.4:

$$\begin{cases} \sigma_1 = \sigma_0 \\ \varphi_1 - \varphi_0 = (2k+1)\pi \quad (where\ k\ is\ Integer) \end{cases} \quad \ldots \quad 6.4$$

When $\sigma = 0$, this indicates that stealth in the direction of the enemy's radar has been achieved.

Compiling the target's RCS database is an important step in designing an active cancellation algorithm for RCS reduction. The RCS entry is a function, rather than just varied number with different frequency, direction, and polarization for incident signal. It is necessary to establish an RCS database due to different frequencies, polarizations, and directions, according to real-time measurements for frequency, direction, power, and polarization of incident signal. This database must support real-time modification for the parameter of the transmitter to produce an effective cancellation signal for transmission. The reconnaissance receiver is used for reception and reconnaissance signals from enemy's radar transmitters. A received radar signal is applied to SPC function which stores a precision copy a received signal and alternately, it analyzed for amplitude and phase adjustment to generate anti phase signal with same amplitude. By comparing the stored signal with the wave signal generated by SPC function, the result from this comparison is checked to obtain a minimum output value (zero balance if possible), when this happen, the SPC function will send the radar signal to the BVDP function, with coherent superposition of clutter and noise to calculate the blind velocity an Doppler effect. The transmitting antenna and PSBF function are used to form and transmit the active cancellation signal. The system’s memory is used for storing the databases, including the noise database, target echo database, and the clutter database. The system work with the assumption that an echo signal consists of three parts: noise, clutter, and target echo so a radar echo can be as follows Eq. 6.5:

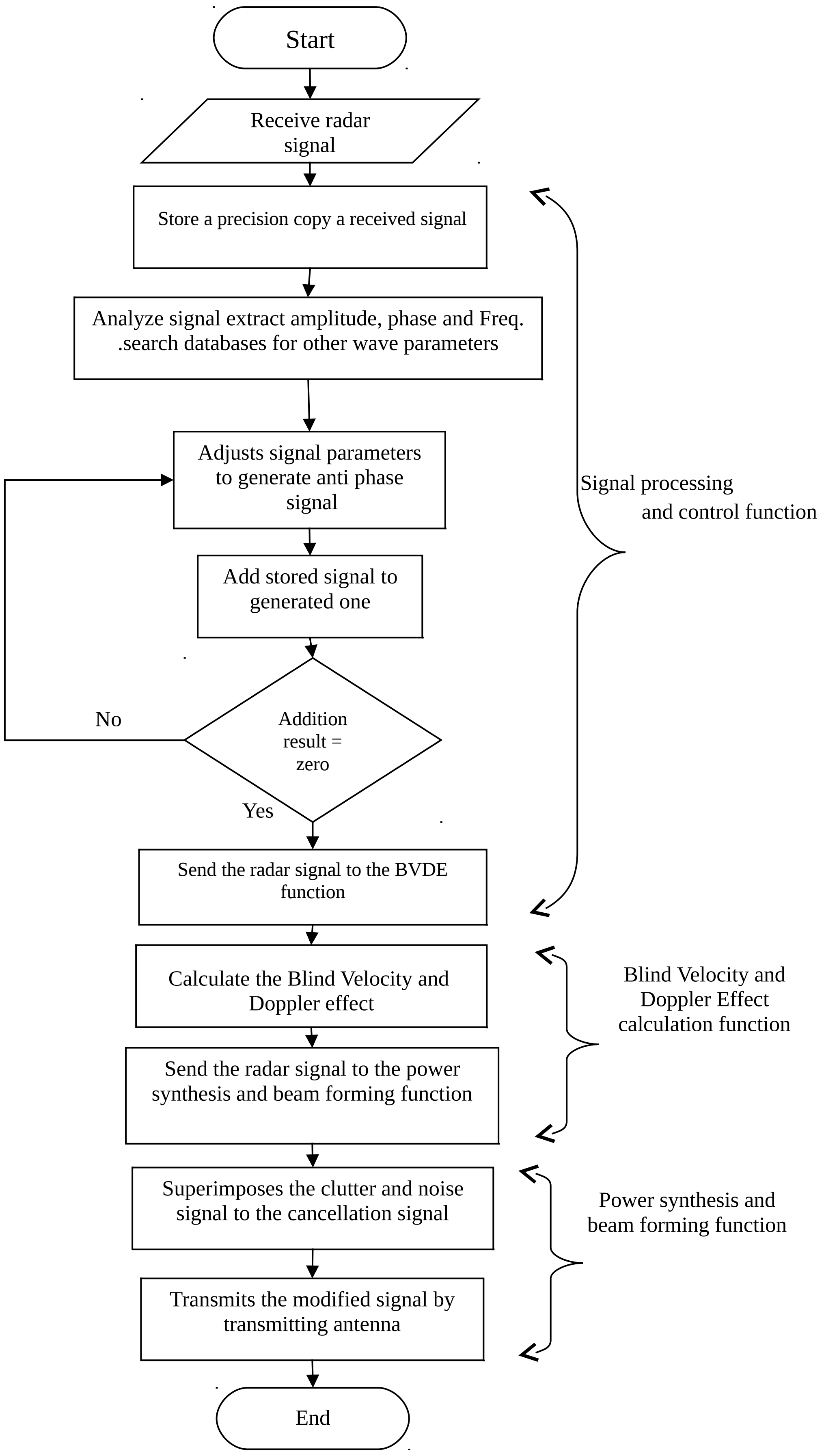
$$X(t) = S(t) + N(t) + C(t)) \quad \ldots \quad 6.5$$

Where *S(t)* is target's echo signal, *N(t)* is the noise signal, and *C(t)* is the clutter signal.

Due to a great deal of calculation and processing power that required to determine the radar's cancellation signal; it is difficult to achieve calculations in real-time without pipeline delays. For this, an offline calculation is used to build a target RCS database.

Numerical prediction method is the based method for obtaining a complex target RCS [1], [13].The database for noise and clutter usually uses a distribution of Gaussian for White noise, which can be produced by the Monte Carlo method [10],[11],[12]. Clutter data is related to aircraft speed, carrier frequency, radar point, altitude radar, distance to target and radar pulse repetition frequency (PRF). When clutter data are calculated, reduction of large amount of data is done because the speed, radar frequency, and aircraft altitude are fixed, and only PRF and radar point are changed.

7.2 Flowchart:



Start
Receive radar signal
Store a precision copy a received signal
Analyze signal extract amplitude, phase and Freq. .search databases for other wave parameters
Adjusts signal parameters to generate anti phase signal
Add stored signal to generated one
Addition result = zero
No
Yes
Send the radar signal to the BVDE function
Calculate the Blind Velocity and Doppler effect
Send the radar signal to the power synthesis and beam forming function
Superimposes the clutter and noise signal to the cancellation signal
Transmits the modified signal by transmitting antenna
End
Signal processing and control function
Blind Velocity and Doppler Effect calculation function
Power synthesis and beam forming function

.Figure 6.2: An active cancellation algorithm for radar cross section reduction flowchart

7.3 Code:

The code consists of three main functions and three databases as follow:

- Signal processing and control function (SPC), used to storing a received signal, database searches, signal analysis, processing, and control of other elements.
- Blind Velocity function (BVDE), used to calculate the blind velocity – *Blind velocity is defined as radial velocity at which the target appears stationary and echoes from it are canceled by the MTI action* [14] - and Doppler Effect to avoid MTI system.
- Power synthesis and beam forming function (PSBF), used to form the modified beam to transmit.
- Target RCS database, is related to frequency, direction, and polarization, power, for incident signal.
- Noise database, is related to the effective noise temperature, input noise power, etc.
- Clutter database, is related to aircraft speed, carrier frequency, radar point, altitude radar, distance to target and radar pulse repetition frequency (PRF).

7.3.1 Signal processing and control function (SPC):

```
function spc (handles,NPB,PRIA,SR,DZN,EN,PWA,STA,AMP,g)
    warning off all
    while get(handles.run,'value')
        if get(handles.CFAR,'value')
             useCFAR = true;
            CFAR = get(handles.Th,'value');
        else
            useCFAR = false;
            Th = 10^(get(handles.Th,'value'));
        End
get(handles.Amp,'value');
currentTime = currentTime+PRI;
radarAngle = mod(antenaTurnVelocity*currentTime,2*pi);
% ------------------------------- Receiving radar pulses -------------------------------------------
targets = [handles.Targets ; handles.mountains];
curentReturnPulses = zeros( bufferSize,1 );
        if ~isempty(targets)
            [t a phi] = targetsReturn(targets,antenaGain,Amp, currentTime,antenaTurnVelocity,
targetsTime,handles.IF_Freq);
tIn = round( t*Fs );
tIn = max(tIn,1);   tIn = min(tIn,bufferSize);
         for n=1:length(t)
```

```
curentReturnPulses(tIn(n):tIn(n)+PWn-1) = curentReturnPulses(tIn(n):tIn(n)+PWn-1) +
a(n)*exp(i*phi(n));
        end
        end
transmitInd = pulseTransmitionPoints( mod(pulseNum-1,nPRI)+1 );
index = transmitInd : transmitInd + bufferSize-1;
index( bufferSize-transmitInd + 2 : bufferSize ) = 1:transmitInd-1;
returnPulses(index) = returnPulses(index)+curentReturnPulses;
% -------------------------------- Processing Buffer -----------------------------------------
if ~mod(pulseNum,nPRI)  %Only processing every N number of pulses
    recievedSignal = returnPulses;
    n = get(handles.RadarBW,'value');
    radarBW=radarBWOptions(n) * 1e6;

%---------------------------------- Culture Noise-------------------------------------------------------------
   n = get(handles.digitizerNoiseLevel,'value');
   temp = get(handles.digitizerNoiseLevel,'string');
   if strcmp(temp{n},'off')
      digitizerNoiseLevel = 0;
   else
      n=str2double(temp{n});
      digitizerNoiseLevel = 10^n;
   end
%  -------------------------------------- Creating RF noise ---------------------------------------------------
          if RFnoiseLevel
              RFnoise = fastSemiRandn( length(returnPulses))* RFnoiseLevel *[1 ; i]*radarBW;
              recievedSignal = recievedSignal+RFnoise;
          end
a = -pi^2*radarBW^2/log(0.5)*log(exp(1));
responseStart = sqrt(-log(0.1)/log(exp(1))/a);
t = -responseStart : 1/Fs : responseStart;
response = exp(-t'.^2*a);
response = response/sum(response);
recievedSignal = conv2(recievedSignal,response,'same');
         if get(handles.useMatchFilter,'value');
              if length(response) > PWn
                 matchFilter = response;
              else
                 matchFilter = ones(PWn,1)/PWn;
              end
         processedRecivedSignal = conv2(recievedSignal,matchFilter,'same');
         else
              processedRecivedSignal = recievedSignal;
```

```
     end
% -------------------------------------------- Was there a radar -------------------------------------------------
    signalInRangeCells = processedRecivedSignal(rangeCellInd);
    isMTIused = get(handles.useMTI,'value');
    if isMTIused
        freqInRangeCells = fft( signalInRangeCells,freqRes,2 );
        energyInFreqRangeCells = abs( freqInRangeCells(:,freqInd) ).^2;
        if useCFAR
          noiseLevel = median( energyInFreqRangeCells(:) );    % Th is per frequency
          freqTh = noiseLevel * CFAR * nPRI;
        else
          freqTh = Th * nPRI;
        end
     [maxFreq maxFreqInd] = max( energyInFreqRangeCells,[],2);
     localMaxEnergy = imdilate( maxFreq , dilateKer );
     rangeCell = find( localMaxEnergy == maxFreq & maxFreq > freqTh );
     targetInd = sub2ind( [numSamplesInPRI freqRes], rangeCell, maxFreqInd( rangeCell ) );
     else
        energyInRangeCells = sum( abs(signalInRangeCells).^2,2 );
        if useCFAR
          noiseLevel = median( energyInRangeCells );
          Th = noiseLevel * CFAR;
        end
     localMaxEnergy = imdilate(energyInRangeCells,dilateKer);
     targetInd = find ( energyInRangeCells == localMaxEnergy & energyInRangeCells > Th);
     end
    if ~isempty(targetInd)
      foundTargetInBuffer = 1;
      if isMTIused
        [rangeCell ans] = ind2sub(size(energyInFreqRangeCells),targetInd(:));
        RCS = energyInFreqRangeCells(targetInd(:));
      else
        rangeCell = targetInd;
        RCS = energyInRangeCells(targetInd);
    end
      targetRange = ( rangeCell -PWn/2) / Fs / 2 * 3e8;
      targetV = ones(length(targetInd),1)*12345;
      for Rind = 1:length(targetInd)
    %Processing each Target
      pos = targetRange(Rind)*[cos(radarAngle) sin(radarAngle)];
      if isMTIused
```

```
                targetV(Rind) =
MTIcalcVelocityFromFourier(energyInFreqRangeCells,targetInd(Rind),freqRes,handles.IF_Freq,PR
I);
               end
    if isempty(handles.foundTargets)
     handles.foundTargets(end+1) = ...
                           createTargetObj( pos, RCS(Rind), targetRange(Rind), targetV(Rind),
radarAngle,0, radarAngle,radarAngle,[],numRadarTurn );
h = plotTarget( handles.foundTargets(end), handles );
handles.foundTargets(end).hPlot = h;
      else
% ---------------- Checking if this radar was already detected ------------------------------
           M = length(handles.foundTargets);
           dAngle = zeros(M,1);
           dRange = zeros(M,1);
           dV = zeros(M,1);
                for mm=1:M
                    d1 = calcDiffAngle( handles.foundTargets(mm).counterClockWise, radarAngle );
                    d2 = pi;
                    dAngle(mm) = min ( d1, d2)/angleRes;
                    dRange(mm) = abs( handles.foundTargets(mm).R - targetRange(Rind) )/rangeRes;
                    dV(mm) = abs( handles.foundTargets(mm).v - targetV(Rind) )/velocityRes ;
               end
 [sameTargetScore sameTargetInd] = min( dAngle.^2 + dRange.^2 + dV.^2 );
        if  sameTargetScore < 1
           if handles.foundTargets(sameTargetInd).RCS < RCS(Rind)
 %------------------------------ Keeping the parameters of the better received pulses;
           handles.foundTargets(sameTargetInd).pos = pos;
           handles.foundTargets(sameTargetInd).RCS = RCS(Rind);
           handles.foundTargets(sameTargetInd).R = targetRange(Rind);
           handles.foundTargets(sameTargetInd).v = targetV(Rind);
           handles.foundTargets(sameTargetInd).angle = radarAngle;
           delete( handles.foundTargets(sameTargetInd).hPlot );
           h = plotTarget( handles.foundTargets(sameTargetInd), handles );
           handles.foundTargets(sameTargetInd).hPlot = h;
      end
          if d1 < d2
                handles.foundTargets(sameTargetInd).counterClockWise = radarAngle;
          else
                 handles.foundTargets(sameTargetInd).clockWise = radarAngle;
          end
    else
          handles.foundTargets(end+1) = ...
          createTargetObj( pos, RCS(Rind), targetRange(Rind), targetV(Rind), radarAngle,0,
radarAngle,radarAngle,[],numRadarTurn );
          h = plotTarget( handles.foundTargets(end), handles );
          handles.foundTargets(end).hPlot = h;
  end
  end
  end
  end
        dt = currentTime-lastUpdate ;
        if dt > updateRate  %Is it time to update the display
        lastUpdate = currentTime;
        currentRealTime = toc;
        delay = dt - (currentRealTime - lastUpdateRealTime); % syncronizing to real time
        lastUpdateRealTime = currentRealTime;
        if delay > 0
               pause(delay);
        end
        isPersistentDisplay = get(handles.persistentDisplay,'value');
        handles =
plotFOV(handles,currentTime,antenaTurnVelocity,radarSector,maxDist,isPersistentDisplay);
           updatedDisplay = true;
           % updating the mini-map display every 5 seconds
           if currentTime > miniDisplayUpdateTime + 5
               if ~get( handles.scopeDisplay,'value' )
                  miniDisplayUpdateTime = currentTime;
                  displayTargets(handles,'in radar display')
                  numRadarTurn = ceil(currentTime*antenaTurnVelocity/2/pi);
               end
               if isPersistentDisplay
                  if length( handles.persistentPlotHandle ) > 1000
                     temp = handles.persistentPlotHandle;
                     delete( temp( 1:end-1000 ) );
                     handles.persistentPlotHandle = temp(end-999:end);
                  end
               end
           end
  % Updating targets position, velocity & accelaration
  targetsTime = currentTime; % the time in which the target position was updated
  for n =1:length(handles.Targets)
      handles.Targets(n).XY = handles.Targets(n).XY +
dt*handles.Targets(n).v+dt^2*handles.Targets(n).a;
      handles.Targets(n).v = handles.Targets(n).v + dt*handles.Targets(n).a;
      handles.Targets(n).a = handles.Targets(n).a * 0.95^dt;
```

```
      targetsManuv = handles.Targets(n).maneuverability;
        if (1-exp(-dt/targetsManuv)) > rand(1)
            phi = atan( handles.Targets(n).v(2)/handles.Targets(n).v(1) );
            phi = phi + 2*pi*(rand(1)-0.5);
            handles.Targets(n).a = randn(1)*20*[cos(phi) sin(phi)] -
handles.Targets(n).v/targetsManuv/2;
        end
 end end
       if isAnalyzeBufferMode
           if get(handles.waitForTarget,'value')
               if foundTargetInBuffer
                   if isMTIused
analyzBufferWithMTI(handles,recievedSignal,processedRecivedSignal,energyInFreqRangeCells,P
Wn,freqTh,freqInRangeCells,numSamplesInPRI,rangeCellInd);
                   else
analyzBuffer(handles,recievedSignal,processedRecivedSignal,energyInRangeCells,PWn,Th,range
CellInd,localMaxEnergy,g,PRI,Fs,nPRI);
                   end
set(handles.run,'string','Pause');
AirecraftSimulation('run_Callback',handles.run,[],guidata(handles.run));
           end
           else
               if isMTIused
analyzBufferWithMTI(handles,recievedSignal,processedRecivedSignal,energyInFreqRangeCells,P
Wn,freqTh,freqInRangeCells,numSamplesInPRI,rangeCellInd);
               else
analyzBuffer(handles,recievedSignal,processedRecivedSignal,energyInRangeCells,PWn,Th,range
CellInd,localMaxEnergy,g,PRI,Fs,nPRI);
                   set(handles.run,'string','Pause');
                   AirecraftSimulation('run_Callback',handles.run,[],guidata(handles.run));
               end
               set(handles.run,'string','Pause');
               AirecraftSimulation('run_Callback',handles.run,[],guidata(handles.run));
           end
       end
       % Display analog scope  !!!!!
       if get(handles.scopeDisplay,'value')
           hold( handles.miniDisplay,'off');
           sig = abs(processedRecivedSignal(rangeCellInd));
           plot(handles.miniDisplay,log( sig( PWn+1:end,:) ),'color',[239 255 250]/256,'linewidth',1);
           set( handles.miniDisplay,'color',[118 219 237]/256);
           ylim(handles.miniDisplay,[-22 -8]);
           xlim( handles.miniDisplay,[1 numSamplesInPRI-PWn] );
```

```
            grid( handles.miniDisplay,'on');
            if ~updatedDisplay
               drawnow;
               updatedDisplay = false;
            end
         end
end
pulseNum = pulseNum+1;
end
```

7.3.2 Blind Velocity and Doppler Effect function (BVDE):

```
function bede = DFSMFromFourier(freqRes,intermidateFreq,PRI)
   temp=3e8/[2*PRI*freqRes];
   vel=temp*3600; %velocity in "m/h"
   de=(2* freqRes*vel)/ 3e8;
   freqRes= freqRes+de;
end
```

7.3.3 Power synthesis and beam forming function (PSBF):

```
function PSBF
(handles,recievedSignal,processedRecivedSignal,energyInFreqRangeCells,PWn,Th,…
freqInRangeCells,PRIn,rangeCellInd);
% -------------------------------------------------build Antenna ---------------------------------------------------
n=get(handles.antenaMode,'value');
antenaMode = get(handles.antenaMode,'string');
switch antenaMode{n}
   case 'Antena Connected'
      reqVector  = [1 1.25].*1e9;       % Frequency range for element pattern
      sAnt       = phased.CustomAntennaElement('FrequencyVector',freqVector,...
                               'AzimuthAngles',az,...
                               'ElevationAngles',el,...
                               'RadiationPattern',pattern_azel);
      fmax = freqVector(end);
      plotResponse(sAnt,fmax,'Format','polar','RespCut','3d');
      view(3)
       c = 3e8;
      lambda = c/fmax;
      wsURA = warning('off', 'phased:system:array:SizeConventionWarning');
      sArray = phased.URA('Element',sAnt,'Size',10,'ElementSpacing',lambda/2)
      animcustantdemopattern(sArray)
       el_ang = -90:90;
      sArrayResponse = phased.ArrayResponse('SensorArray',sArray,'PropagationSpeed',c);
```

```
        el_pat = abs(step(sArrayResponse,fmax,el_ang));  % elevation pattern
        freespace_rng = 100;  % in km
        ant_height = 20;      % in m
        radarvcd(fmax,freespace_rng,ant_height, 'HeightUnit','m','RangeUnit','km',...
            'AntennaPattern',el_pat/max(el_pat),'PatternAngles',el_ang.');
        antenaGain = sinc(-1:0.001:1).^500;
        s = trapz(antenaGain)*2*pi/length(antenaGain); %integral over antenna gain
        antenaGain = antenaGain/s;
    case 'Antena with Side Lobes'
         antenaGain = sinc(-2.5:0.001:2.5).^4;
        s = trapz(antenaGain)*2*pi/length(antenaGain); %integral over antenna gain
        antenaGain = antenaGain/s;
    case 'Omni'
        antenaGain = [ 1/2/pi 1/2/pi];
    case 'Antena Disconnected'
        antenaGain = 0;
% --------------------------------------------Transmitting Cancellation signal ---------------------------------------
transmisionInd( 1:PWn ) = (1:PWn);
options = get(handles.stagger,'string');
stagger = str2double( options{temp} );
ratio = 2*PW / PRI;
ratio = max(ratio, 0.05);
intervals = (1+( (0:stagger-1)-(stagger-1)/2)*ratio)*PRI;
numSamplesInInterval = round( intervals*Fs );
numSamplesInInterval(end)=round( numSamplesInInterval(end)-
mod(sum(numSamplesInInterval),PRISize));
numIntervals = length(intervals);
numSamplesInPRI = round( sum( intervals )*Fs );
maxDist = numSamplesInPRI*150/Fs*1e6;
for n=2:nPRI
    interval = numSamplesInInterval(mod(n-1,numIntervals)+1);
    transmisionInd( (n-1)*PWn+1:n*PWn ) = transmisionInd( (n-2)*PWn+1:(n-1)*PWn ) + interval;
     pulseTransmitionPoints( n ) = transmisionInd( (n-1)*PWn+1 );
    end
rangeCellInd = zeros(nPRI,numSamplesInPRI);
    for n = 1 : nPRI
        rangeCellInd(n,:) = transmisionInd( (n-1)*PWn+1 );
    end
temp = repmat( 0:numSamplesInPRI-1,nPRI, 1 );
rangeCellInd = rangeCellInd + temp;
rangeCellInd = mod( rangeCellInd-1,bufferSize ) + 1;
rangeCellInd = rangeCellInd';
end
```

Chapter Seven

8. CHAPTER SEVEN: TEST AND RESULTS

For the purpose of evaluation, the active cancellation algorithm for radar cross section reduction was simulated using MATLAB R2013a software and the following conditions:

1. A coherent pulse train with 5 MHz modulation rate for radar transmit signal.
2. The signal has PRF of 1.25 kHz and a pulse width of 16 µs.
3. Uniform speed for the target movement with 60 km initial distance away from the radar transmitter, 500 m/s initial radial velocity, 0 deg of both azimuth and elevation angles and 2 m^2 target's RCS.
4. The reference pattern function for reconnaissance is described by Eq. 7.1

$$F(\theta) = \begin{cases} ASa\quad [a\theta / (\theta_1/2)]k_0 & |\theta| \leq a_1 \\ BSa\quad [a(\theta \pm a_2)/(\theta_2/2)]k_0 & |\theta| > a_1 \end{cases} \qquad \ldots \qquad 7.1$$

Where $\kappa_0 = (\cos\theta_0)^{0.5}$ is the control factor for phased-array antenna modified beam gain with variation of scanning angle, θ_0 is the beam of scanning angle, θ_1 is the unbiased-beam main-lobe beam width of 3 dB, θ_2 is the unbiased-beam first side-lobe 3-dB beam width, AS is the unbiased-beam main-lobe gain value, BS is the unbiased-beam first side-lobe gain value, a = 2.783, ($a_1 = \pi\theta_1/a$) is first zero unbiased-beam (in rads), and ($a_2 = \pi(\theta_1 + \theta_2/a$) is the unbiased-beam first side-lobe peak point of view (in rads).The 3D Electromagnetic pattern can be simplified into an elevation and azimuth multiplication pattern result as shown in Eq. 7.2.

$$F(\theta,\varphi) = F_\theta(\theta)F_\varphi(\varphi) \qquad \ldots \qquad 7.2$$

Where $F_\varphi(\varphi)$ is the elevation pattern and $F_\theta(\theta)$ is the azimuth pattern. Assuming that the radar antenna vertical main-lobe beam-width is 2 degree, the main-lobe gain is 40 dB, the gain is 9 dB, and the first side-lobe-width is 1 degree. The algorithm can used for (ECM) Electronic countermeasures function using rectangular array antenna M x N pattern function described below as Eq. 7.3 [15]:

$$g(\theta,\varphi) = G(\theta,\varphi)\,|E(\theta,\varphi)|\,|e(\theta,\varphi)| \qquad \ldots \qquad 7.3$$

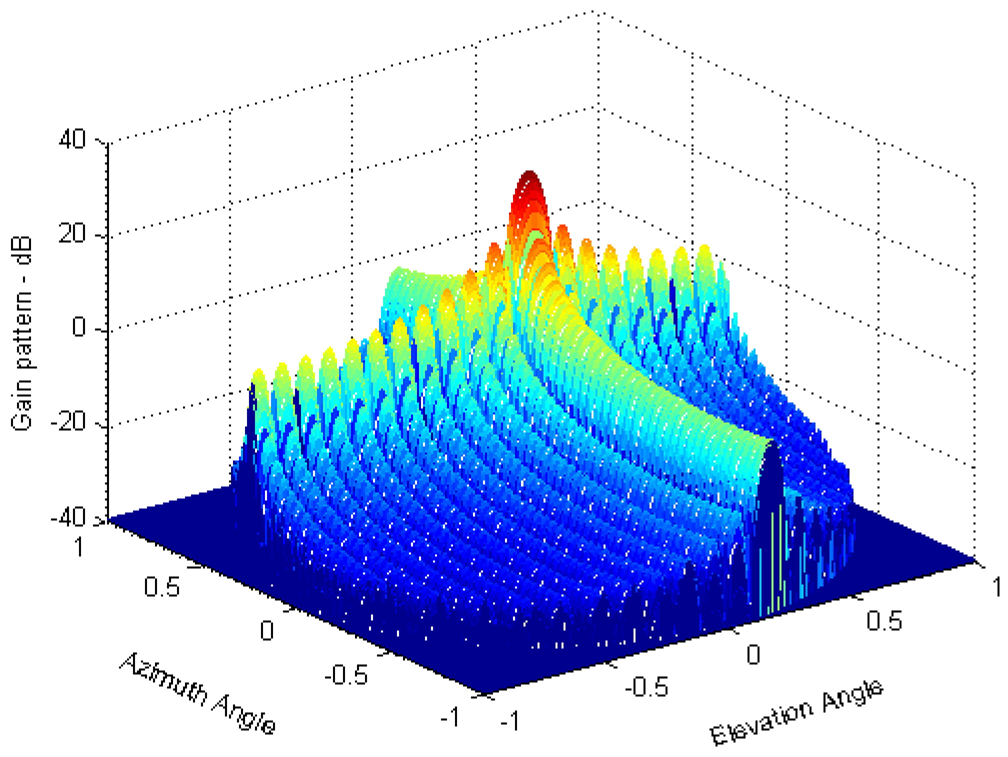
Where g(θ, φ) is the pattern of the antenna, G(θ, φ) is the factor of the directivity, E(θ, φ) is the array factor for the beam shape determines, e(θ, φ) is the factor of the array element with ($e(\theta, \varphi) \approx 1$), φ is the elevation angle on spherical coordinates array, θ is the azimuth angle on spherical coordinates array, ($\varphi \in [0, \pi/2]$), and($\theta \in [0, 2\pi]$). Adjacent-array element spacing of $d = \lambda/2$ can be described in the x and y directions, $E(\theta, \varphi)$ as Eq. 7.4

$$E(\theta,\varphi) = \sum_{m=1}^{M} \sum_{n=1}^{N} I_{mn} \exp[\, jkd\ (m\tau_x\ n\tau_y)] \qquad \ldots \qquad 7.4$$

Where $k = 2\pi/\lambda$ is the wave number and I_{mn} is the weighting coefficient.

$$\begin{cases} \tau_x = \sin\theta\cos\varphi - \sin\theta_0\cos\varphi_0 \\ \tau_Y = \sin\theta\cos\varphi - \sin\theta_0\cos\varphi_0 \end{cases} \qquad \ldots \qquad 7.5$$

Where (θ_0, φ_0) is a beam pointing vector. If M = 51, N = 21, θ_0 = 30 deg., φ_0 = 20 deg, the (51 x 21) will result array antenna pattern shown in Figure 7.1.



.Figure 7.1: Pattern from multiple-element array antenna

Below figures, shows the result obtained from the algorithm, Figure 7.2, shows an amplitude and phase of the received signal, Figure 7.3, shows superimposed of coherent pulse train on the noise and clutter waveform, with completely target signal submerged under noise and clutter, and Figure 7.4, shows the contrast before (top) the cancellation signal and (bottom) after the cancellation signal has been added to radar return.

The return signal can be described as Eq. 7.6:

$$S = \left(\left| \frac{\Delta E_{\max}}{E_x} \right| \right) E_x \quad \ldots \quad 7.6$$

$$\Delta E_{\max} = E_c + E_x \quad \ldots \quad 7.8$$

Where $\Delta E_{\max}$ is the cancellation residual field, E_c Cancellation field and E_x is the target's scattering field. Complete stealth is realized when S = 0. From Figure 7.4, it can be seen that corresponding return signal, S to $\Delta E_{\max}$ (2 dB) is 5 dB, so the reduction of radar detection maximum range is 50% of original value.

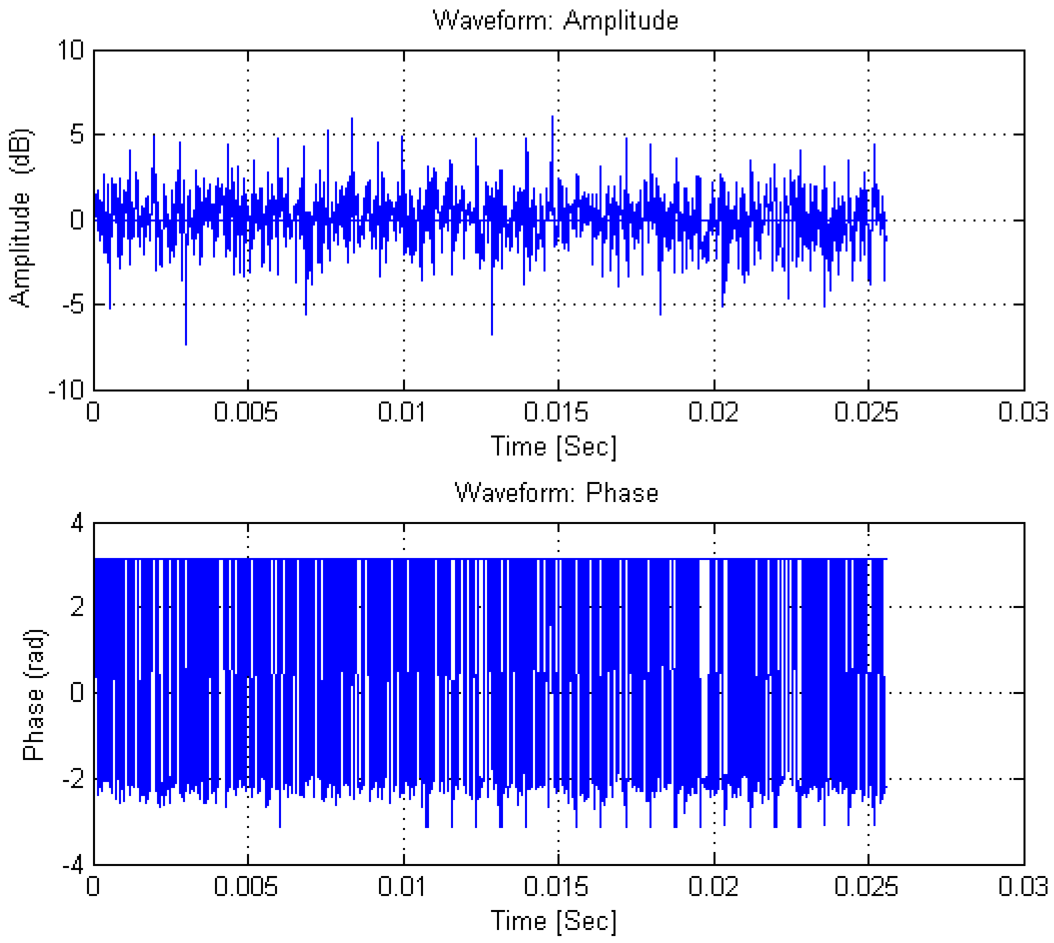


Figure 7.2: shows (top) an amplitude and (bottom) phase of the received signal

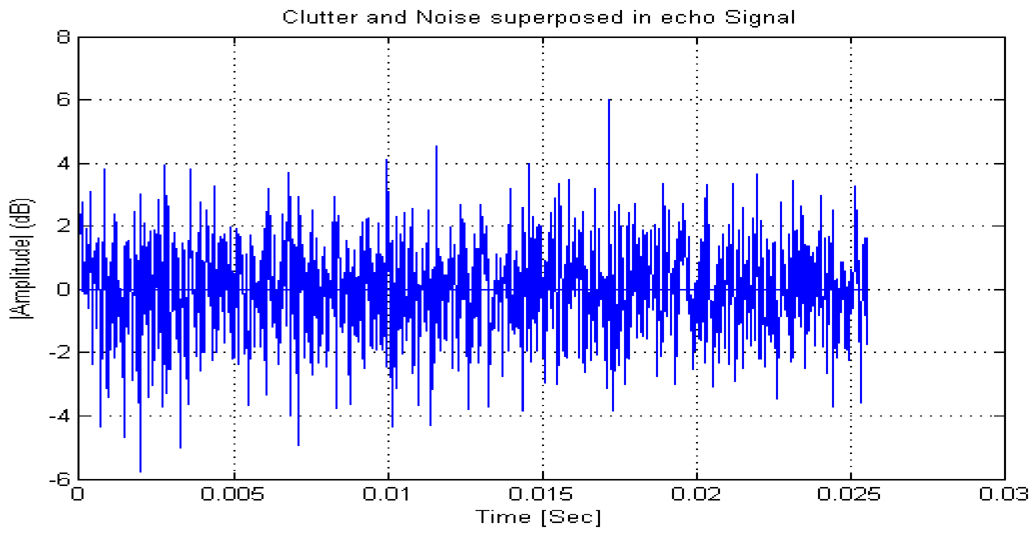


Figure 7.3: shows superimposed of coherent pulse train on the noise and clutter waveform

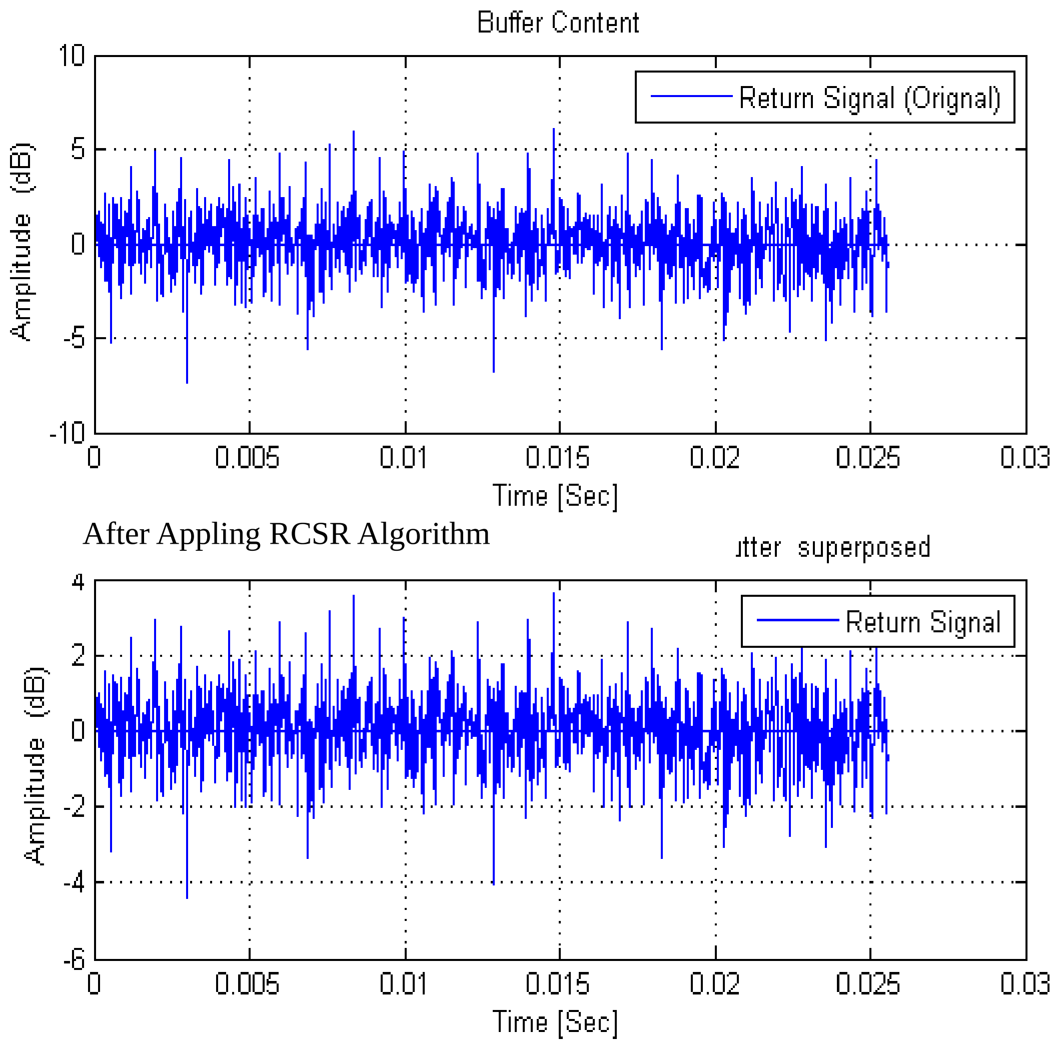


Figure 7.4: shows the contrast before (top) the cancellation signal and (bottom) after the .cancellation signal has been added to radar return

Chapter Eight

9. CHAPTER EIGHT: CONCLUSION

9.1 Conclusion:

The world orientation after World War II to develop its Combat system in an attempt to hit the fortifications of the enemy in a surprise way using fighter jets, but the radar systems and anti aircraft weapons were an obstacle without realizing it, which encouraged researchers and scientists in the field of military systems to search for ways to hide these fighter systems from enemy's radar, which known by (Stealth Technology) by reducing the radar cross section of the target and upon which the regulations radar to detect targets.

The researchers was able to development group of methods or techniques to hide Combat Systems, which are categorizes in Target shaping, Radar Absorbent Material, Passive Cancellation (Discrete loading) and Active Cancellation System, the first three types are widely use and the latter one faced some difficulties in its implementation for many reasons including the cost of economic and difficulty processing radio signals.

Due to the modern components for signal processing and rapid increase in computer speed make it possible to achieve radar visibility reduction, that requires reduce the radar cross section (RCS) of an aircraft or a system because it seems to be on the enemy's radar detection capabilities .

To achieve this goal, this research proposed an Active cancellation algorithm for radar cross section reduction using MATLAB 8.1.0.604 (R2013a), digital radio-frequency memory (DRFM), and phased array technology to generate the desired signal to cancel the reflected radar returns.

The main result of the thesis showed the possibilities of enhancing reduce the reflecting power from the target.

This approach can be used with different number of others radio echo scenarios.

Radar and stealth technologies have become significantly more advanced in the last fifty years and this trend will continue because the two technologies are against each other. It is somewhat of an arms race except it isn’t between specific countries.

9.2 Future Work:

The proposed algorithm is based on the recording time between radar pulses to calculate the pulse repetition frequency, so it may not work probably if radar constantly changing of the pulse repetition time (Staggered PRT).

The proposed algorithm is dedicated to pulsed radar, so it may not work probably if radar is Continuous Wave radar.

The proposed algorithm works on the analysis of the received pulses and make sure they have the same frequency to be handled as a signal received from the same radar, so it may not work probably if radar constantly changing of the transmitter - frequency (Frequency-Diversity).

The proposed algorithm is based on transmitting the cancellation signal in the direction of the radar transmitter, so it may not work probably in case of bistatic radar.

REFRENCES:

1. Eugene F. Knott, John F. Shaeffer, Michael T. Tuley, Radar Cross Section, SciTech Publishing Inc., 2004.
2. Bassem R. Mahafza, Radar Systems Analysis and Design using MATLAB, Chapman& Hall/CRC, 2005.
3. Bassem R. Mahafza, Atef Elsherbeni, MATLAB Simulations for Radar Systems Design, Chapman& Hall/CRC, 2004.
4. Merrill I. Skolnik, Radar Handbook, Third Edition, McGmw-Hill, 2008.
5. Electronic Warfare and Radar Systems Engineering Handbook, NAVAIR Electronic Warfare/Combat Systems, 2012.
6. Merrill I. Skolnik, Radar Handbook, First Edition, McGmw-Hill, 1990.
7. S.M. Rao, D.R. Wilton, and A.W. Glisson, “Electromagnetic scattering by surfaces of arbitrary shape”, IEEE Transactions on Antenna and Propagation, Vol. 30, No. 3, pp. 409-418, 1982.
8. Virga K.L and Rahmat-Sami Y, “RCS characterization of a finite ground plane with perforated apertures: simulations and measurements”, IEEE Transactions on Antennas and wave propagation, Vol. 42, No. 11, pp. 1491-1501, 1994.
9. E.F. Knott, J.F. Shaffer and M.T. Tuley, “Radar Cross Section”, 2nd Edition, Artech House Radar Library, 1993.
10. Paolo Brandimarte, Handbook in Monte Carlo Simulation, John Wiley & Sons, Inc., 2014.
11. http://www.mathworks.com/discovery/monte-carlo-simulation.html, Last Visit 12/12/2014.
12. Dirk P. Kroese, Thomas Taimre, Zdravko I. Botev, Handbook of Monte Carlo Methods, John Wiley & Sons, Inc., 2011.
13. M. Madheswaran, P. Suresh Kumar, “Estimation of Wide Band Radar Cross Section (RCS) of regular shaped objects using Method of Moments (MoM)”, in ICTACT Journal on Communication Technology Vol. 3, No. 2, pp 536-541, 2012.
14. G.S.N. Raju, Radar Engineering and Fundamental of Navigation Aids, I.K. International publishing house Pvt. Ltd, 2012.
15. K. Barton, Sergey A. Leonov, Radar Technology Encyclopedia, David Artech House, 1998.

Appendixes

10. APPENDIXES:

A.Radar Simulation Graphical user interface:

```
function varargout = radarSimulation(varargin)
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @radarSimulation_OpeningFcn, ...
                   'gui_OutputFcn',  @radarSimulation_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --- Executes just before radarSimulation is made visible.
function radarSimulation_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to radarSimulation (see VARARGIN)
global r;   r=0; global i;   i=0; global inputdata; global z; z=0; global p; p=0; global rang;
% Choose default command line output for radarSimulation
handles.output = hObject;
handles.FOV = [];
handles.mountains = [];
handles.IF_Freq = 3e7;
handles.currentTime = 0;
handles.targetsFigure = [];
handles.Targets = [];
handles.pulseNum = 0;
handles.plotedTargets = [];
plotDistLines(handles.radarDisplay,10);
PW_Callback(handles.PW, eventdata, handles);    % updating the current PW value
% Update handles structure
guidata(hObject, handles);
reset_Callback(hObject, eventdata, handles)
% UIWAIT makes radarSimulation wait for user response (see UIRESUME)
% uiwait(handles.figure1);
 % --- Outputs from this function are returned to the command line.
function varargout = radarSimulation_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;
% --- Executes on key press with focus on figure1 and none of its controls.
function figure1_KeyPressFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  structure with the following fields (see FIGURE)
%   Key: name of the key that was pressed, in lower case
%   Character: character interpretation of the key(s) that was pressed
%   Modifier: name(s) of the modifier key(s) (i.e., control, shift) pressed
% handles    structure with handles and user data (see GUIDATA)
tcpipClient = tcpip('127.0.0.1',500,'NetworkRole','Client');%10.1.1.2 is the Aircraft IP
set(tcpipClient,'InputBufferSize',100);
set(tcpipClient,'Timeout',30);
fopen(tcpipClient);
rawData = fread (tcpipClient,1,'double');
fclose (tcpipClient);
delete (tcpipClient);
clear tcpipClient
global i; global z; global r;global p; global rang;
isMTIused = get(handles.useMTI,'value');
            switch (rawData)
                      case 1
                                  if isMTIused
                                         i = rang;
                                         handles = createTargets1(hObject,handles,i)
                                         r=1;
                                         p=1;
                                   else
                                       handles = createTargets1(hObject,handles,i)
                                       r=1;
                                       p=1;
                                   end
                       case 0
                                   handles = createTargets1(hObject,handles,i);
                                   r=0;
                                   p=0;
           end
h = findobj(handles.radarDisplay,'type','line');
if ishandle(h) delete(h); end
h = findobj(handles.radarDisplay,'type','text');
if ishandle(h) delete(h); end
plotDistLines(handles.radarDisplay,10);
handles.numRadarTurn = 0;
handles.currentTime = 0;
set(handles.bufferAnalyze,'value',0);
switch p
              case 1
                 [NPB,PRIA,SR,DZN,EN,PWA,STA,AMP]=runRadarSim_v2(handles,p,i);
              case 0
                 [NPB,PRIA,SR,DZN,EN,PWA,STA,AMP]=runRadarSim_v3(handles,p,i);
end

function PRI_Callback(hObject, eventdata, handles)
% hObject    handle to PRI (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of PRI as text
%        str2double(get(hObject,'String')) returns contents of PRI as a double
% --- Executes during object creation, after setting all properties.
function PRI_CreateFcn(hObject, eventdata, handles)
% hObject    handle to PRI (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ZSA_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to ZSA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of ZSA as text
%        str2double(get(hObject,'String')) returns contents of ZSA as a double
% --- Executes during object creation, after setting all properties.
function ZSA_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ZSA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
 % Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on slider movement.
function PW_Callback(hObject, eventdata, handles)
    percent = get(hObject,'value');
    str = ['PW = ' num2str(percent*100) '% of the PRI'];
    set(handles.PWstr,'string',str);
% hObject    handle to PW (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
% --- Executes during object creation, after setting all properties.
function PW_CreateFcn(hObject, eventdata, handles)
% hObject    handle to PW (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
 % Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
    set(hObject,'BackgroundColor',[.9 .9 .9]);
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on button press in run.

function run_Callback(hObject, eventdata, handles)
global i; global r; global inputdata; global z; global p;
switch get(hObject,'string')
        case 'Pause'
            set(hObject,'string', 'Continue','value',0);
            return;
        case  'Start'
            if p==1
             i;
                handles = createTargets1(hObject,handles,i);
            else
                %handles = createTargets0(hObject,handles);
                 handles = createTargets1(hObject,handles,i);
end
set(handles.pushbutton12,'enable','on');
set(handles.bufferAnalyze,'value',0);
handles.currentTime = 0;
set(hObject,'string', 'Pause');
handles.FOV = [];
handles.persistentPlotHandle = [];
handles.pulseNum = 1;
guidata(hObject, handles);
```

```
plotDistLines(handles.radarDisplay,10);
set(handles.nTargets,'enable','off');
set(handles.nMountains,'enable','off');
set(handles.RCS,'enable','off');
handles.numRadarTurn = 0;
            case 'Continue'
            set(hObject,'string', 'Pause','value',1);
            end
switch p
              case 1
                [NPB,PRIA,SR,DZN,EN,PWA,STA,AMP]=runRadarSim_v2(handles,p,i);
              case 0
                [NPB,PRIA,SR,DZN,EN,PWA,STA,AMP]=runRadarSim_v3(handles,p,i);
end
inputdata = [NPB PRIA SR DZN EN PWA STA  AMP];
function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a double
% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes during object creation, after setting all properties.
function bufferSize_CreateFcn(hObject, eventdata, handles)
% hObject    handle to bufferSize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function Amp_Callback(hObject, eventdata, handles)
% hObject    handle to Amp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of Amp as text
%        str2double(get(hObject,'String')) returns contents of Amp as a double
% --- Executes during object creation, after setting all properties.
function Amp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Amp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function Th_Callback(hObject, eventdata, handles)
    Th = get(hObject,'value');
if get(handles.CFAR,'value')
        str = ['Relative Th = ' num2str(Th)];
else
        str = ['Absolute Th = ' num2str(10^(Th))];
end
set(handles.ThStr,'string',str);
% hObject    handle to Th (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of Th as text
%        str2double(get(hObject,'String')) returns contents of Th as a double
% --- Executes during object creation, after setting all properties.
function Th_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Th (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function updateRate_Callback(hObject, eventdata, handles)
% hObject    handle to updateRate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of updateRate as text
%        str2double(get(hObject,'String')) returns contents of updateRate as a double
% --- Executes during object creation, after setting all properties.
function updateRate_CreateFcn(hObject, eventdata, handles)
% hObject    handle to updateRate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on selection change in recieveChannelBWType.
function recieveChannelBWType_Callback(hObject, eventdata, handles)
% hObject    handle to recieveChannelBWType (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String') returns recieveChannelBWType contents as cell array
%        contents{get(hObject,'Value')} returns selected item from recieveChannelBWType
% --- Executes during object creation, after setting all properties.
function recieveChannelBWType_CreateFcn(hObject, eventdata, handles)
% hObject    handle to recieveChannelBWType (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function RadarBW_Callback(hObject, eventdata, handles)
```

```matlab
% hObject    handle to RadarBW (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of RadarBW as text
%        str2double(get(hObject,'String')) returns contents of RadarBW as a double
% --- Executes during object creation, after setting all properties.
function RadarBW_CreateFcn(hObject, eventdata, handles)
% hObject    handle to RadarBW (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function samplingRate_Callback(hObject, eventdata, handles)
% hObject    handle to samplingRate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of samplingRate as text
%        str2double(get(hObject,'String')) returns contents of samplingRate as a double
% --- Executes during object creation, after setting all properties.
function samplingRate_CreateFcn(hObject, eventdata, handles)
% hObject    handle to samplingRate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function antenaAperture_Callback(hObject, eventdata, handles)
% hObject    handle to antenaAperture (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of antenaAperture as text
%        str2double(get(hObject,'String')) returns contents of antenaAperture as a double
% --- Executes during object creation, after setting all properties.
function antenaAperture_CreateFcn(hObject, eventdata, handles)
% hObject    handle to antenaAperture (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function antenaMode_Callback(hObject, eventdata, handles)
% hObject    handle to antenaMode (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of antenaMode as text
%        str2double(get(hObject,'String')) returns contents of antenaMode as a double
% --- Executes during object creation, after setting all properties.
function antenaMode_CreateFcn(hObject, eventdata, handles)
% hObject    handle to antenaMode (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function edit9_CreateFcn(hObject, eventdata, handles)
% --- Executes on button press in reset.
function reset_Callback(hObject, eventdata, handles)
set(handles.run,'value',0);
global p; p=0;global i; i=0;
set(handles.pushbutton12,'value',0);
set(handles.run,'string','Start');
set(handles.pushbutton12,'string','Enable Transmission');
set(handles.pushbutton12,'enable','off');
h = findobj(handles.radarDisplay,'type','line');
if ishandle(h) delete(h); end
h = findobj(handles.radarDisplay,'type','text');
if ishandle(h) delete(h); end
handles.mountains = [];
handles.foundTargets = createTargetObj( {},{},{},{},{},{},{},{},{},{} );  % creating an empty object
guidata(hObject,handles);
handleRadarControlls(handles,'on');
set(handles.nTargets,'enable','on');
set(handles.nMountains,'enable','on');
set(handles.bufferAnalyze,'enable','off');
set(handles.bufferAnalyze,'value',0);
set(handles.RCS,'enable','on');
set(handles.scopeDisplay,'value',1);
scopeDisplay_Callback(handles.scopeDisplay, eventdata, handles)
set(gca,'xlim',[-100 100]*1e3, 'ylim', [-100 100]*1e3);
grid on;
plotDistLines(handles.radarDisplay,10);
% hObject    handle to reset (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
function nTargets_Callback(hObject, eventdata, handles)
% hObject    handle to nTargets (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of nTargets as text
%        str2double(get(hObject,'String')) returns contents of nTargets as a double
% --- Executes during object creation, after setting all properties.
function nTargets_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nTargets (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on selection change in RCS.
function RCS_Callback(hObject, eventdata, handles)
% hObject    handle to RCS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String') returns RCS contents as cell array
%        contents{get(hObject,'Value')} returns selected item from RCS
% --- Executes during object creation, after setting all properties.
function RCS_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to RCS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on button press in useMatchFilter.
function useMatchFilter_Callback(hObject, eventdata, handles)
% hObject    handle to useMatchFilter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of useMatchFilter
% --- Executes during object creation, after setting all properties.
function useMatchFilter_CreateFcn(hObject, eventdata, handles)
% hObject    handle to useMatchFilter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% --- Executes during object creation, after setting all properties.
function radarDisplay_CreateFcn(hObject, eventdata, handles)
grid on;
% hObject    handle to radarDisplay (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: place code in OpeningFcn to populate radarDisplay
% --- Executes on button press in placeMaountins.
function placeMaountins_Callback(hObject, eventdata, handles)
if get(hObject,'value') %placing mountains
set(handles.run,'visible','off');
set(handles.bufferAnalyze,'visible','off');
set(hObject,'FontWeight','bold','FontSize',10);
title(handles.radarDisplay,'click on display to place mountains');
hold on;
set(gcf,'userdata',handles);
str = ['temp = get(gca,''CurrentPoint''); h=placeClutter(get(gcf,''userdata''),temp(1,1:2)); ' ...
        'set(gcf,''userdata'',h); plot(temp(1,1),temp(1,2),''*k'',''MarkerSize'',10,''tag'',''mountain'');'];
set(handles.radarDisplay,'buttondownfcn',str);
else    % finished placing mountains
set(handles.run,'visible','on');
set(handles.bufferAnalyze,'visible','on');
hold off;
set(hObject,'FontWeight','normal','FontSize',8);
title(handles.radarDisplay,[]);
h = findobj(handles.radarDisplay,'tag','mountain');
delete(h);
set(handles.radarDisplay,'buttondownfcn',[]);
guidata(hObject, handles);
end
% hObject    handle to placeMaountins (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of placeMaountins
% --- Executes on button press in useMTI.
function useMTI_Callback(hObject, eventdata, handles)
% --- Executes on button press in displayTargets.
function displayTargets_Callback(hObject, eventdata, handles)
displayTargets(handles,'in diffrent figure')
% hObject    handle to displayTargets (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of displayTargets
```

```
function digitizerNoiseLevel_Callback(hObject, eventdata, handles)
% hObject    handle to digitizerNoiseLevel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of digitizerNoiseLevel as text
%        str2double(get(hObject,'String')) returns contents of digitizerNoiseLevel as a double
% --- Executes during object creation, after setting all properties.
function digitizerNoiseLevel_CreateFcn(hObject, eventdata, handles)
% hObject    handle to digitizerNoiseLevel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on button press in bufferAnalyze.
function bufferAnalyze_Callback(hObject, eventdata, handles)
set(handles.run,'value',1);
global r;  global i;global rang; global p;
h = findobj(handles.radarDisplay,'type','line');
if ishandle(h) delete(h); end
h = findobj(handles.radarDisplay,'type','text');
if ishandle(h) delete(h); end
plotDistLines(handles.radarDisplay,10);
handles.currentTime = 0 ;
if p==1
isMTIused = get(handles.useMTI,'value');
if isMTIused
i = rang
end
handles = createTargets1(hObject,handles,i);
else
handles = createTargets1(hObject,handles,i);
end
if p==1
       [NPB,PRIA,SR,DZN,EN,PWA,STA,AMP]=runRadarSim_v2(handles,p,i);
else
      [NPB,PRIA,SR,DZN,EN,PWA,STA,AMP]=runRadarSim_v3(handles,p,i);
end
% hObject    handle to bufferAnalyze (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- Executes on button press in waitForTarget.
function waitForTarget_Callback(hObject, eventdata, handles)
% hObject    handle to waitForTarget (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of waitForTarget
% --- Executes on selection change in backgrundColor.
function backgrundColor_Callback(hObject, eventdata, handles)
% hObject    handle to backgrundColor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String') returns backgrundColor contents as cell array
%        contents{get(hObject,'Value')} returns selected item from backgrundColor
% --- Executes during object creation, after setting all properties.
function backgrundColor_CreateFcn(hObject, eventdata, handles)
% hObject    handle to backgrundColor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on selection change in RFnoise.
function RFnoise_Callback(hObject, eventdata, handles)
% hObject    handle to RFnoise (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String') returns RFnoise contents as cell array
%        contents{get(hObject,'Value')} returns selected item from RFnoise
% --- Executes during object creation, after setting all properties.
function RFnoise_CreateFcn(hObject, eventdata, handles)
% hObject    handle to RFnoise (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on button press in CFAR.
function CFAR_Callback(hObject, eventdata, handles)
    if get(hObject,'value')
        % using relative threshold
        set(handles.Th,'min',1.1,'max',10,'value',2);
        Th_Callback(handles.Th, eventdata, handles);
    else
        % using absolute threshold
        set(handles.Th,'min',-20,'max',-1,'value',-9);
        Th_Callback(handles.Th, eventdata, handles);
    end
% --- Executes on selection change in nMountains.
function nMountains_Callback(hObject, eventdata, handles)
% hObject    handle to nMountains (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String') returns nMountains contents as cell array
%        contents{get(hObject,'Value')} returns selected item from nMountains
% --- Executes during object creation, after setting all properties.
function nMountains_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nMountains (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on selection change in stagger.
function stagger_Callback(hObject, eventdata, handles)
% hObject    handle to stagger (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String') returns stagger contents as cell array
%        contents{get(hObject,'Value')} returns selected item from stagger
% --- Executes during object creation, after setting all properties.
function stagger_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to stagger (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on button press in playSound.
function playSound_Callback(hObject, eventdata, handles)
% hObject    handle to playSound (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of playSound
% --- Executes on button press in soundOn.
function soundOn_Callback(hObject, eventdata, handles)
val = get(hObject,'value');
if val
     field = 'soundOnIcon';
else
     field = 'soundOffIcon';
end
icon = getappdata(hObject,field);
set(hObject,'units','pixels');
pos = get(hObject,'position');
set(hObject,'units','normalized');
icon = imresize( icon,[pos(4) pos(3)] );
set( hObject,'cdata',icon );
% hObject    handle to soundOn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of soundOn
% --- Executes during object creation, after setting all properties.
function soundOn_CreateFcn(hObject, eventdata, handles)
if exist('soundOn.jpg','file') & exist('soundOff.jpg','file');        % both icon files exist
soundOnIcon = imread('soundOn.jpg');
soundOffIcon = imread('soundOff.jpg');
setappdata(hObject,'soundOnIcon',soundOnIcon);
setappdata(hObject,'soundOffIcon',soundOffIcon);
set(hObject,'cdata',soundOffIcon, 'value',0);
soundOn_Callback(hObject, eventdata, handles)
else
        set(hObject,'enable','off');
end
% hObject    handle to soundOn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% --- Executes on button press in scopeDisplay.
function scopeDisplay_Callback(hObject, eventdata, handles)
val = get(hObject,'value');
    if val
        field = 'scopeOnIcon';
    else
        field = 'scopeOffIcon';
   end
icon = getappdata(hObject,field);
set(hObject,'units','pixels');
pos = get(hObject,'position');
set(hObject,'units','normalized');
icon = imresize( icon,[pos(4) pos(3)] );
set( hObject,'cdata',icon );
% hObject    handle to scopeDisplay (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of scopeDisplay
% --- Executes during object creation, after setting all properties.
function scopeDisplay_CreateFcn(hObject, eventdata, handles)
if exist('osciloscopeOn.jpg','file') & exist('osciloscopeOff.jpg','file');     % both icon files exist
scopeOnIcon = imread('osciloscopeOn.jpg');
scopeOffIcon = imread('osciloscopeOff.jpg');
setappdata(hObject,'scopeOnIcon',scopeOnIcon);
setappdata(hObject,'scopeOffIcon',scopeOffIcon);
set(hObject,'cdata',scopeOffIcon, 'value',0);
scopeDisplay_Callback(hObject, eventdata, handles)
else
set(hObject,'enable','off');
end
% hObject    handle to scopeDisplay (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% --- Executes on button press in persistentDisplay.
function persistentDisplay_Callback(hObject, eventdata, handles)
% hObject    handle to persistentDisplay (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of persistentDisplay
% --- Executes when figure1 is resized.
function figure1_ResizeFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- Executes on key press with focus on run and none of its controls.
function run_KeyPressFcn(hObject, eventdata, handles)
% hObject    handle to run (see GCBO)
% eventdata  structure with the following fields (see UICONTROL)
%   Key: name of the key that was pressed, in lower case
%   Character: character interpretation of the key(s) that was pressed
%   Modifier: name(s) of the modifier key(s) (i.e., control, shift) pressed
% handles    structure with handles and user data (see GUIDATA)
% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over run.
function run_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to run (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- Executes on button press in togglebutton4.
function togglebutton4_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of togglebutton4
% --- Executes on button press in pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
guidata(hObject, handles);
global r;global z;global p;
switch get(hObject,'string')
case 'Disable Transmission'
set(hObject,'string', 'Enable Transmission','value',1)
```

```
return;
case  'Enable Transmission'
z=1;   r=1;    p=1;
h = findobj(handles.radarDisplay,'type','line');
if ishandle(h) delete(h); end
h = findobj(handles.radarDisplay,'type','text');
if ishandle(h) delete(h); end
plotDistLines(handles.radarDisplay,10);
handles.numRadarTurn = 0;
handles.currentTime = 0;
set(handles.pushbutton12,'enable','off');
set(hObject,'string', 'Disable Transmission','value',1);
global i;  global inputdata;
s = whos('inputdata');  s.size;   s.bytes;
tcpipServer = tcpip('0.0.0.0',180,'NetworkRole','Server');
set(tcpipServer,'OutputBufferSize',500);
fopen(tcpipServer);
fwrite(tcpipServer,inputdata(:),'double');
fclose(tcpipServer);
delete (tcpipServer);
clear tcpipServer
tcpipClient = tcpip('127.0.0.1',180,'NetworkRole','Client');%10.1.1.2 is the Radar IP
set(tcpipClient,'InputBufferSize',500);
set(tcpipClient,'Timeout',30);
fopen(tcpipClient);
rawData = fread (tcpipClient,1,'double');
fclose (tcpipClient);
delete (tcpipClient);
clear tcpipClient;
i= rawData;
handles = createTargets1(hObject,handles,i);
end
set(handles.run,'value',1)
[NPB,PRIA,SR,DZN,EN,PWA,STA,AMP]=runRadarSim_v2(handles,z,i);
% --- Executes during object creation, after setting all properties.
function pushbutton12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pushbutton12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% --- Executes during object creation, after setting all properties.
function run_CreateFcn(hObject, eventdata, handles)
% hObject    handle to run (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% --- Executes during object deletion, before destroying properties.
function run_DeleteFcn(hObject, eventdata, handles)
% hObject    handle to run (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

B. Aircraft Simulation Graphical user interface:

```
function varargout = AirecraftSimulation(varargin)
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @AirecraftSimulation_OpeningFcn, ...
    'gui_OutputFcn',  @AirecraftSimulation_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
```

```matlab
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --- Executes just before AirecraftSimulation is made visible.
function AirecraftSimulation_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to AirecraftSimulation (see VARARGIN)
global r;   r = 0; global i;   i = 60000; global NPB;global PRIA; global SR; global DZN; global EN;
global PWA; global STA; global AMP; NPB = 32; PRIA = 8.0000e-04; SR = 50000; DZN = 9; EN =
13; PWA = 1.6000e-05; STA = 1; AMP = 1.0000e+10; global g; g=1;
% Choose default command line output for AirecraftSimulation
handles.output = hObject;
handles.FOV = []; handles.mountains = []; handles.IF_Freq = 3e7; handles.currentTime = 0;
handles.targetsFigure = []; handles.Targets = []; handles.pulseNum = 0; handles.plotedTargets = [];
plotDistLines(handles.radarDisplay,10);
PW_Callback(handles.PW, eventdata, handles);
guidata(hObject, handles);
reset_Callback(hObject, eventdata, handles)
% --- Outputs from this function are returned to the command line.
function varargout = AirecraftSimulation_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;
function PRI_Callback(hObject, eventdata, handles)
% hObject    handle to PRI (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of PRI as text
%        str2double(get(hObject,'String')) returns contents of PRI as a double
% --- Executes during object creation, after setting all properties.
function PRI_CreateFcn(hObject, eventdata, handles)
% hObject    handle to PRI (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ZSA_Callback(hObject, eventdata, handles)
% hObject    handle to ZSA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of ZSA as text
%        str2double(get(hObject,'String')) returns contents of ZSA as a double
% --- Executes during object creation, after setting all properties.
function ZSA_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ZSA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
```

```
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on slider movement.
function PW_Callback(hObject, eventdata, handles)
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
% --- Executes during object creation, after setting all properties.
function PW_CreateFcn(hObject, eventdata, handles)
% hObject    handle to PW (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
    set(hObject,'BackgroundColor',[.9 .9 .9]);
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on button press in run.
function run_Callback(hObject, eventdata, handles)
global i; global r; global NPB;global PRIA; global SR; global DZN; global EN; global PWA; global
STA; global AMP; global g;
switch get(hObject,'string')
case 'Pause'
set(hObject,'string', 'Continue','value',0);
return;
case  'Start'
if r==1
i;
handles = createTargets1(hObject,handles,i);
else
handles = createTargets0(hObject,handles);
end
handles.currentTime = 0;
set(hObject,'string', 'Pause');
set(handles.pushbutton12,'enable','on');
handles.FOV = [];
handles.persistentPlotHandle = [];
handles.pulseNum = 1;
guidata(hObject, handles);
plotDistLines(handles.radarDisplay,10);
set(handles.nTargets,'enable','off');
set(handles.nMountains,'enable','off');
set(handles.RCS,'enable','off');
handles.numRadarTurn = 0;
case 'Continue'
set(hObject,'string', 'Pause','value',1);
end
runAircraftSim_v2(handles,NPB,PRIA,SR,DZN,EN,PWA,STA,AMP,i);
function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a double
% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes during object creation, after setting all properties.
function bufferSize_CreateFcn(hObject, eventdata, handles)
% hObject    handle to bufferSize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function Amp_Callback(hObject, eventdata, handles)
% hObject    handle to Amp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of Amp as text
%        str2double(get(hObject,'String')) returns contents of Amp as a double
% --- Executes during object creation, after setting all properties.
function Amp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Amp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function Th_Callback(hObject, eventdata, handles)
Th = get(hObject,'value');
if get(handles.CFAR,'value')
    str = ['Relative Th = ' num2str(Th)];
else
    str = ['Absolute Th = ' num2str(10^(Th))];
end
set(handles.ThStr,'string',str);
% hObject    handle to Th (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of Th as text
%        str2double(get(hObject,'String')) returns contents of Th as a double
% --- Executes during object creation, after setting all properties.
function Th_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Th (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function updateRate_Callback(hObject, eventdata, handles)
% hObject    handle to updateRate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of updateRate as text
%        str2double(get(hObject,'String')) returns contents of updateRate as a double
% --- Executes during object creation, after setting all properties.
function updateRate_CreateFcn(hObject, eventdata, handles)
% hObject    handle to updateRate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on selection change in recieveChannelBWType.
function recieveChannelBWType_Callback(hObject, eventdata, handles)
% hObject    handle to recieveChannelBWType (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String') returns recieveChannelBWType contents as cell array
%        contents{get(hObject,'Value')} returns selected item from recieveChannelBWType
% --- Executes during object creation, after setting all properties.
function recieveChannelBWType_CreateFcn(hObject, eventdata, handles)
% hObject    handle to recieveChannelBWType (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function RadarBW_Callback(hObject, eventdata, handles)
% hObject    handle to RadarBW (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of RadarBW as text
%        str2double(get(hObject,'String')) returns contents of RadarBW as a double
% --- Executes during object creation, after setting all properties.
function RadarBW_CreateFcn(hObject, eventdata, handles)
% hObject    handle to RadarBW (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function samplingRate_Callback(hObject, eventdata, handles)
% hObject    handle to samplingRate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of samplingRate as text
%        str2double(get(hObject,'String')) returns contents of samplingRate as a double
% --- Executes during object creation, after setting all properties.
function samplingRate_CreateFcn(hObject, eventdata, handles)
% hObject    handle to samplingRate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function antenaAperture_Callback(hObject, eventdata, handles)
% hObject    handle to antenaAperture (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of antenaAperture as text
%        str2double(get(hObject,'String')) returns contents of antenaAperture as a double
% --- Executes during object creation, after setting all properties.
function antenaAperture_CreateFcn(hObject, eventdata, handles)
% hObject    handle to antenaAperture (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function antenaMode_Callback(hObject, eventdata, handles)
% hObject    handle to antenaMode (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of antenaMode as text
%        str2double(get(hObject,'String')) returns contents of antenaMode as a double
% --- Executes during object creation, after setting all properties.
function antenaMode_CreateFcn(hObject, eventdata, handles)
% hObject    handle to antenaMode (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function edit9_CreateFcn(hObject, eventdata, handles)
% --- Executes on button press in reset.
function reset_Callback(hObject, eventdata, handles)
set(handles.run,'value',0);
set(handles.pushbutton12,'value',0);
set(handles.run,'string','Start');
set(handles.pushbutton12,'string','Enable Transmission');
set(handles.pushbutton12,'enable','off');
set(handles.stealth,'string', 'Enable Stealth')
set(handles.pushbutton14,'enable','off');
set(handles.stealth,'enable','off');
set(handles.togglebutton4,'value',0);
togglebutton4_Callback(handles.togglebutton4, eventdata, handles)
h = findobj(handles.radarDisplay,'type','line');
if ishandle(h) delete(h); end
h = findobj(handles.radarDisplay,'type','text');
if ishandle(h) delete(h); end
handles.mountains = [];
handles.foundTargets = createTargetObj( {},{},{},{},{},{},{},{},{},{} );
guidata(hObject,handles);
handleRadarControlls(handles,'on');
set(handles.nTargets,'enable','on');
set(handles.nMountains,'enable','on');
set(handles.bufferAnalyze,'value',0);
set(handles.bufferAnalyze,'enable','off');
set(handles.RCS,'enable','on');
set(handles.scopeDisplay,'value',1);
scopeDisplay_Callback(handles.scopeDisplay, eventdata, handles)
set(gca,'xlim',[-100 100]*1e3, 'ylim', [-100 100]*1e3);
grid on;
plotDistLines(handles.radarDisplay,10);
% hObject    handle to reset (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
function nTargets_Callback(hObject, eventdata, handles)
% hObject    handle to nTargets (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of nTargets as text
%        str2double(get(hObject,'String')) returns contents of nTargets as a double
% --- Executes during object creation, after setting all properties.
function nTargets_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nTargets (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on selection change in RCS.
function RCS_Callback(hObject, eventdata, handles)
% hObject    handle to RCS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String') returns RCS contents as cell array
%        contents{get(hObject,'Value')} returns selected item from RCS
% --- Executes during object creation, after setting all properties.
function RCS_CreateFcn(hObject, eventdata, handles)
% hObject    handle to RCS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on button press in useMatchFilter.
function useMatchFilter_Callback(hObject, eventdata, handles)
% hObject    handle to useMatchFilter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of useMatchFilter
% --- Executes during object creation, after setting all properties.
function useMatchFilter_CreateFcn(hObject, eventdata, handles)
% hObject    handle to useMatchFilter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% --- Executes during object creation, after setting all properties.
function radarDisplay_CreateFcn(hObject, eventdata, handles)
grid on;
% hObject    handle to radarDisplay (see GCBO)
```

```matlab
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: place code in OpeningFcn to populate radarDisplay
% --- Executes on button press in placeMaountins.
function placeMaountins_Callback(hObject, eventdata, handles)
if get(hObject,'value') %placing mountains
set(handles.run,'visible','off');
set(handles.bufferAnalyze,'visible','off');
set(hObject,'FontWeight','bold','FontSize',10);
title(handles.radarDisplay,'click on display to place mountains');
hold on;
set(gcf,'userdata',handles);
str = ['temp = get(gca,''CurrentPoint''); h=placeClutter(get(gcf,''userdata''),temp(1,1:2)); ' ...
       'set(gcf,''userdata'',h); plot(temp(1,1),temp(1,2),''*k'',''MarkerSize'',10,''tag'',''mountain'');'];
set(handles.radarDisplay,'buttondownfcn',str);
else    % finished placing mountains
set(handles.run,'visible','on');
set(handles.bufferAnalyze,'visible','on');
hold off;
set(hObject,'FontWeight','normal','FontSize',8);
title(handles.radarDisplay,[]);
h = findobj(handles.radarDisplay,'tag','mountain');
delete(h);
set(handles.radarDisplay,'buttondownfcn',[]);
guidata(hObject, handles);
end
% hObject    handle to placeMaountins (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of placeMaountins
% --- Executes on button press in useMTI.
function useMTI_Callback(hObject, eventdata, handles)
% --- Executes on button press in displayTargets.
function displayTargets_Callback(hObject, eventdata, handles)
displayTargets(handles,'in diffrent figure')
% hObject    handle to displayTargets (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of displayTargets
function digitizerNoiseLevel_Callback(hObject, eventdata, handles)
% hObject    handle to digitizerNoiseLevel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of digitizerNoiseLevel as text
%        str2double(get(hObject,'String')) returns contents of digitizerNoiseLevel as a double
% --- Executes during object creation, after setting all properties.
function digitizerNoiseLevel_CreateFcn(hObject, eventdata, handles)
% hObject    handle to digitizerNoiseLevel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on button press in bufferAnalyze.
function bufferAnalyze_Callback(hObject, eventdata, handles)
%set(handles.run,'value',0);
set(handles.run,'value',1);
global r; global g; g=1; r=1; global i;
h = findobj(handles.radarDisplay,'type','line');
if ishandle(h) delete(h); end
```

```
h = findobj(handles.radarDisplay,'type','text');
if ishandle(h) delete(h); end
plotDistLines(handles.radarDisplay,10);
handles.currentTime = 0 ;
handles = createTargets1(hObject,handles,i);
run_Callback(handles.run, eventdata, handles)
% hObject    handle to bufferAnalyze (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- Executes on button press in waitForTarget.
function waitForTarget_Callback(hObject, eventdata, handles)
% hObject    handle to waitForTarget (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of waitForTarget
% --- Executes on selection change in backgrundColor.
function backgrundColor_Callback(hObject, eventdata, handles)
% hObject    handle to backgrundColor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String') returns backgrundColor contents as cell array
%        contents{get(hObject,'Value')} returns selected item from backgrundColor
% --- Executes during object creation, after setting all properties.
function backgrundColor_CreateFcn(hObject, eventdata, handles)
% hObject    handle to backgrundColor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on selection change in RFnoise.
function RFnoise_Callback(hObject, eventdata, handles)
% hObject    handle to RFnoise (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String') returns RFnoise contents as cell array
%        contents{get(hObject,'Value')} returns selected item from RFnoise
% --- Executes during object creation, after setting all properties.
function RFnoise_CreateFcn(hObject, eventdata, handles)
% hObject    handle to RFnoise (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on button press in CFAR.
function CFAR_Callback(hObject, eventdata, handles)
if get(hObject,'value')
set(handles.Th,'min',1.1,'max',10,'value',2);
Th_Callback(handles.Th, eventdata, handles);
else
set(handles.Th,'min',-20,'max',-1,'value',-9);
Th_Callback(handles.Th, eventdata, handles);
end
% --- Executes on selection change in nMountains.
function nMountains_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to nMountains (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String') returns nMountains contents as cell array
%        contents{get(hObject,'Value')} returns selected item from nMountains
% --- Executes during object creation, after setting all properties.
function nMountains_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nMountains (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on selection change in stagger.
function stagger_Callback(hObject, eventdata, handles)
% hObject    handle to stagger (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String') returns stagger contents as cell array
%        contents{get(hObject,'Value')} returns selected item from stagger
% --- Executes during object creation, after setting all properties.
function stagger_CreateFcn(hObject, eventdata, handles)
% hObject    handle to stagger (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on button press in playSound.
function playSound_Callback(hObject, eventdata, handles)
% hObject    handle to playSound (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of playSound
% --- Executes on button press in soundOn.
function soundOn_Callback(hObject, eventdata, handles)
val = get(hObject,'value');
if val
field = 'soundOnIcon';
else
field = 'soundOffIcon';
end
icon = getappdata(hObject,field);
set(hObject,'units','pixels');
pos = get(hObject,'position');
set(hObject,'units','normalized');
icon = imresize( icon,[pos(4) pos(3)] );
set( hObject,'cdata',icon );
% hObject    handle to soundOn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of soundOn
% --- Executes during object creation, after setting all properties.
function soundOn_CreateFcn(hObject, eventdata, handles)
if exist('soundOn.jpg','file') & exist('soundOff.jpg','file');    % both icon files exist
soundOnIcon = imread('soundOn.jpg');
```

```
soundOffIcon = imread('soundOff.jpg');
setappdata(hObject,'soundOnIcon',soundOnIcon);
setappdata(hObject,'soundOffIcon',soundOffIcon);
set(hObject,'cdata',soundOffIcon, 'value',0);
soundOn_Callback(hObject, eventdata, handles)
else
    set(hObject,'enable','off');
end
% hObject    handle to soundOn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% --- Executes on button press in scopeDisplay.
function scopeDisplay_Callback(hObject, eventdata, handles)
% hObject    handle to scopeDisplay (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of scopeDisplay
val = get(hObject,'value');
if val
field = 'scopeOnIcon';
else
field = 'scopeOffIcon';
end
icon = getappdata(hObject,field);
set(hObject,'units','pixels');
pos = get(hObject,'position');
set(hObject,'units','normalized');
icon = imresize( icon,[pos(4) pos(3)] );
set( hObject,'cdata',icon );
% --- Executes during object creation, after setting all properties.
function scopeDisplay_CreateFcn(hObject, eventdata, handles)
if exist('ociloscopeOn.jpg','file') & exist('ociloscopeOff.jpg','file');    % both icon files exist
scopeOnIcon = imread('ociloscopeOn.jpg');
scopeOffIcon = imread('ociloscopeOff.jpg');
setappdata(hObject,'scopeOnIcon',scopeOnIcon);
setappdata(hObject,'scopeOffIcon',scopeOffIcon);
set(hObject,'cdata',scopeOffIcon, 'value',0);
scopeDisplay_Callback(hObject, eventdata, handles)
else
set(hObject,'enable','off');
end
% hObject    handle to scopeDisplay (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% --- Executes on button press in persistentDisplay.
function persistentDisplay_Callback(hObject, eventdata, handles)
% hObject    handle to persistentDisplay (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of persistentDisplay
% --- Executes when figure1 is resized.
function figure1_ResizeFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- Executes on key press with focus on run and none of its controls.
function run_KeyPressFcn(hObject, eventdata, handles)
% hObject    handle to run (see GCBO)
% eventdata  structure with the following fields (see UICONTROL)
%   Key: name of the key that was pressed, in lower case
%   Character: character interpretation of the key(s) that was pressed
%   Modifier: name(s) of the modifier key(s) (i.e., control, shift) pressed
% handles    structure with handles and user data (see GUIDATA)
% --- Executes on button press in pushbutton7.
```

```
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over run.
function run_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to run (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- Executes on button press in togglebutton4.
function togglebutton4_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
valtest = get(hObject,'value');
if valtest
field = 'EnableIcon';
else
field = 'DisableIcon';
end
icon = getappdata(hObject,field);
set(hObject,'units','pixels');
pos = get(hObject,'position');
set(hObject,'units','normalized');
icon = imresize( icon,[pos(4) pos(3 )]);
set( hObject,'cdata',icon );
set( hObject,'cdata',icon );
% Hint: get(hObject,'Value') returns toggle state of togglebutton4
% --- Executes on button press in pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global NPB;global PRIA; global SR; global DZN; global EN; global PWA; global STA; global AMP;
switch get(hObject,'string')
case 'Disable Transmission'
set(hObject,'string', 'Enable Transmission','value',1)
set(handles.pushbutton14,'enable','off');
set(handles.stealth,'enable','off');
return;
case  'Enable Transmission'
open('AIRCRAFT.EXE');
set(handles.pushbutton14,'enable','on');
set(handles.stealth,'enable','on');
h = findobj(handles.radarDisplay,'type','line');
if ishandle(h) delete(h); end
h = findobj(handles.radarDisplay,'type','text');
if ishandle(h) delete(h); end
plotDistLines(handles.radarDisplay,10);
handles.numRadarTurn = 0;
handles.currentTime = 0;
set(handles.pushbutton12,'enable','off');
tcpipClient = tcpip('127.0.0.1',180,'NetworkRole','Client');%127.0.0.1 is the Radar IP
set(tcpipClient,'InputBufferSize',500);
set(tcpipClient,'Timeout',100);
fopen(tcpipClient);
rawData = fread (tcpipClient,8,'double');
fclose (tcpipClient);
delete (tcpipClient);
clear tcpipClient;
NPB = rawData(1);PRIA = rawData(2); SR = rawData(3); DZN = rawData(4); EN = rawData(5);
PWA = rawData(6); STA = rawData(7); AMP = rawData(8); global i;
tcpipServer = tcpip('0.0.0.0',180,'NetworkRole','Server');
```

```
set(tcpipServer,'OutputBufferSize',500);
fopen(tcpipServer);
fwrite(tcpipServer,i(:),'double');
fclose(tcpipServer);
delete (tcpipServer);
clear tcpipServer
handles = createTargets1(hObject,handles,i);
guidata(hObject, handles);
axes(handles.axes4);
h = phased.LinearFMWaveform;
% Set the values of the properties
h.SampleRate =  5e7;
h.PulseWidth = 1.6e-05;
h.PRF = 1/PRIA;
h.SweepBandwidth = 5e6;
h.NumPulses = 3;
Fs = h.SampleRate;
x = step(h);
l = (0:length(x)-1)/Fs;
plot(l,log10(AMP)*x);
ylabel('Amplitude  (dB)');xlabel('[Sec]');
end
set(handles.bufferAnalyze,'enable','off');
runAircraftSim_v2(handles,NPB,PRIA,SR,DZN,EN,PWA,STA,AMP,i);
% --- Executes during object creation, after setting all properties.
function pushbutton12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pushbutton12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% --- Executes during object creation, after setting all properties.
function run_CreateFcn(hObject, eventdata, handles)
% hObject    handle to run (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% --- Executes on button press in pushbutton14.
function pushbutton14_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.bufferAnalyze,'value',1);
global g; g=0;
bufferAnalyze_Callback(handles.bufferAnalyze, eventdata, handles);


% --- Executes during object creation, after setting all properties.
function togglebutton4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to togglebutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
if exist('Enable.jpg','file') & exist('Disable.jpg','file');        % both icon files exist
EnableIcon = imread('Enable.jpg');
DisableIcon = imread('Disable.jpg');
setappdata(hObject,'EnableIcon',EnableIcon);
setappdata(hObject,'DisableIcon',DisableIcon);
set(hObject,'cdata',DisableIcon, 'value',0)
togglebutton4_Callback(hObject, eventdata, handles)
else
set(hObject,'enable','off');
end
% --- Executes on button press in stealth.
function stealth_Callback(hObject, eventdata, handles)
% hObject    handle to stealth (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
switch get(hObject,'string')
case 'Disable Stealth'
set(hObject,'string', 'Enable Stealth','value',1);
set(handles.togglebutton4,'value',0);
togglebutton4_Callback(handles.togglebutton4, eventdata, handles)
data =1;
tcpipServer = tcpip('0.0.0.0',500,'NetworkRole','Server');
set(tcpipServer,'OutputBufferSize',100);
fopen(tcpipServer);
fwrite(tcpipServer,data(:),'double');
fclose(tcpipServer);
delete (tcpipServer);
clear tcpipServer
case  'Enable Stealth'
set(hObject,'string', 'Disable Stealth','value',1);
set(handles.togglebutton4,'value',1);
togglebutton4_Callback(handles.togglebutton4, eventdata, handles)
data = 0;
tcpipServer = tcpip('0.0.0.0',500,'NetworkRole','Server');
set(tcpipServer,'OutputBufferSize',100);
fopen(tcpipServer);
fwrite(tcpipServer,data(:),'double');
fclose(tcpipServer);
delete (tcpipServer);
clear tcpipServer
end
% --- Executes during object creation, after setting all properties.
function miniDisplay_CreateFcn(hObject, eventdata, handles)
% hObject    handle to miniDisplay (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: place code in OpeningFcn to populate miniDisplay
% --- Executes during object creation, after setting all properties.
function axes4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: place code in OpeningFcn to populate axes4
% --- Executes during object deletion, before destroying properties.
function run_DeleteFcn(hObject, eventdata, handles)
% hObject    handle to run (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- Executes on key press with focus on figure1 and none of its controls.
function figure1_KeyPressFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  structure with the following fields (see FIGURE)
%   Key: name of the key that was pressed, in lower case
%   Character: character interpretation of the key(s) that was pressed
%   Modifier: name(s) of the modifier key(s) (i.e., control, shift) pressed
% handles    structure with handles and user data (see GUIDATA)
global i;i = 60000;
% --- Executes on button press in togglebutton5.
function togglebutton5_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of togglebutton5
```

C.runAircraftSim_v2

```
function runAircraftSim_v2(handles,NPB,PRIA,SR,DZN,EN,PWA,STA,AMP,k)
warning off all
displayTargets(handles,'in Aire craft radar display')
```

```
isAnalyzeBufferMode = get(handles.bufferAnalyze,'value');
nPRI = NPB;
freqRes = nPRI * 2;
freqInd = fftshift( 1:freqRes );
handles.antenaGain = buildAntenaGain(handles);
figure(handles.figure1);
currentTime = handles.currentTime;
miniDisplayUpdateTime = 0;
pulseNum = handles.pulseNum;
PRI = PRIA; PRI was entered in msec
temp = get(handles.ZSA,'string');
ind = get(handles.ZSA,'value');
antenaTurnVelocity = str2num( temp{ind} ) ;
temp = get(handles.updateRate,'string');
ind = get(handles.updateRate,'value');
updateRate = str2double( temp{ind} );
Fs = SR; % Sampling rate
lastUpdate = currentTime;
antenaGain = handles.antenaGain;
temp = get(handles.RadarBW,'string');
radarBWOptions = zeros( length(temp),1 );
for n=1 :length(temp)
    radarBWOptions(n) = str2double( temp(n) );
end
PRISize = PRI*Fs;   %number of samples in PRI
bufferSize = round(PRISize*nPRI); %number of samples in buffer
if bufferSize > 1e5
set(handles.run,'string','Pause');
radarSimulation('run_Callback',handles.run,[],guidata(handles.run));
ERRORDLG('requested PRI, buffer size & sampling rate require too large a vector for
simulation !!!');
end
returnPulses = zeros( bufferSize,1);
numRadarTurn = handles.numRadarTurn;
randVecLength = 1e6;
randVec = randn(randVecLength,1);
if Fs > 44e3
soundFs = 11e3;
sound2SampleRatio = round(Fs/soundFs);
else
soundFs = 11e3;
sound2SampleRatio = 1;
end
player = audioplayer(0, soundFs);
% Digitizer Noise
n = DZN; % Digitizer Noise
temp = get(handles.digitizerNoiseLevel,'string');
if strcmp(temp{n},'off')
digitizerNoiseLevel = 0;
else
n=str2double(temp{n});
digitizerNoiseLevel = 10^n;
end
% RF noise
n = EN; %Elecromagnatic Noise
temp = get(handles.RFnoise,'string');
if strcmp(temp{n},'off')
RFnoiseLevel = 0;
else
n=str2double(temp{n});
RFnoiseLevel = 10^n;
end
hold on;
PW =  PWA; %     Pule width
```

```
PWn = round( PW*Fs );   PWn = max(1,PWn);
transmisionInd = zeros(PWn*nPRI,1);
pulseTransmitionPoints = ones( nPRI,1 );
transmisionInd( 1:PWn ) = (1:PWn);
temp = STA;
options = get(handles.stagger,'string');
stagger = str2double( options{temp} );
ratio = 2*PW / PRI;
ratio = max(ratio, 0.05);
intervals = (1+( (0:stagger-1)-(stagger-1)/2)*ratio)*PRI;
numSamplesInInterval = round( intervals*Fs );
numSamplesInInterval(end) = round( numSamplesInInterval(end)-
mod(sum(numSamplesInInterval),PRISize));
numIntervals = length(intervals);
numSamplesInPRI = round( sum( intervals )*Fs );
maxDist = numSamplesInPRI*150/Fs*1e6;
for n=2:nPRI
interval = numSamplesInInterval(mod(n-1,numIntervals)+1);
transmisionInd( (n-1)*PWn+1:n*PWn ) = transmisionInd( (n-2)*PWn+1:(n-1)*PWn ) + interval;
pulseTransmitionPoints( n ) = transmisionInd( (n-1)*PWn+1 );
end
rangeCellInd = zeros(nPRI,numSamplesInPRI);
for n = 1 : nPRI
rangeCellInd(n,:) = transmisionInd( (n-1)*PWn+1 );
end
temp = repmat( 0:numSamplesInPRI-1,nPRI, 1 );
rangeCellInd = rangeCellInd + temp;
rangeCellInd = mod( rangeCellInd-1,bufferSize ) + 1;
rangeCellInd = rangeCellInd';
targetsTime = 0;
rangeRes = PWn; rangeRes = rangeRes + ~mod(rangeRes,2);
dilateKer = zeros(max(numSamplesInInterval)+rangeRes-1,1);
dilateKer(1:rangeRes) = 1;
dilateKer(1:3) = 1; %In case PWn == 1
stagger = length(numSamplesInInterval);
for n=1:stagger-1
for m = 1:stagger
ind = mod( m:m+n-1, stagger )+1;
offset = numSamplesInPRI - sum(numSamplesInInterval(ind));
dilateKer(offset:offset+PWn) = 1;
end
end
dilateKer = [dilateKer(end:-1:rangeRes+1) ; dilateKer];
angleRes = 1.1*2*pi*(nPRI*PRI)*(antenaTurnVelocity/2/pi);
rangeRes = PW*3e8/2;
velocityRes = 3e8/(handles.IF_Freq)/ nPRI*1000/2;
foundTargetInBuffer = 0;
radarSector = antenaTurnVelocity*updateRate;  % a buffer sector
updatedDisplay = false;
tic;
lastUpdateRealTime = toc;
% ----------------------------- Start the Aircraft radar scan ---------------------------------
while get(handles.run,'value')
if get(handles.CFAR,'value')
useCFAR = true;
CFAR = get(handles.Th,'value');
else
useCFAR = false;
Th = 10^(get(handles.Th,'value'));
end
Amp = AMP;
currentTime = currentTime+PRI;
radarAngle = mod(antenaTurnVelocity*currentTime,2*pi); %radar angle in the middle of the buffer
% -------------------------------- Finding the return pulses -------------------------------------------
```

```
targets = [handles.Targets ; handles.mountains];
curentReturnPulses = zeros( bufferSize,1 );
if ~isempty(targets)
[t a phi] = targetsReturn(targets,antenaGain,Amp, currentTime,antenaTurnVelocity,
targetsTime,handles.IF_Freq);
tIn = round( t*Fs );
tIn = max(tIn,1);   tIn = min(tIn,bufferSize);
for n=1:length(t)
curentReturnPulses(tIn(n):tIn(n)+PWn-1) = curentReturnPulses(tIn(n):tIn(n)+PWn-1) +
a(n)*exp(i*phi(n));
end
end
transmitInd = pulseTransmitionPoints( mod(pulseNum-1,nPRI)+1 );
index = transmitInd : transmitInd + bufferSize-1;
index( bufferSize-transmitInd + 2 : bufferSize ) = 1:transmitInd-1;
returnPulses(index) = returnPulses(index)+curentReturnPulses;
% -------------------------------- Processing Buffer ----------------------------------------------
if ~mod(pulseNum,nPRI)  %Only processing every N number of pulses
recievedSignal = returnPulses;
n = get(handles.RadarBW,'value');
radarBW=radarBWOptions(n) * 1e6;
%  ------------------------- Creating RF noise --------------------------------------------------
if RFnoiseLevel
RFnoise=wgn(length(returnPulses),1,0);
recievedSignal = recievedSignal+1e-8*RFnoise;
end
a = -pi^2*radarBW^2/log(0.5)*log(exp(1));
responseStart = sqrt(-log(0.1)/log(exp(1))/a);
t = -responseStart : 1/Fs : responseStart;
response = exp(-t'.^2*a);
response = response/sum(response);
recievedSignal = conv2(recievedSignal,response,'same');
%  ------------------------- Creating Digitizer noise -----------------------------------------------
if digitizerNoiseLevel
DigiNoise = fastSemiRandn( bufferSize )* digitizerNoiseLevel * [1 ; i];
end
recievedSignal(transmisionInd) = 0;
returnPulses = zeros( bufferSize,1);    % Cleaning the pulses buffer
if get(handles.useMatchFilter,'value');
if length(response) > PWn
matchFilter = response;
else
matchFilter = ones(PWn,1)/PWn;
end
processedRecivedSignal = conv2(recievedSignal,matchFilter,'same');
else
processedRecivedSignal = recievedSignal;
end
% ------------------------------ Was there a target ------------------------------------------------
signalInRangeCells = processedRecivedSignal(rangeCellInd);
isMTIused = get(handles.useMTI,'value');
if isMTIused
freqInRangeCells = fft( signalInRangeCells,freqRes,2 );
energyInFreqRangeCells = abs( freqInRangeCells(:,freqInd) ).^2;
if useCFAR
noiseLevel = median( energyInFreqRangeCells(:) );    % Th is per frequency
freqTh = noiseLevel * CFAR * nPRI;
else
freqTh = Th * nPRI;
end
% Find in each range cell the maximum frequency (I only allow one target per range cell)
[maxFreq maxFreqInd] = max( energyInFreqRangeCells,[],2);
localMaxEnergy = imdilate( maxFreq , dilateKer );
rangeCell = find( localMaxEnergy == maxFreq & maxFreq > freqTh );
```

```
targetInd = sub2ind( [numSamplesInPRI freqRes], rangeCell, maxFreqInd( rangeCell ) );
else
energyInRangeCells = sum( abs(signalInRangeCells).^2,2 );
if useCFAR
noiseLevel = median( energyInRangeCells );
Th = noiseLevel * CFAR;
end
% Thresholding to find targets
localMaxEnergy = imdilate(energyInRangeCells,dilateKer);
targetInd = find ( energyInRangeCells == localMaxEnergy & energyInRangeCells > Th);
end
 if ~isempty(targetInd)
 foundTargetInBuffer = 1;
 if isMTIused
[rangeCell ans] = ind2sub(size(energyInFreqRangeCells),targetInd(:));
RCS = energyInFreqRangeCells(targetInd(:));
else
rangeCell = targetInd;
RCS = energyInRangeCells(targetInd);
end
targetRange = ( rangeCell -PWn/2) / Fs / 2 * 3e8;   %target ranges
targetV = ones(length(targetInd),1)*12345;
for Rind = 1:length(targetInd)
%Processing each Target
pos = [k 0];
if isMTIused
% Calculating the targets velocity acording to its dopler frequency
targetV(Rind)
=TIcalcVelocityFromFourier(energyInFreqRangeCells,targetInd(Rind),freqRes,handles.IF_Freq,PRI
);
end
if isempty(handles.foundTargets)
% adding the new target to the foundTargets structure
handles.foundTargets(end+1) = ...
                          createTargetObj( pos, RCS(Rind), targetRange(Rind), targetV(Rind),
radarAngle,0, radarAngle,radarAngle,[],numRadarTurn );
h = plotTarget( handles.foundTargets(end), handles );
handles.foundTargets(end).hPlot = h;
else
M = length(handles.foundTargets);
dAngle = zeros(M,1);
dRange = zeros(M,1);
dV = zeros(M,1);
for mm=1:M
d1 = calcDiffAngle( handles.foundTargets(mm).counterClockWise, radarAngle );
d2 = pi;
dAngle(mm) = min ( d1, d2)/angleRes;
dRange(mm) = abs( handles.foundTargets(mm).R - targetRange(Rind) )/rangeRes;
dV(mm) = abs( handles.foundTargets(mm).v - targetV(Rind) )/velocityRes ;
end
 [sameTargetScore sameTargetInd] = min( dAngle.^2 + dRange.^2 + dV.^2 );
if  sameTargetScore < 1
if handles.foundTargets(sameTargetInd).RCS < RCS(Rind)
handles.foundTargets(sameTargetInd).pos = pos;
handles.foundTargets(sameTargetInd).RCS = RCS(Rind);
handles.foundTargets(sameTargetInd).R = targetRange(Rind);
handles.foundTargets(sameTargetInd).v = targetV(Rind);
handles.foundTargets(sameTargetInd).angle = radarAngle;
delete( handles.foundTargets(sameTargetInd).hPlot );
h = plotTarget( handles.foundTargets(sameTargetInd), handles );
handles.foundTargets(sameTargetInd).hPlot = h;
end
if d1 < d2 % new target is before old target
handles.foundTargets(sameTargetInd).counterClockWise = radarAngle;
```

```
else
handles.foundTargets(sameTargetInd).clockWise = radarAngle;
end
else
handles.foundTargets(end+1) = ...
createTargetObj( pos, RCS(Rind), targetRange(Rind), targetV(Rind), radarAngle,0,
radarAngle,radarAngle,[],numRadarTurn );
h = plotTarget( handles.foundTargets(end), handles );
handles.foundTargets(end).hPlot = h;
end
end
end
end
dt = currentTime-lastUpdate ;
if dt > updateRate  %Is it time to update the display
lastUpdate = currentTime;
currentRealTime = toc;
delay = dt - (currentRealTime - lastUpdateRealTime); % syncronizing to real time
lastUpdateRealTime = currentRealTime;
if delay > 0
pause(delay);
end
isPersistentDisplay = get(handles.persistentDisplay,'value');
handles =
plotFOV(handles,currentTime,antenaTurnVelocity,radarSector,maxDist,isPersistentDisplay);
updatedDisplay = true;
% updating the mini-map display every 5 seconds
if currentTime > miniDisplayUpdateTime + 5
if ~get( handles.scopeDisplay,'value' )
miniDisplayUpdateTime = currentTime;
displayTargets(handles,'in radar display')
numRadarTurn = ceil(currentTime*antenaTurnVelocity/2/pi);
end
if isPersistentDisplay
if length( handles.persistentPlotHandle ) > 1000
temp = handles.persistentPlotHandle;
delete( temp( 1:end-1000 ) );
handles.persistentPlotHandle = temp(end-999:end);
end
end
end
targetsTime = currentTime;
for n =1:length(handles.Targets)
handles.Targets(n).XY = handles.Targets(n).XY +
dt*handles.Targets(n).v+dt^2*handles.Targets(n).a;
handles.Targets(n).v = handles.Targets(n).v + dt*handles.Targets(n).a;
handles.Targets(n).a = handles.Targets(n).a * 0.95^dt;
targetsManuv = handles.Targets(n).maneuverability;
if (1-exp(-dt/targetsManuv)) > rand(1)
phi = atan( handles.Targets(n).v(2)/handles.Targets(n).v(1) );
phi = phi + 2*pi*(rand(1)-0.5);
handles.Targets(n).a = randn(1)*20*[cos(phi) sin(phi)] - handles.Targets(n).v/targetsManuv/2;
end
end
end % if dt > updateRate  %Is it time to update the display
if isAnalyzeBufferMode
if get(handles.waitForTarget,'value')
if foundTargetInBuffer
if isMTIused
analyzBufferWithMTI(handles,recievedSignal,processedRecivedSignal,energyInFreqRangeCells,P
Wn,freqTh,freqInRangeCells,numSamplesInPRI,rangeCellInd);
else
analyzBuffer(handles,recievedSignal,processedRecivedSignal,energyInRangeCells,PWn,Th,range
CellInd,localMaxEnergy,NPB,PRIA,SR);
```

```
end
set(handles.run,'string','Pause');
AirecraftSimulation('run_Callback',handles.run,[],guidata(handles.run));
end
else
if isMTIused
analyzBufferWithMTI(handles,recievedSignal,processedRecivedSignal,energyInFreqRangeCells,P
Wn,freqTh,freqInRangeCells,numSamplesInPRI,rangeCellInd);
else
analyzBuffer(handles,recievedSignal,processedRecivedSignal,energyInRangeCells,PWn,Th,range
CellInd,localMaxEnergy,NPB,PRIA,SR);
set(handles.run,'string','Pause');
AirecraftSimulation('run_Callback',handles.run,[],guidata(handles.run));
end
set(handles.run,'string','Pause');
AirecraftSimulation('run_Callback',handles.run,[],guidata(handles.run));
end
end
if get(handles.soundOn,'value')
soundSig = tanh( abs(processedRecivedSignal) );
soundSig = soundSig / max(soundSig);
soundSamples = decimate(soundSig,sound2SampleRatio );
player.stop;
player = audioplayer( soundSamples, soundFs );
play(player);
end
% Display analog scope  !!!!!
if get(handles.scopeDisplay,'value')
hold( handles.miniDisplay,'off');
sig = abs(processedRecivedSignal(rangeCellInd));
plot(handles.miniDisplay,log( sig( PWn+1:end,:) ),'color',[239 255 250]/256,'linewidth',1);
set( handles.miniDisplay,'color',[118 219 237]/256);
ylim(handles.miniDisplay,[-22 -8]);
xlim( handles.miniDisplay,[1 numSamplesInPRI-PWn] );
grid( handles.miniDisplay,'on');
if ~updatedDisplay
drawnow;
updatedDisplay = false;
end
end
end %if ~mod(pulseNum,nPRI)  %Only processing every N number of pulses
pulseNum = pulseNum+1;
end % while get(handles.run,'value')
handles.numRadarTurn = numRadarTurn;
handles.pulseNum = pulseNum;
handles.currentTime = currentTime;
guidata(handles.run,handles);
handleRadarControlls(handles,'on');
set(handles.PW,'value',PW/PRI);
function x = fastSemiRandn(N)
% this nested function recives the number of elements to return
% it allways returns an N by 2 matrix of random numbers sammpled
% from the vector randVec
vInd = ceil(rand(2)* (randVecLength-N));
x = [ randVec(vInd(1):vInd(1)+N-1) randVec(vInd(2):vInd(2)+N-1)];
end
end
```

D.Target return:

```
function [t a phi] = targetsReturn(targets, antenaGain,Amp,currTime,w,targetsTime,IF_Freq)
% This function returns the time phase and amplitude of the return signal from existing targets.
dt = targetsTime-currTime;
radarAngle = currTime*w;
```

```
if length(antenaGain)==1 && ~antenaGain %order of comparison is important for performance
% antena is diconected
t = [];
a = [];
phi = [];
return
end
N = length(antenaGain);
M = length(targets);
meterPerSec = 2/3e8;    %I multiply by two as the distance travelled is doubled then targets
distance (pulse has to come back as well)
RCS = [targets.RCS];
v = [targets(:).v];
v = reshape(v,2,M)';
cor = [targets(:).XY];
cor = reshape(cor,2,M)';
acc = [targets(:).a];
acc = reshape(acc,2,M)';
cor = cor + v*dt+acc/2*dt^2;
distSquer = sum( cor.^2, 2 )';
t = sqrt( distSquer ) * meterPerSec;
targetsRelAngle = atan2(cor(:,2) , cor(:,1));
targetsRelAngle = mod(targetsRelAngle-radarAngle+pi,2*pi);
in = round( (targetsRelAngle)/2/pi*N );    %finding the antena gain in this angle
in = min( in,N(ones(M,1)) );
in = max( in,ones(M,1) );
a = RCS .* (antenaGain(in).^2) ./ (distSquer.^2) * Amp ; % here should come the radar formula
(dist2 - is already the distance squared)
phi = mod(IF_Freq*2*pi*t,2*pi)';
t = t';
a = a';
```

E. Plot Target:

```
function h = plotTarget( target,handles)
% This function plots a target in the radar display
x = target.pos(1);
y = target.pos(2);
targetsAngle = atan2(y,x);
v = target.v;
switch v
case 12345
% This target was detected without MTI so it isn't known if it is clutter or plane...
c = 'r';
h = plot(handles.radarDisplay, x,y,'X','MarkerSize',8,'linewidth',2,'tag','foundTarget','color',c);
case 0
% this is clutter (mountain)
h = plot(handles.radarDisplay, x,y,'X','MarkerSize',8,'linewidth',2,'tag','foundTarget','color','r');
otherwise
yDir =  -sign(y);
xDir = -1;
h = quiver(handles.radarDisplay,x,y,xDir*v*cos(targetsAngle),yDir*v*abs(sin(targetsAngle)),20,...
        'color','r','linewidth',2,'marker','<','MarkerSize',8,'tag','foundTarget');
end
```

F. Analyze Buffer:

```
function
analyzBuffer(handles,recievedSignal,processedRecivedSignal,energyInRangeCells,PWn,Th,range
CellInd,localMaxEnergy)
temp = get(handles.samplingRate,'string');
ind = get(handles.samplingRate,'value');
Fs = str2num( temp{ind} )*1e3 ;
```

```
temp = get(handles.bufferSize,'string');
ind = get(handles.bufferSize,'value');
nPRI = str2num( temp{ind} ) ;
temp = get(handles.PRI,'string');
ind = get(handles.PRI,'value');
PRI = str2num( temp{ind} )/1e3 ;
isMatchFilterUsed = get(handles.useMatchFilter,'value');
axes(handles.axes4);
N = length(recievedSignal);
plot([0:N-1]/Fs,(recievedSignal));
if isMatchFilterUsed
hold on;
semilogy([0:N-1]/Fs,abs(processedRecivedSignal),'g');
legend( {'Recived Signal' ; 'Signal after match Filter'} );
else
end
grid on
xlabel('[Sec]');    ylabel('Volt (log scale)');
nRangeCells = length(energyInRangeCells);
set(handles.bufferAnalyze,'value',0);
```

G. Create Target:

```
function targetObj = createTargetObj( pos, RCS, R, v, angle, plotted, clockWise, counterClockWise,
hPlot, foundInTurn )
% function target = createTargetObj( pos, RCS, range, plotted, v, angle, clockWise,minAgnel)
% pos - target cordinate
% RCS - Radar Cross Section of the target
% R - target range (from radar)
% v - target velocity
% angle - target angle (to the radar)
% plotted - was this target plotted (new target wasn't plotted yet)
% max & min Agnle are the limits of angles in which the target was detected
targetObj = struct( 'pos'   , pos, ...
               'RCS'   , RCS, ...
               'R'     , R , ...
               'plotted',plotted , ...
               'v'     , v ,                'angle' , angle, ...
               'clockWise', clockWise, 'counterClockWise',...
               counterClockWise, 'hPlot' ,   hPlot, 'foundInTurn', foundInTurn        );
```

H. Place Cluture:

```
function h=placeClutter(h,XY)
% h=placeClutter(h,XY)
% This function adds a mountain (clutter) at cordinate XY
mountains = h.mountains;
RCS = 1e4;
mountains(end+1).RCS =RCS;
mountains(end).XY = XY;
mountains(end).v = [0 0];
mountains(end).a = [0 0];
mountains(end).maneuverability = 0;
h.mountains = mountains(:);
guidata(h.radarDisplay,h);
```

I. Publications:

1. **Isam Abdelnabi Osman**, Mustafa Osman Ali, Abdelrasoul Jabar Alzebaidi. “Active Cancellation Algorithm for Radar Cross Section Reduction”. Published in International Journal of Computational Engineering Research (IJCER) Volume 3, Issue 7, p.p. 19-24, July 2013.
2. Mustafa Osman Ali, **Isam Abdelnabi Osman**, Rameshwar Rao. “Invisible Watermark Algorithm Embedded in VLSI Chip for Authenticating Digital Image”. Published in International Journal of Engineering Science and Innovative Technology (IJESIT) Volume 2, Issue 4, p.p. 72-79, July 2013.

Isam Abdelnabi Osman, Abdelrasoul Jabar Alzebaidi. “Active Cancellation System for Radar Cross Section Reduction”. Published in International Journal of Education and Research (IJER) Volume 1, Issue 7, p.p. 141- 146, July 2013.