2003-12

Bistatic radar system analysis and software development

Teo, Ching Leong

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

BISTATIC RADAR SYSTEM ANALYSIS AND SOFTWARE DEVELOPMENT

by

Teo, Ching Leong

December 2003

Thesis Advisor:	David C. Jenn
Second Reader:	D. Curtis Schleher

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2003	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Bistatic Radar System Analysis And Software Development			**5. FUNDING NUMBERS**
6. AUTHOR(S) Teo, Ching Leong			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Bistatic radar has some properties that are distinctly different from monostatic radar. Recently bistatic radar has received attention for its potential to detect stealth targets due to enhanced target forward scatter. Furthermore, the feasibility of hitchhiker radar has been demonstrated, which allows passive radar receivers to detect and track targets. This thesis developed a software simulation package in Matlab that provides a convenient tool to examine the bistatic radar design parameters and predict system performance. The software model is suitable for instructional purposes due to its user-friendly graphical user interface. Several bistatic radar applications were used to illustrate the software features, and their results were analyzed and discussed.

14. SUBJECT TERMS Bistatic Radar, Multistatic Radar, Oval of Cassini, Performance Prediction, Computer Simulation, Graphical User Interface			**15. NUMBER OF PAGES** 116
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

BISTATIC RADAR SYSTEM ANALYSIS AND SOFTWARE DEVELOPMENT

Teo, Ching Leong
Singapore
B.Eng., University of Essex, 1997

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING SCIENCE (ELECTRICAL ENGINEERING)

from the

NAVAL POSTGRADUATE SCHOOL
December 2003

Author: Teo, Ching Leong

Approved by: Professor David C. Jenn
Thesis Advisor

Professor Curtis Schleher
Second Reader

Professor John Powers
Chairman,
Department of Electrical & Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Bistatic radar has some properties that are distinctly different from monostatic radar. Recently bistatic radar has received attention for its potential to detect stealth targets due to enhanced target forward scatter. Furthermore, the feasibility of hitchhiker radar has been demonstrated, which allows passive radar receivers to detect and track targets. This thesis developed a software simulation package in MATLAB that provides a convenient tool to examine the bistatic radar design parameters and predict system performance. The software model is suitable for instructional purposes due to its user-friendly graphical user interface. Several bistatic radar applications were used to illustrate the software features, and their results were analyzed and discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to express my most sincere gratitude to Professor David C. Jenn, my thesis supervisor, for his guidance and patience throughout the year while working on this thesis. His encouragement and advice have helped me overcome many difficulties along the way in order to complete this thesis on time.

I would also like to thank Mr. Nicholas J. Willis, the author of *Bistatic Radar* book, for the useful discussions and encouragement given. His book has been an excellent source for understanding the subject.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Bistatic radar uses different antennas separated at a substantial distance, for transmission and reception. This fundamental difference in the bistatic radar has generated several properties that are distinctly different from the monostatic radar. These unique properties have offered bistatic radar certain advantages in several commercial and military applications.

Recently bistatic radar has received attention for its potential to detect stealth targets due to enhanced target forward scatter. Furthermore, the feasibility of hitchhiker radar has been demonstrated, which allows passive radar receivers to detect and track targets, using transmitter of opportunity. Thus, it is likely that bistatic radar will play a much bigger role in near-future military warfare.

Design, development, and testing of radar systems have traditionally been expensive and time-consuming tasks. Radar designers are often faced with many design iterations to optimize the system performance. Thus, using computer simulation and modeling will provide the designer the essential tools to examine design parameters and to evaluate the system performance before the actual implementation in hardware.

There are many software applications available for monostatic radar simulation but none exits in the public domain for bistatic radar. The objective of this thesis is to develop a general software model to analyze the bistatic radar system performance as a function of the system design parameters.

The thesis focused on the design and development of a Graphical User Interface (GUIs) in MATLAB to integrate the bistatic radar parameters and to examine several critical performance tradeoffs. The end product is a program named *Bidarsa*, which provides a convenient instructional tool for engineering communities to understand, analyze, and predict bistatic radar performance.

The thesis presents an overview of bistatic radar definitions and the parameters involved in the range equation. The radar detection contour, the calculation to determine target location, considerations for the bistatic footprint, the doppler relationship for a

moving target, and the electronic countermeasures (ECM) issues are discussed with their supporting mathematical equations. These equations were used to formulate the software model.

The *Bidarsa* program consists of eight major modules developed and integrated to carry out the required performance prediction and system analyses. A set of radar parameters was compiled to demonstrate the software functions. The thesis further illustrates the software functions with two bistatic radar applications. The first application used bistatic radar to monitor shipping channel, and the second application investigated the feasibility of using direct television broadcast satellites and ground-based bistatic receivers to detect targets.

I. INTRODUCTION

Bistatic radar has come a long way from its initial days well before the outbreak of World War II. Many scientists were trying new radar configurations where the receiver was located separately from the transmitter. The main motivation at that time was to gain sufficient radio frequency isolation between the two electronics sets, particularly with the numerous ongoing continuous wave (CW) experiments [1]. This fundamental difference in the bistatic radar over monostatic radar has offered many opportunities in both commercial and military applications. Over the years and through three resurgences, many bistatic radar systems have been proposed, developed, and tested in many countries including the United States, United Kingdom, France, Soviet Union, Germany and Japan.

This unique difference has offered bistatic radar certain advantages for particular tasks [2]:

- The receiver is completely passive, and hence undetectable, by the electronic support measures, and it is safe from attack by anti-radiation missiles or deliberate directional interference and jamming.
- The receiver can be located in a favorable area where transmitters are not allowed, such as flammable-liquids stores, gas terminals, etc.
- No transmit-receive switch or duplexer is required. These devices are lossy, expensive, and heavy (important for air-borne radar applications).
- With certain configurations, less transmitter power is required compared to the monostatic radar.
- Higher pulse repetition frequencies (PRFs) can be used because a bistatic system does not suffer the same range blindness as the equivalent monostatic system.
- If the target angle can be measured at both sites, as well as the bistatic range, data can be checked for self-consistency to remove false alarms.

Taking a closer examination of the military applications, bistatic radar principles have been deployed in systems such as semiactive homing missiles, forward-scatter fences, multistatic radar, and hitchhikers [3]. Much research is being done, and continues, in particular with regard to the last two applications mentioned.

The multistatic radar uses several distributed receivers associated with a single or multiple transmitters. It is suitable for wide area tracking and it also increases the probability of detection. The bistatic hitchhiker uses a transmitter of opportunity, either friendly (cooperative) or hostile (non-cooperative), to detect and locate targets near the transmitter or receiver site. There is no doubt that these bistatic radar principles are playing a bigger role in today's military for the detection of low signature targets and improving the electronic countermeasures and counter-countermeasures.

Design, development, and testing of radar systems have traditionally been expensive and time consuming tasks. Radar designers are often faced with many design iterations to optimize the system performance. Using computer simulation and modeling provides the designer with the essential tools to examine design parameters and evaluate the system performance before the actual implementation in hardware.

There are many software applications available for monostatic radar simulation but none exits in the public domain for bistatic radar. The objective of this thesis was to develop a general software model to analyze the bistatic radar system performance as a function of the system design parameters.

The thesis focused on the design and development of a Graphical User Interface (GUIs) in MATLAB to integrate the bistatic radar parameters, and examine several critical performance tradeoffs. The end product is a program named *Bidarsa*, which provides a convenient instructional tool for engineering communities to understand, analyze, and predict bistatic radar performance.

Chapter II provides the theoretical background required in the formulation of the software model. It presents an overview of bistatic radar definitions and the parameters involved in the range equation. The radar detection contour, the calculation to determine target location, considerations for the bistatic footprint, the doppler relationship for a

moving target, and the electronic countermeasures (ECM) issues are discussed with their supporting mathematical equations.

Chapter III presents the design approach for the *Bidarsa* program. A system block diagram detailing the interfaces and data flow sequences is used to illustrate the functions of the eight major modules and how they are integrated.

Chapter IV uses a set of radar system parameters compiled to demonstrate the *Bidarsa* program functions and discuss its results.

Chapter V uses the *Bidarsa* software to predict the system performance in different practical scenarios suggested in References [2] and [4]. The result is analyzed and compared with the monostatic radar configuration.

Chapter VI summarizes the research and suggests further work that could be performed to enhance the code.

Finally, the appendix lists the software codes written for the *Bidarsa* program.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BISTATIC RADAR THEORY

A. DEFINITIONS

The IEEE Standard 686-1997 [5] has defined *bistatic radar* as a radar using antennas for transmission and reception at sufficiently different locations that the angles or ranges from those locations to the target are significantly different. Figure 2.1 is used to illustrate this definition, showing a transmitter T_x and receiver R_x being situated at two locations with a baseline separation distance L. The bistatic angle, β, is one of the important parameters which characterizes bistatic radar and affects system performance. These relationships will be discussed in the subsequent sections.

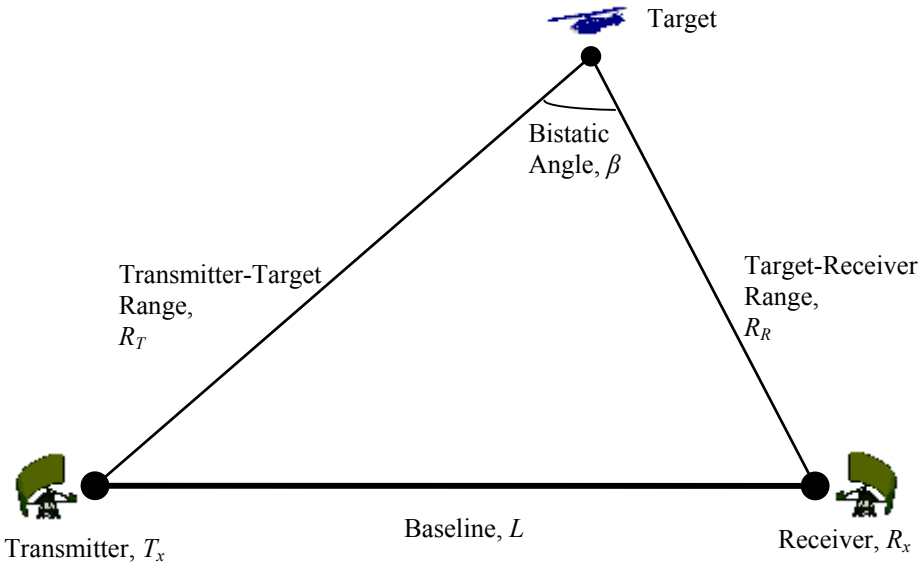

Figure 2.1 Bistatic radar geometry.

There is no strict definition specifying the distance between the transmitter and receiver sites. Several discussions [1, 3, 6] suggest that the baseline, L, could range from tens of meters to hundreds of kilometers, depending on the required functions of the

particular bistatic radar system. The distance R_T is the range from the transmitter-to-target, while R_R is the range from the target-to-receiver.

B. RANGE EQUATION

The bistatic range equation is derived in a similar manner as the monostatic radar range equation. The bistatic maximum range product is given by [3]

$$\left(R_T R_R\right)_{\max} = \left[\frac{P_T G_T G_R \lambda^2 \sigma_B F_T^2 F_R^2}{\left(4\pi\right)^3 k T_s B_n \left(S/N\right)_{\min} L_T L_R}\right]^{\frac{1}{2}} \tag{2.1}$$

or

$$\left(R_T R_R\right)_{\max} = \kappa \tag{2.2}$$

where

R_T = transmitter-to-target range (m),

R_R = target-to-receiver range (m),

P_T = transmitted power output (W),

G_T = transmitting antenna power gain,

G_R = receiving antenna power gain,

λ = transmitting frequency wavelength (m),

σ_B = target bistatic radar cross section (RCS) (m^2),

F_T = pattern propagation factor for the transmitter-to-target path,

F_R = pattern propagation factor for the target-to-receiver path,

k = Boltzmann's constant, 1.3807×10^{-23} J/K,

T_S = receiver noise temperature (°K),

B_n = receiver noise bandwidth (Hz),

$(S/N)_{min}$ = minimum signal-to-noise power ratio required for detection,

L_T = transmitting system losses (≥ 1),

L_R = receiving system losses (≥ 1),

κ = bistatic maximum range product.

In the bistatic radar range equation, it is observed that the maximum range $(R_T R_R)^2$ has replaced the range R^4 in the monostatic radar equation, due to the different locations of the transmitter and receiver.

The pattern-propagation factors for the transmitter-to-target and target-to-receiver, F_T and F_R, also need to be taken care of separately in the bistatic case. This is because the transmitting path and receiving path can be very different in nature as compared to the monostatic implementation.

C. OVALS OF CASSINI

The oval of Cassini is the locus of a points, the product of whose distance from two fixed points is a constant. In the bistatic radar case, it is interesting to analyze the system performance by plotting the ovals of Cassini as a function of signal-to-noise power ratio, *S/N*, with the ranges, R_T and R_R, at the vertices of the triangle.

Equation (2.1) is modified and written as

$$\left(R_T R_R\right)^2_{\max} = \frac{K}{\left(S/N\right)_{\min}} \tag{2.3}$$

where K is the bistatic radar constant and given by

$$K = \frac{P_T G_T G_R \lambda^2 \sigma_B F_T^2 F_R^2}{\left(4\pi\right)^3 k T_s B_n L_T L_R}. \tag{2.4}$$

The constant K is related to the bistatic maximum range product, κ, by

$$K = \kappa^2 \left(S/N\right)_{\min}. \tag{2.5}$$

Figure 2.2 shows R_T and R_R being converted into the polar coordinate system (r,θ). From geometry

$$R_T^{\,2} = \left(r^2 + L^2/4\right) + rL\cos\theta \tag{2.6}$$

and

$$R_R^{\,2} = \left(r^2 + L^2/4\right) - rL\cos\theta\,. \tag{2.7}$$

Combining Equations (2.6) and (2.7) yields

$$\left(R_T R_R\right)^2 = \left(r^2 + L^2/4\right)^2 - r^2L^2\cos^2\theta\,. \tag{2.8}$$

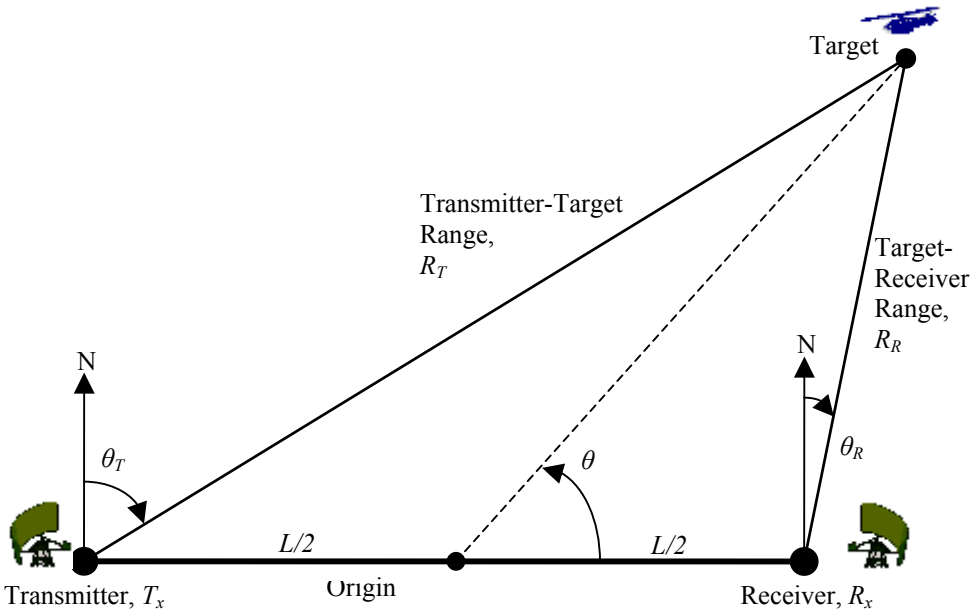

Figure 2.2 Bistatic radar geometry converted to the polar coordinate system. (After Ref. [3].)

The constant S/N ratio can subsequently be plotted as ovals of Cassini using the equation

$$\left(S/N\right) = \frac{K}{\left(r^2 + L^2/4\right)^2 - r^2L^2\cos^2\theta}\,. \tag{2.9}$$

Reference [3] has provided an example of the oval of Cassini plot, as shown in Figure 2.3, with K arbitrarily set to $30L^4$ and constant S/N ratios from 10 dB to 30 dB.

For analytical purposes, σ_B, F_T and F_R are assumed to be invariant with r and θ so as to generate a constant-SNR oval-of-Cassini plot. A series of plots shows that bistatic radar operation can be divided into three distinct regions:

1. $L > 2\sqrt{\kappa}$. Two separate ellipses result enclosing the transmitter and receiver. It is considered receiver centered when $R_T >> R_R$; transmitter centered when $R_R >> R_T$.

2. $L < 2\sqrt{\kappa}$. A single continuous ellipse results.

3. $L = 2\sqrt{\kappa}$. A lemniscate with a cusp at origin results.

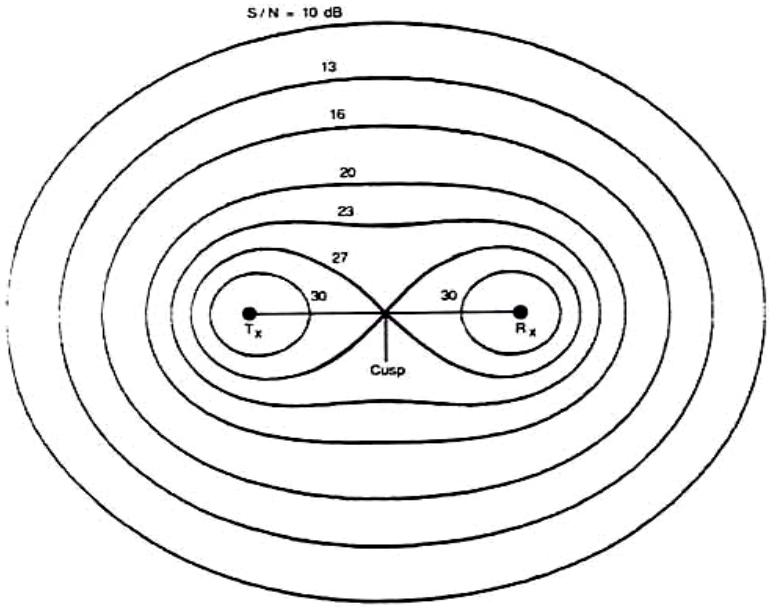

Figure 2.3 Ovals of Cassini, with constant SNR plots for K=30L^4. (From Ref. [3].)

D. TARGET LOCATION

A bistatic receiver usually measures the angles of arrival of the target echo and estimates the range. The receiver-to-target range, R_R, cannot be measured directly, but can be calculated by solving the bistatic triangle. When the transmitter location is made known to the receiver, the baseline range, L, is known. The total signal propagation time

from transmitter to target to receiver is also required to be established. This is needed for the receiver to convert the time measurement into the range sum estimation, $R_T + R_R$.

From Figures 2.2 and 2.3, it can be seen that the isorange contour has to be an ellipse, rather than the tangent approximation. It is defined by

$$R_T + R_R = 2a \tag{2.10}$$

where $2a$ is the major axis of the ellipse, the eccentricity of the ellipse, e, is

$$e = L / 2a \tag{2.11}$$

$$e = L / \left(R_T + R_R \right). \tag{2.12}$$

Using the law of cosines, R_R and R_T can be calculated by measuring the L, ($R_T + R_R$), and either the angle θ_T or θ_R.

For the case when θ_T is measured

$$R_R^2 = R_T^2 + L^2 - 2R_T L \cos\left(90° - \theta_T\right) \tag{2.13}$$

$$R_T = \frac{\left(R_T + R_R\right)^2 - L^2}{2\left(R_T + R_R - L\sin\theta_T\right)} \tag{2.14}$$

$$R_R = \left(R_T^2 + L^2 - 2R_T L \sin\theta_T\right)^{\frac{1}{2}}. \tag{2.15}$$

Combining Equations (2.12) and (2.14) yields

$$R_T = \frac{L\left(1 - e^2\right)}{2e\left(1 - e\sin\theta_T\right)} \tag{2.16}$$

and combining Equations (2.12) and (2.15) yields

$$R_R = \frac{L\left(e^2 + 1 - 2R\sin\theta_T\right)}{2e\left(1 - e\sin\theta_T\right)}. \tag{2.17}$$

Similarly, for the case when θ_R is measured

$$R_T^2 = R_R^2 + L^2 - 2R_R L \cos\left(90° + \theta_R\right) \tag{2.18}$$

$$R_R = \frac{\left(R_T + R_R\right)^2 - L^2}{2\left(R_T + R_R + L\sin\theta_R\right)} \tag{2.19}$$

$$R_T = \left(R_R^2 + L^2 + 2R_R L\sin\theta_R\right)^{\frac{1}{2}}. \tag{2.20}$$

Combining Equations (2.12) and (2.19) yields

$$R_R = \frac{L\left(1-e^2\right)}{2e\left(1+e\sin\theta_R\right)}. \tag{2.21}$$

Finally, combining Equations (2.12) and (2.20) yields

$$R_T = \frac{L\left(e^2+1+2R\sin\theta_R\right)}{2e\left(1+e\sin\theta_R\right)}. \tag{2.22}$$

Using the law of sines, it is also possible to establish the parameter ratios to calculate the bistatic angle, β, with the measured R_R and R_T and the calculated θ_T and θ_R. Their relationship is given by

$$\frac{R_R}{\sin\left(90°-\theta_T\right)} = \frac{R_T}{\sin\left(90°+\theta_R\right)} = \frac{L}{\sin\beta}. \tag{2.23}$$

Thus

$$\frac{R_R}{R_T} = \frac{\cos\theta_T}{\cos\theta_R} \tag{2.24}$$

$$\frac{R_R}{L} = \frac{\cos\theta_T}{\sin\beta} \tag{2.25}$$

$$\beta = \sin^{-1}\left(\frac{L\cos\theta_T}{R_R}\right). \tag{2.26}$$

Also,

$$\frac{R_T}{L} = \frac{\cos\theta_R}{\sin\beta} \tag{2.27}$$

$$\beta = \sin^{-1}\left(\frac{L\cos\theta_R}{R_T}\right). \tag{2.28}$$

E. COVERAGE AREA

There are two considerations for establishing the coverage area of the bistatic radar: (1) the detection-constraint which is determined by the maximum range oval of Cassini, and (2) the line-of-sight (LOS) constraint which is determined by the geometry between the transmitter, target, and receiver.

1. Detection-Constrained Coverage

The coverage is determined by the area within a maximum-range oval of Cassini, which is established by the bistatic maximum range product, κ, in a bistatic radar system.

Depending on region of operation, i.e., the relationship between L and $2\sqrt{\kappa}$, different expressions for the area, A_B, apply. The detailed derivation of each expression is provided in Appendix D of [3].

a. $L < 2\sqrt{\kappa}$. A single continuous ellipse.

$$A_{B1} = \pi\kappa\left[1-\left(\frac{1}{2}\right)^2\left(\frac{L^4}{16\kappa^2}\right)\left(\frac{1}{1}\right)-\left(\frac{1\bullet 3}{2\bullet 4}\right)^2\left(\frac{L^4}{16\kappa^2}\right)^2\left(\frac{1}{3}\right)-\cdots\right] \tag{2.29}$$

$$A_{B1} \approx \pi\kappa\left[1-\left(\frac{1}{64}\right)\left(\frac{L^4}{\kappa^2}\right)-\left(\frac{3}{16384}\right)\left(\frac{L^8}{\kappa^4}\right)\right]. \tag{2.30}$$

b. $L > 2\sqrt{\kappa}$. Two separate ellipses enclosing the transmitter and receiver.

$$A_{B2} = \frac{2\pi\kappa^2}{L^2}\left[\begin{array}{l} 1+\left(\frac{1}{2^2\bullet 2!}\right)^2\left(\frac{16\kappa^2}{L^4}\right)+\left(\frac{3^2}{2^4\bullet 3!\bullet 2!}\right)^2\left(\frac{16\kappa^2}{L^4}\right)^2 \\ +\left(\frac{3^2\bullet 5^2}{2^6\bullet 4!\bullet 3!}\right)^2\left(\frac{16\kappa^2}{L^4}\right)^3+\cdots \end{array}\right] \tag{2.31}$$

$$A_{B2} \approx \frac{2\pi\kappa^2}{L^2}\left[1+\frac{2\kappa^2}{L^4}+\frac{12\kappa^4}{L^8}+\frac{100\kappa^6}{L^{12}}\right]. \tag{2.32}$$

c. $L = 2\sqrt{\kappa}$. A lemniscate with a cusp at the origin

$$A_{B3} = 2\kappa. \tag{2.33}$$

Comparing a bistatic radar with maximum range product, κ, to a monostatic radar with an equivalent monostatic range, $\sqrt{\kappa}$, it is found that the area of coverage for a bistatic radar is smaller than that of the monostatic radar [3].

2. Line-of-Sight-Constrained Coverage

The bistatic line-of-sight (LOS) coverage is affected by multipath, diffraction, refraction, and shadowing, including the earth curvature. The coverage area, A_C, is established by the common area covered by both the transmitter and receiver coverage circles, as shown in Figure 2.4. The 4/3 earth model gives radii of coverage circles as follows

$$r_T = 130\left(\sqrt{h_t}+\sqrt{h_T}\right) \tag{2.34}$$

$$r_R = 130\left(\sqrt{h_t}+\sqrt{h_R}\right) \tag{2.35}$$

where

h_t = target altitude (km),

h_T = transmitter antenna altitude (km),

h_R = receiver antenna altitude (km).

For $L + r_T > r_R > L - r_T$ or $L + r_R > r_T > L - r_R$, the intersection of the two coverage circles determines the common coverage area

$$A_C = \frac{1}{2}\left[r_R^2\left(\phi_R - \sin\phi_R\right) + r_T^2\left(\phi_T - \sin\phi_T\right)\right] \tag{2.36}$$

where

$$\phi_T = 2\cos^{-1}\left(\frac{r_T^2 - r_R^2 + L^2}{2r_T L}\right) \tag{2.37}$$

$$\phi_R = 2\cos^{-1}\left(\frac{r_R^2 - r_T^2 + L^2}{2r_R L}\right) \tag{2.38}$$

otherwise,

$$\text{if } r_T + r_R \le L \text{, then } A_C = 0; \tag{2.39}$$

$$\text{if } r_T \ge L + r_R \text{, then } A_C = \pi r_R^2; \tag{2.40}$$

$$\text{if } r_R \ge L + r_T \text{, then } A_C = \pi r_T^2. \tag{2.41}$$

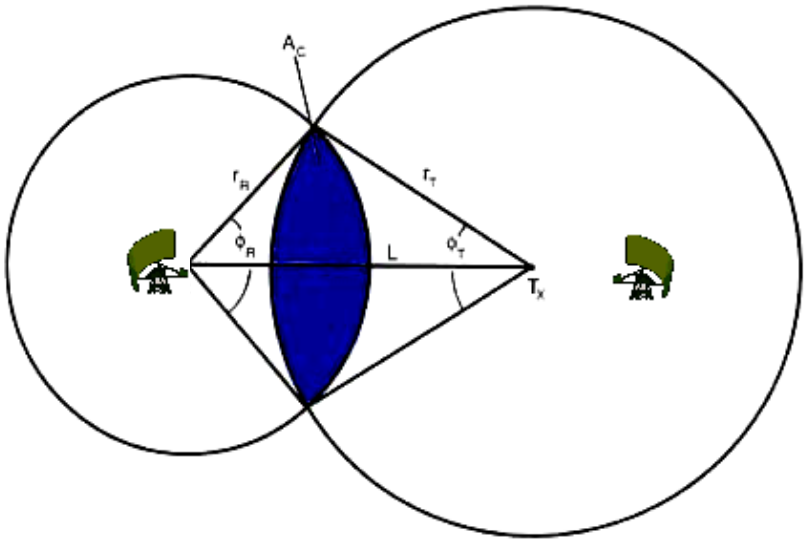

Figure 2.4 Coverage area geometry. (From Ref. [3].)

F. CUTTER CELL AREA

The intersection of the range cell and the mainbeam footprint is used to determine the bistatic mainlobe clutter cell area, A_c. It is useful to determine if a range cell is beamwidth limited or pulsewidth limited.

Figure 2.5 illustrates the beamwidth limited case with clutter cell area given by

$$\left(A_c\right)_b \approx \frac{\left(R_R \Delta\theta_R\right)\left(R_T \Delta\theta_T\right)}{\sin\beta}. \tag{2.42}$$

The terms $R_T\Delta\theta_T$ and $R_R\Delta\theta_R$ are the cross-range dimension of the transmitting and receiving beam at the clutter cell, respectively. The angles $\Delta\theta_T$ and $\Delta\theta_R$ are the 3-dB beamwidths of the transmitting and receiving antennas, respectively.

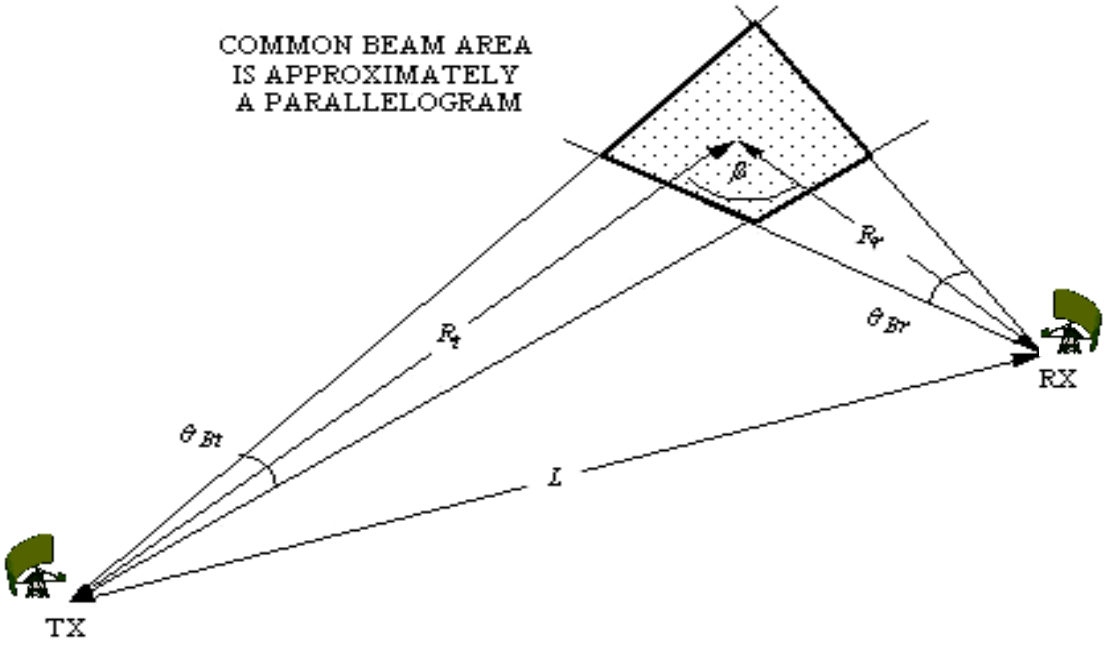

Figure 2.5 Beamwidth limited clutter cell area. (From Ref. [7].)

Figure 2.6 illustrates the pulsewidth limited case with clutter cell area given by

$$\left(A_c\right)_b \approx \Delta R\left[\frac{\left(R_R \Delta\theta_R\right)}{\cos\left(\beta/2\right)}\right] \approx \frac{c\tau R_R \Delta\theta_R}{2\cos^2\left(\beta/2\right)}. \tag{2.43}$$

The term τ is the compressed pulsewidth, whereas ΔR is the approximation to the bistatic range cell given by

$$\Delta R \approx \frac{c\tau}{2\cos(\beta/2)}. \tag{2.44}$$

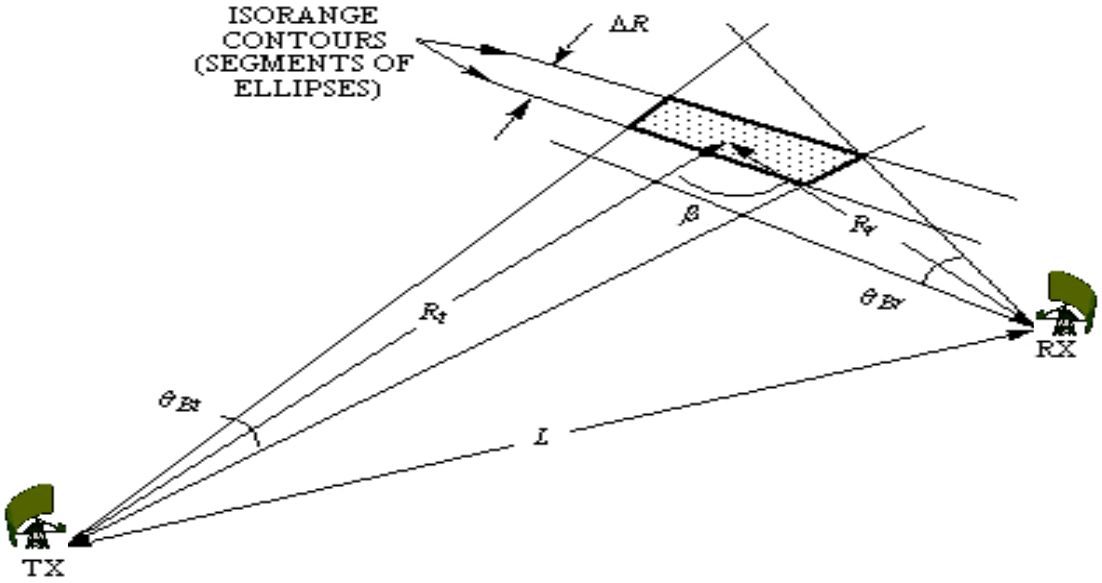

Figure 2.5 Pulsewidth limited clutter cell area. (From Ref. [7].)

Appendix B of Reference [3] has provided an estimation for the maximum error of a range cell, $\varepsilon_{\max}$. The maximum error occurs on the perpendicular bisector of the baseline, where β is a maximum. It is given by

$$\varepsilon_{\max} = \frac{a(a'-a)}{b(b'-b)} - 1 \tag{2.45}$$

where

a = semimajor axis of inner ellipse = $(R_T + R_R)/2$,

a' = semimajor axis of outer ellipse = $(a + c\tau/2)$,

b = semiminor axis of inner ellipse = $(a^2 - L^2/4)^{\frac{1}{2}}$,

b' = semiminor axis of outer ellipse = $\left[(a')^2 - L^2/4\right]^{\frac{1}{2}}$.

G. MAXIMUM UNAMBIGUOUS RANGE AND PRF

The maximum unambiguous range for a bistatic radar is given by

$$\left(R_T + R_R\right)_u = \frac{c}{PRF} \tag{2.46}$$

and the maximum unambiguous *PRF* is given by

$$\left(PRF_B\right)_u = \frac{c}{\left(R_T + R_R\right)_{\max}} \tag{2.47}$$

$$\left(PRF_B\right)_u = \frac{c}{\sqrt{L^2 + 2\kappa\left(1+\cos\beta\right)}}. \tag{2.48}$$

H. DOPPLER RELATIONSHIP

The doppler shift, f_B, is the time rate of change of the total path length of the scattered signal, normalized by the wavelength. For bistatic radar, the doppler shift is given by

$$f_B = \frac{1}{\lambda}\left[\frac{d}{dt}\left(R_T + R_R\right)\right] \tag{2.49}$$

$$f_B = \frac{1}{\lambda}\left[\frac{dR_T}{dt} + \frac{dR_R}{dt}\right]. \tag{2.50}$$

Figure 2.6 shows a bistatic doppler geometry. The target's velocity vector has magnitude V and aspect angle δ. They are being projected to the transmitter and receiver as V_T, V_R and δ_T, δ_R respectively. A stationary transmitter and receiver will have the projection of the target velocity vector onto the transmitter-to-target LOS and receiver-to-target LOS as follows

$$\frac{dR_T}{dt} = V\cos\left(\delta - \beta/2\right) \tag{2.51}$$

$$\frac{dR_R}{dt} = V\cos\left(\delta + \beta/2\right). \tag{2.52}$$

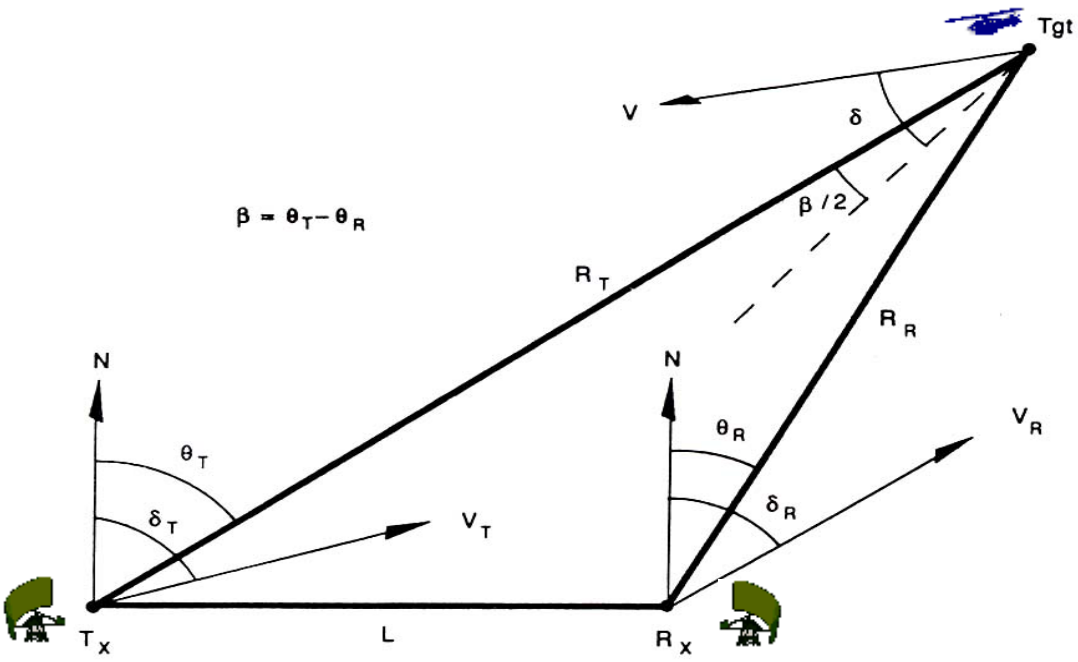

Figure 2.6 Bistatic doppler geometry. (From Ref. [3].)

Combining Equations (2.50), (2.51) and (2.52) yields

$$f_B = (V/\lambda)\left[\cos(\delta - \beta/2) + \cos(\delta + \beta/2)\right] \tag{2.53}$$

$$f_B = (2V/\lambda)\cos\delta\cos(\beta/2). \tag{2.54}$$

I. ELECTRONIC COUNTERMEASURES

Bistatic radar can also be subjected to intentional interference initiated by hostile transmissions from other sources. Thus it is essential to examine how the performance will be affected under such an environment. When a noise jammer is employed, the radar receiver will be subjected to the jamming power spectrum density, J_0, given by

$$J_0 = \frac{P_J G_J G_R \lambda^2 F_J^2}{(4\pi)^2 B_J R_J^2 L_J} \tag{2.55}$$

where

P_J = jammer transmitted power (W),

G_J = jammer transmit antenna power gain,

G_R = radar receiving antenna power gain,

F_J = jammer pattern propagation factor = $F_J' f_J f_{RJ}$,

F_J' = propagation factor,

f_J = jammer antenna patter factor, referenced to the receiver site,

f_{RJ} = radar receiving antenna patter factor, referenced to jammer site,

B_J = jammer bandwidth (Hz),

R_J = jammer-to-receiver range (m),

L_J = jammer system losses (≥ 1).

Assuming the case of a single jammer, substituting Equation (2.55) into (2.1), the bistatic burn-through equation is given by

$$\left(R_T R_R\right)_{J1} = \left[\frac{P_T G_T}{P_J G_J} \times \frac{F_T^2 F_R^2}{F_J^2} \times \frac{L_J}{L_T L_R} \times \frac{B_J}{B_n} \times \frac{\sigma_B R_J^2}{4\pi \left(S/N\right)_{\min}}\right]^{\frac{1}{2}}. \quad (2.56)$$

J. SUMMARY

The bistatic radar equation is derived in a manner similar to the monostatic radar range equation, with the bistatic range $(R_T R_R)^2$ replacing the range R^4 in the monostatic radar case. A constant SNR plot has the shape of an oval of Cassini. By resolving the bistatic triangle, it is possible to calculate the target location. The radar coverage is defined by the region where the target is detectable by the receiver and within the LOS of both transmitter and receiver. The bistatic footprint can be either beamwidth or pulsewidth limited. Both doppler relationships and electronic countermeasures considerations were also discussed. Chapter III uses the theory discussed in this chapter to establish the approach for the bistatic radar simulation, and the integration of the various software modules.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SOFTWARE DESIGN APPROACH

A. OVERVIEW

Bidarsa is a bistatic radar simulation developed using the graphical user interfaces (GUIs) available in MATLAB. The program is designed to provide easy user interfaces for radar designers, engineers, and students to examine the various design parameters for a bistatic radar and analyze its performance.

The bistatic radar model block diagram is shown in Figure 3.1. Eight main modules were developed to carry out different bistatic radar system analyses as discussed in Chapter II. Figure 3.1 also indicates the data flow sequences. It illustrates how certain modules will need to be executed prior to the others, as their results are used in subsequent analyses by the other modules.

The *Bidarsa.m* module allows a user to select one of the three group functions: (1) specify the system parameters, (2) carry out a basic system performance analysis, or (3) to carry out an extended system performance analysis.

The *Defpara.m* module provides a convenient way for a user to specify the bistatic radar's transmitter and receiver design parameters. It also allows target information and the environmental information to be specified. The parameters can be updated at any point in time to allow a new scenario to be examined, and its effect on the overall system performance evaluated.

The *RngCal.m* module computes the bistatic maximum range product, bistatic radar constant, and the maximum range-unambiguous pulse repetition frequency (PRF). This module allows the user to carry out system analysis on four different design parameters: (1) transmitter power, (2) system frequency, (3) target RCS, and (4) signal-to-noise ratio. The user is also able to plot the detection contour and examine the oval of Cassini with four different constant SNR plots.

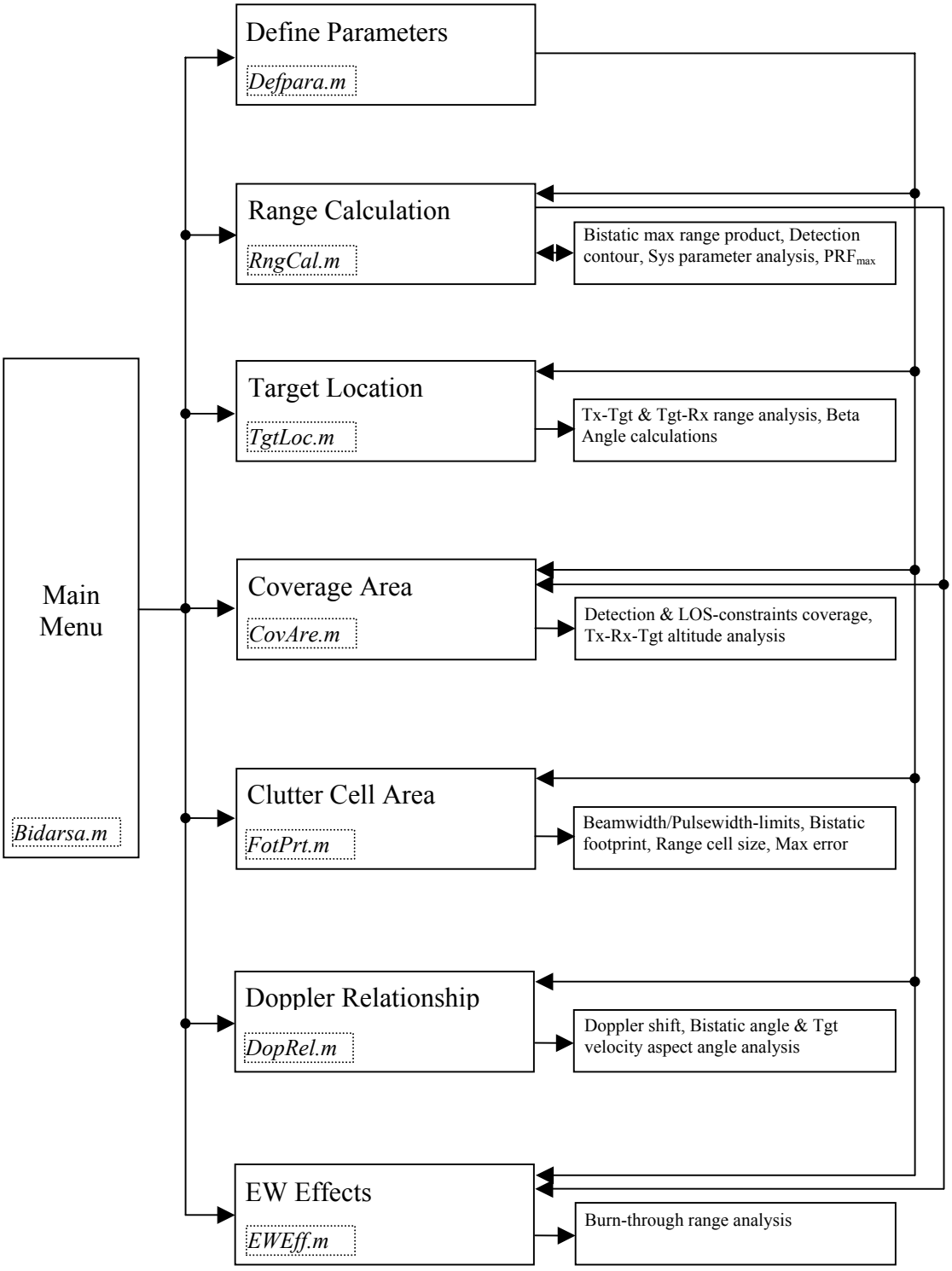

Figure 3.1 Bistatic radar system analysis model block diagram.

The *TgtLoc.m* module uses the range-sum value and angle θ to compute the location of the target. It provides the transmitter-to-target and target-to-receiver ranges and calculates the bistatic angle.

The *CovAre.m* module evaluates the coverage areas for both detection-constrained and LOS-constrained cases. The module also provides an analysis on the transmitter-target-receiver height relationship requirement, and the effects of different baseline ranges.

The *FotPrt.m* module uses the transmitter-target-receiver ranges, antenna beamwidth and transmitter pulsewidth, to evaluate the bistatic footprint and clutter area. The module examines the results and determines if the detection is beamwidth or pulsewidth limited. The range cell size and maximum range errors are also computed in this module.

The *DopRel.m* module examines the doppler shift effects of a moving target. It also provides an analysis of the target-velocity aspect angle versus bistatic angle, and how they affect the doppler frequency shift.

The *EWEff.m* module provides a means to evaluate the bistatic radar performance under different jammer environments. The module provides an analysis on the burn-through range for different target RCS and jammer power scenarios.

A Help menu is provided in all modules to assist the user to navigate and understand how to use each module.

Print and Close functions are also designed into all modules to allow the user to make a hard copy and exit the selected menu whenever desired.

The detailed MATLAB programs for the eight modules are listed in the Appendix of this thesis.

B. BIDARSA MAIN MENU

To run the *Bidarsa* GUI, the user has to start MATLAB and set the path of directory to where the *Bidarsa* code is stored. Typing *Bidarsa* at the command window will execute the GUI as shown in Figure 3.2. Among the three groupings, the user should define the system parameters first before running the other analysis functions. An explanation of the *Bidarsa* GUI may be obtained by clicking on the *Help* button.

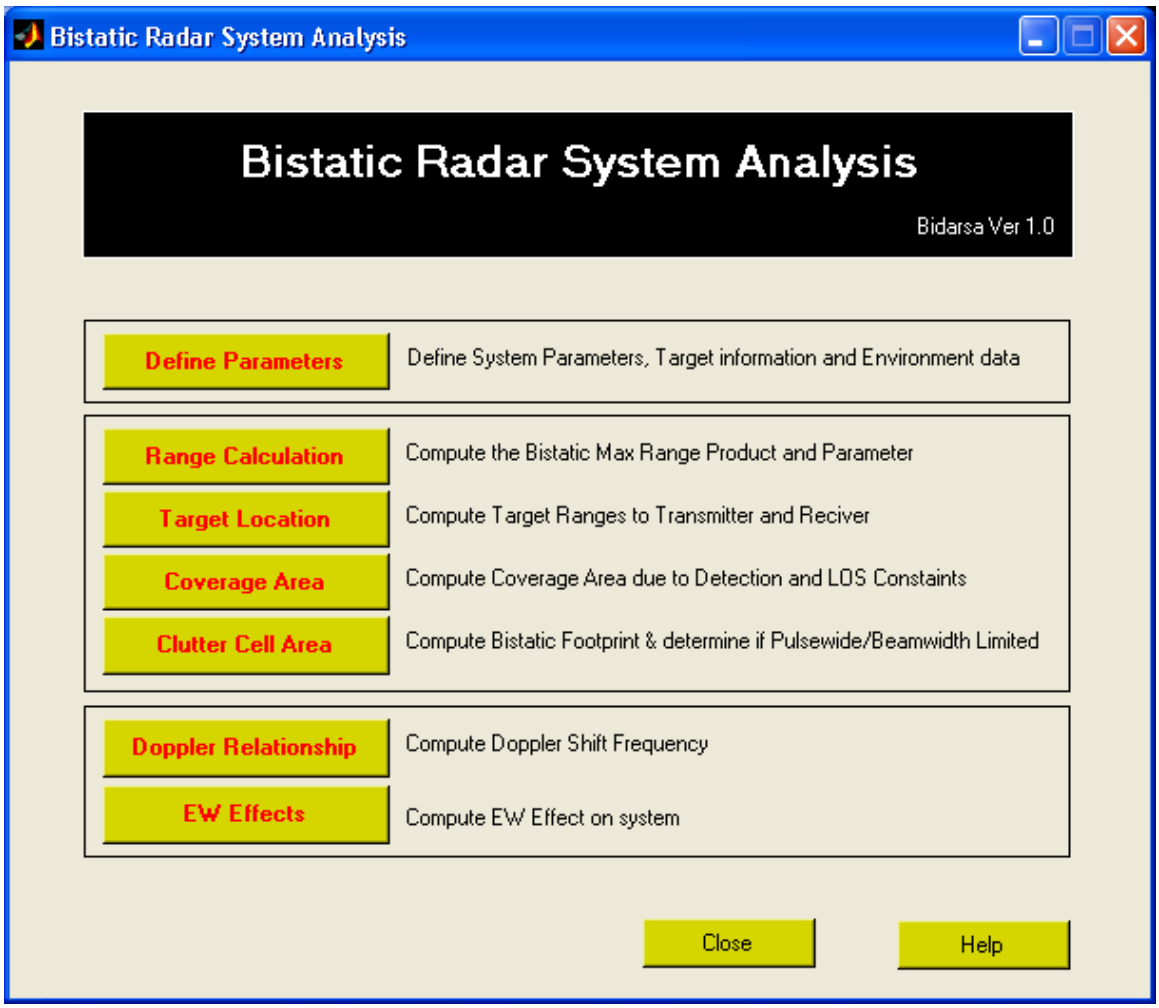

Figure 3.2 *Bidarsa* main menu GUI.

C. DESIGN PARAMETERS MENU

This module is designed to collect the system parameters, target information and environmental data. These inputs are generally required by all the other modules. It functions as a central database for the program, and converts each parameter to a suitable unit for subsequent computations. A pop up window is designed to inform the user that new parameters are being loaded for any follow-on analysis. Figure 3.3 shows the GUI layout designed for the module. A typical value for each parameter was chosen as the default value for demonstration and initial calculation purposes.

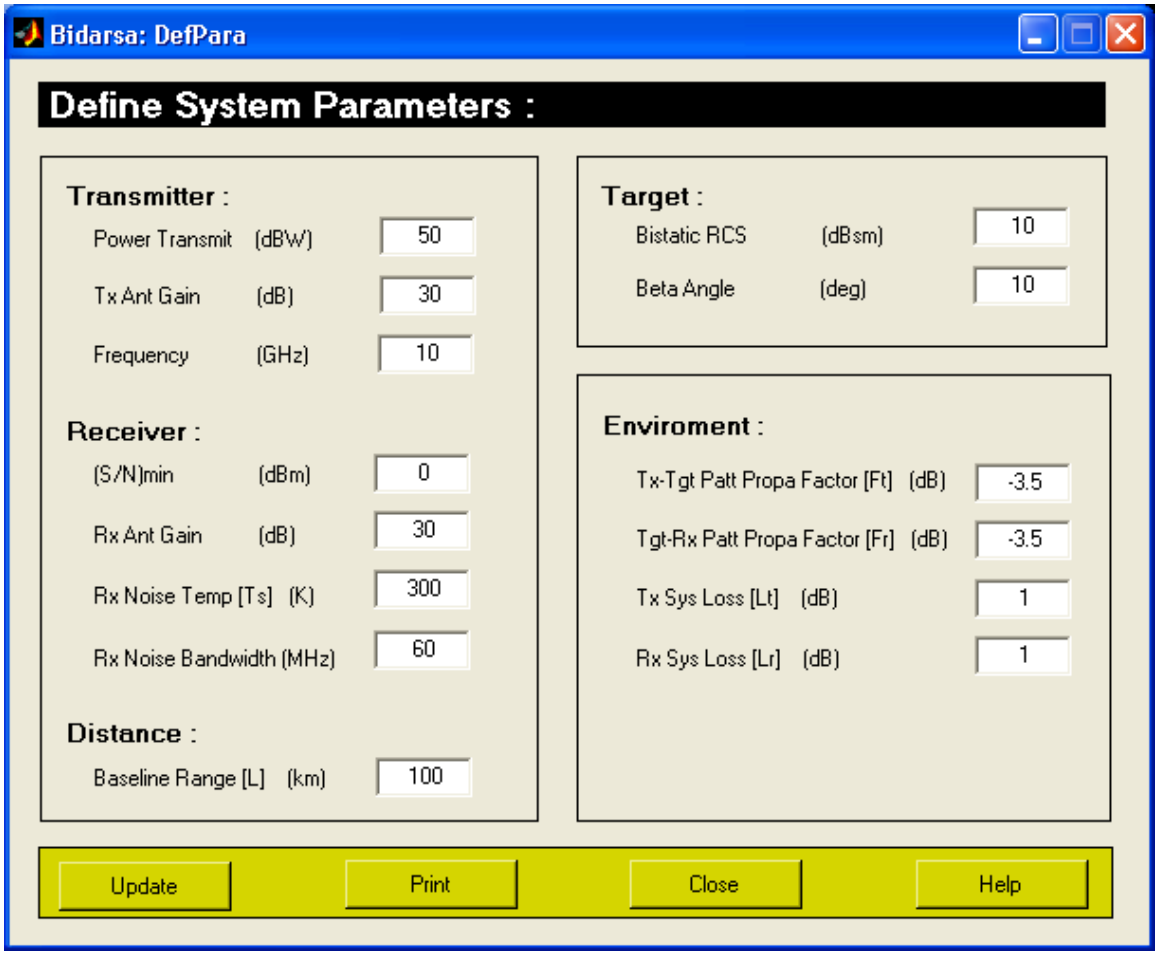

Figure 3.3 Define parameter GUI.

D. RANGE CALCULATION MENU

This module is designed to carry out the most fundamental analysis for a bistatic radar system. Several other modules use its result to carry out further analysis. Figure 3.4 shows the GUI layout designed for the module. With the input data from the *Defpara.m* module, it computes the following:

1. Bistatic maximum range product, $\left(R_T R_R\right)_{\max}$, as detailed in Equation (2.1).
2. Bistatic radar constant, *K*, as detailed in Equation (2.4).
3. Maximum range-unambiguous PRF, as detailed in Equation (2.48).
4. System Analysis for
   a. Transmitting power selection
   b. System frequency selection
   c. Target RCS variation
   d. Receiver SNR variation
5. Detection contour plot for ovals of Cassini, as detailed in Equation (2.9).

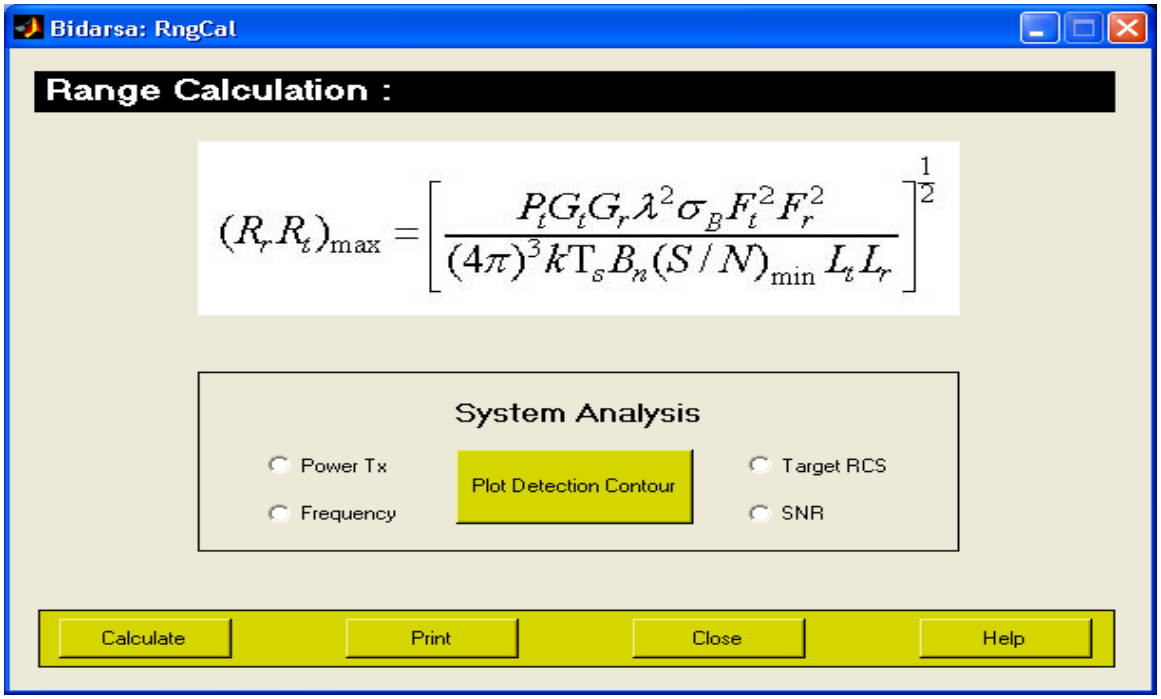

Figure 3.4 Range calculation GUI.

E. TARGET LOCATION MENU

This module is designed to estimate the location of the target using the range-sum value and measured angle θ. Figure 3.5 shows the GUI layout designed for the module. Only one of the angles, θ_T or θ_R, is required for the reference, as discussed in Chapter II Section D. It computes:

1. Transmitter-to-target range, as detailed in Equations (2.16) and (2.22).
2. Target-to-receiver range, as detailed in Equations (2.17) and (2.21).
3. Bistatic beta angle, as detailed in Equations (2.26) and (2.28).

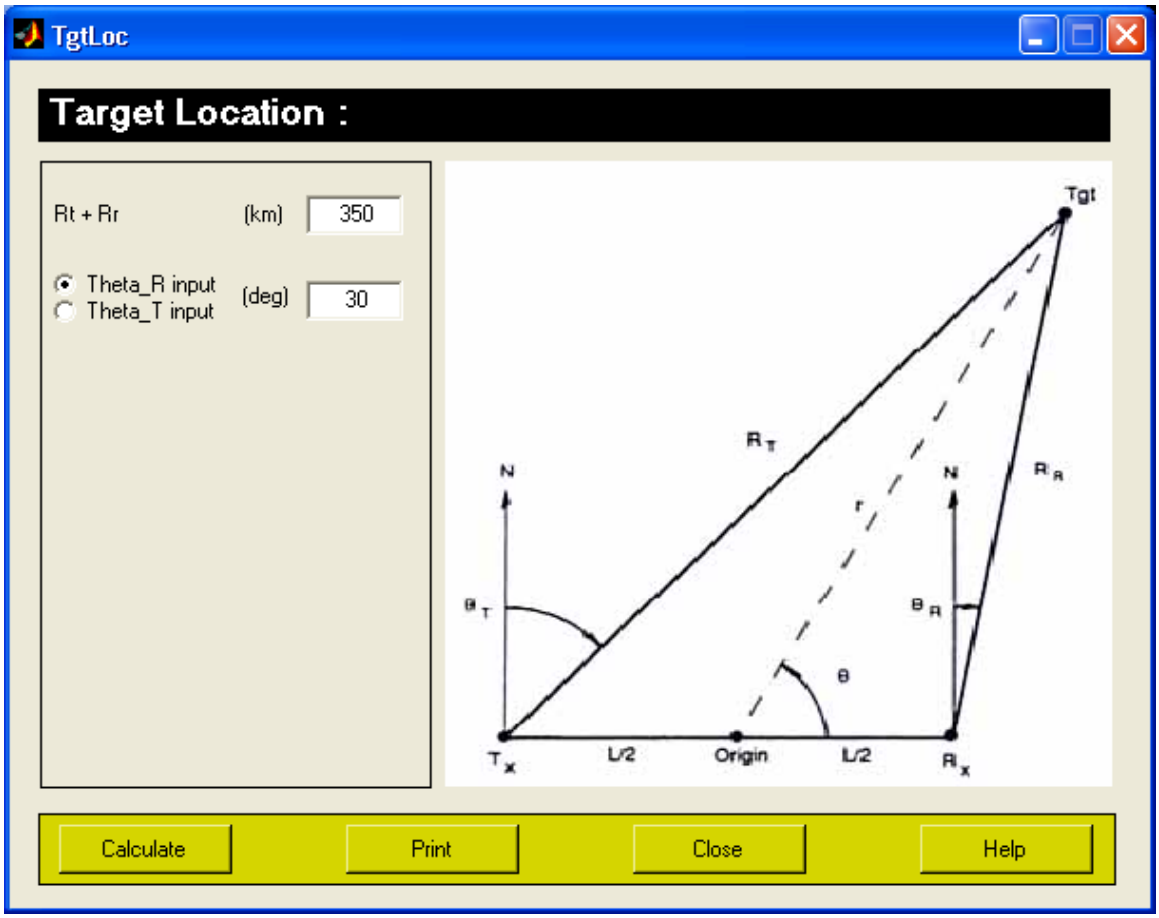

Figure 3.5 Target location GUI.

F. COVERAGE AREA MENU

This module is designed to evaluate the region or area on the bistatic plane where the target is detectable by the receiver and within the LOS of both transmitter and receiver. The user can specify a different altitude for the target, transmitter antenna, and receiver antenna for this analysis. Figure 3.6 shows the GUI layout designed for the module. It computes:

1. The region of operation, based on the relationship of L and $2\sqrt{\kappa}$, as discussed in Chapter II Section E.
2. Coverage area for the detection-constrained case, as detailed in Equations (2.30), (2.32), and (2.33).
3. Coverage area for the LOS-constrained case, as detailed in Equations (2.36), (2.39), (2.40), and (2.41).
4. System analysis for
    a. Transmitter antenna altitude requirements.
    b. Receiver antenna altitude requirements.
    c. Target altitude requirements.
    d. Baseline distance variations.

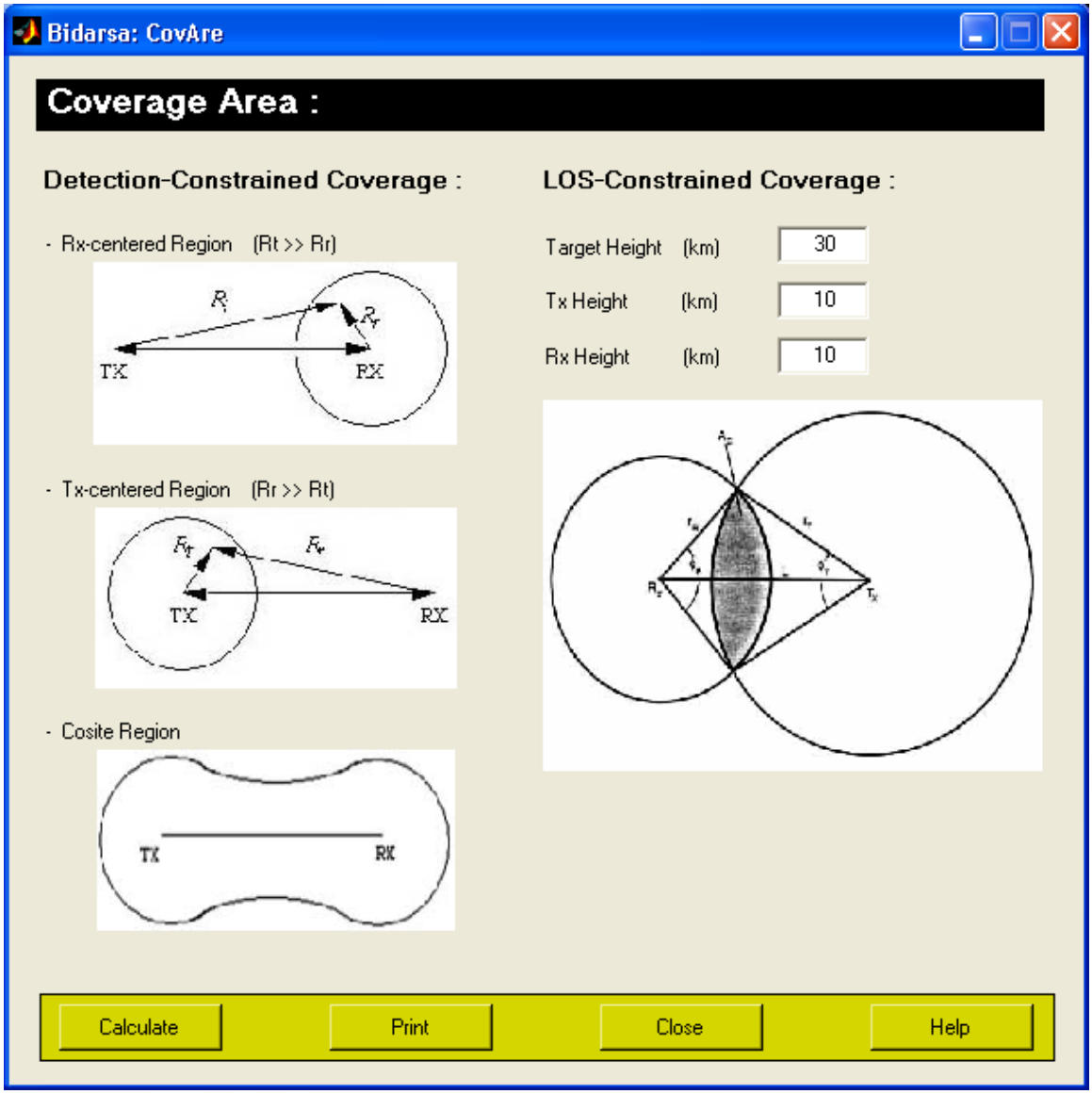

Figure 3.6 Coverage area GUI.

G. BISTATIC FOOTPRINT AND CLUTTER AREA MENU

This module is designed to evaluate the bistatic footprint and clutter area of the bistatic radar system. The user can specify the transmitter-target-receiver ranges, antenna beamwidth and transmitter pulse information for different analysis. Figure 3.7 shows the GUI layout designed for the module. It computes:

1. Whether the system is a beamwidth or pulsewidth limited case.
2. The bistatic footprint area, as detailed in Equations (2.42) and (2.43).
3. The range cell size, as detailed in Equation (2.44).
4. Maximum error in the range cell, as detailed in Equation (2.45).

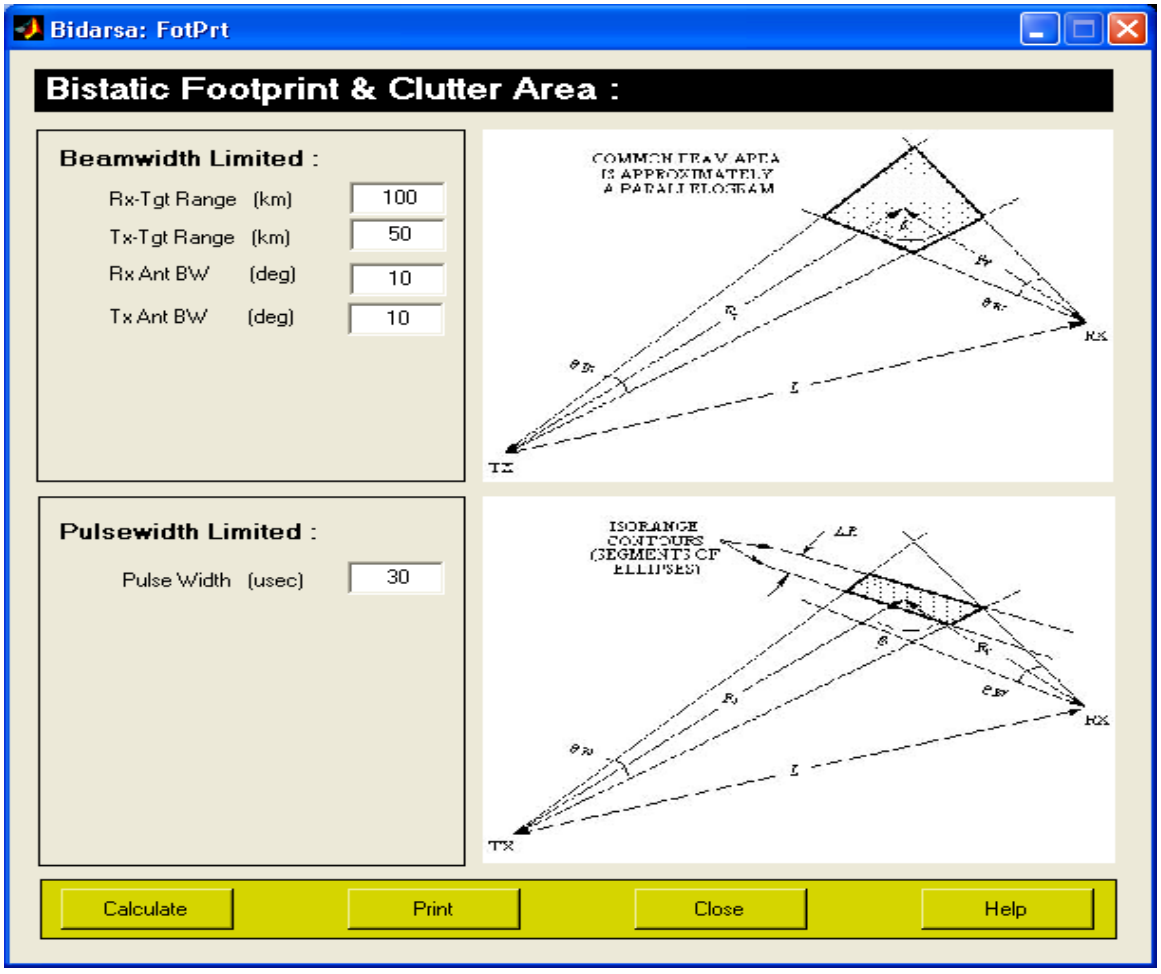

Figure 3.7 Clutter area GUI.

H. DOPPLER RELATIONSHIP MENU

This module is designed to evaluate the doppler shift effects of a moving target. The user can provide different target velocities and aspect angles for the analysis. Figure 3.8 shows the GUI layout designed for the module. It computes:

1. Bistatic doppler shift, as detailed in Equation (2.54).
2. System analysis for
    a. Target velocity aspect angle variation.
    b. Bistatic angle variation.
    c. Bistatic doppler shift variation.

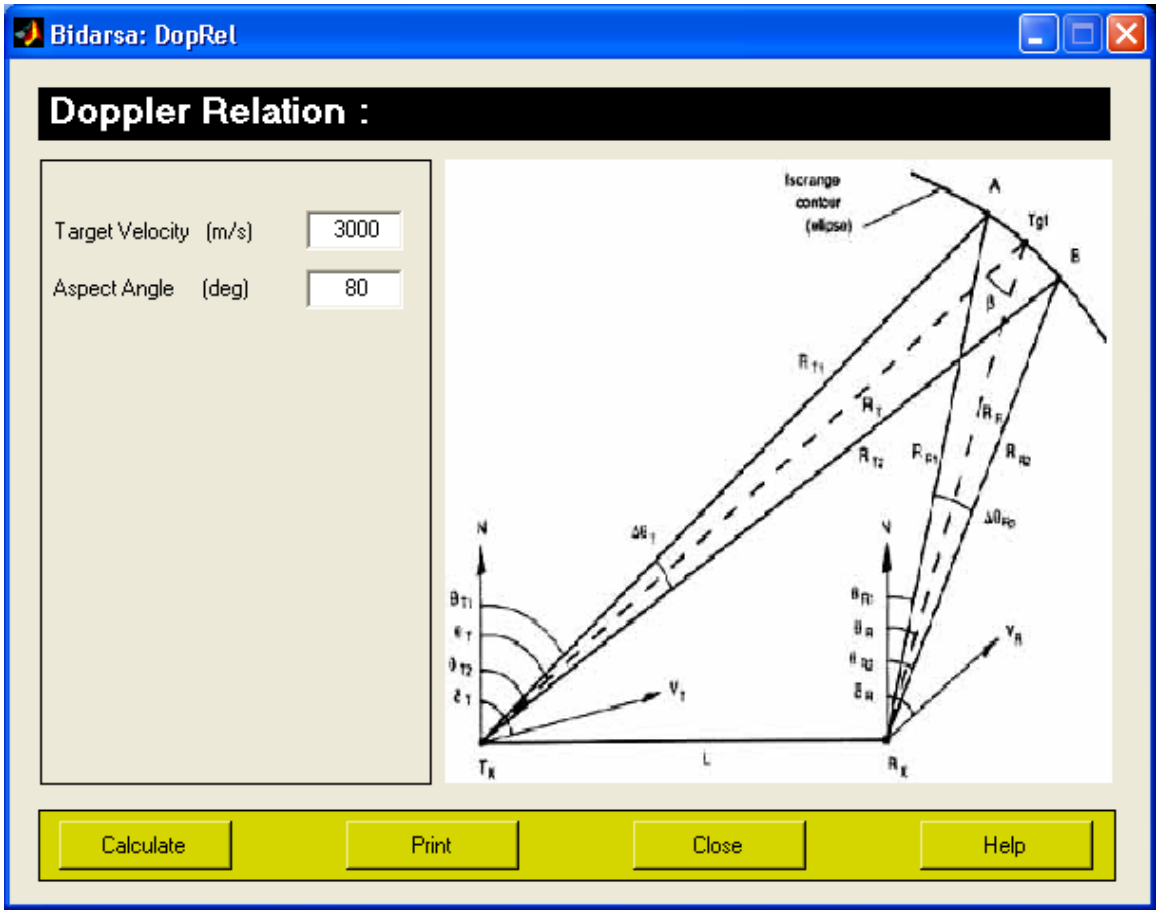

Figure 3.8 Doppler relation GUI.

I. EW EFFECTS MENU

This module is designed to evaluate the bistatic radar's performance when subjected to a jamming transmission. The user can define the jammer specifications for simulating different operational scenarios. Figure 3.9 shows the GUI layout designed for the module. It computes:

1. Bistatic maximum range product with jammer, as detailed in Equation (2.56).
2. System analysis for
   a. Jammer power variation.
   b. Target RCS variation.
   c. Burn-through range analysis.

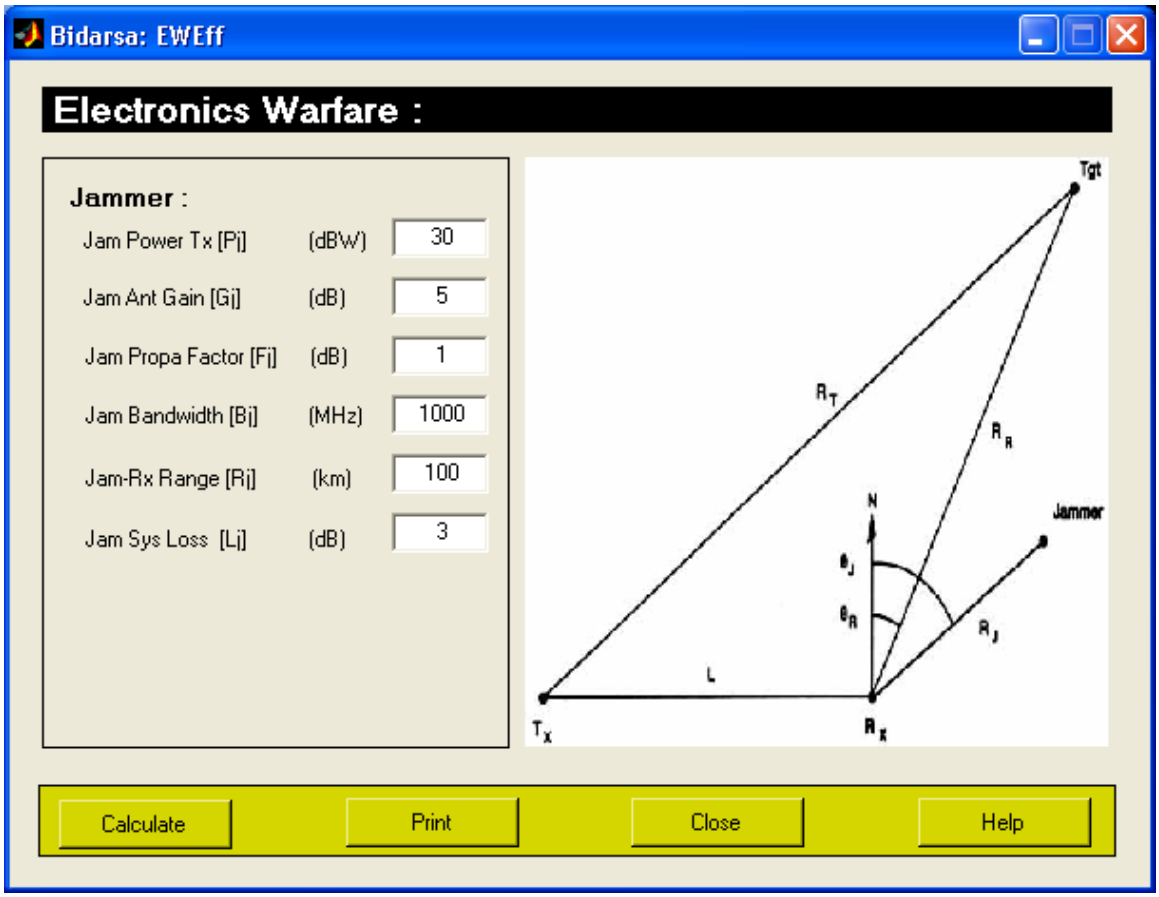

Figure 3.9 EW Effects GUI.

J. SUMMARY

The MATLAB GUI is used to develop the *Bidarsa* simulation program, as it offers easy user interfaces and excellent data visualization. Eight modules were developed and integrated as a complete suite. The *Bidarsa* program provides a user the options to specify the bistatic radar system parameters, the target information and environmental data. Several critical system performance measures are calculated, and data presentation tailored to the study of design tradeoffs. The bistatic doppler relationship and electronic countermeasures considerations are also included to provide extended analyzes. Chapter III provides sample analyzes on a set of radar parameters. Plots generated from the different modules will also be discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DESCRIPTION OF SOFTWARE OPERATION

A. TEST DATA

To evaluate the *Bidarsa* program, a set of system parameters were compiled from various published sources [1, 3] and used for a detailed system analysis. The test values are as shown in Table 1.

Transmitting power	50 dBW
Transmitter antenna gain	30 dB
Frequency	10 GHz
Receiver min SNR	0 dBm
Receiver antenna gain	30 dB
Receiver noise temperature	300 K
Receiver noise bandwidth	60 MHz
Baseline range	100 km
Target RCS	10 dBm^2
Bistatic beta angle	1 degree
Transmitter system loss	1 dB
Receiver system loss	1 dB
Tx-Tgt pattern propagation factor	−3.5 dB
Tgt-Rx pattern propagation factor	−3.5 dB

Table 1. Test parameters used to evaluate *Bidarsa* program.

The *Bidarsa* program is run and the above values are entered and updated in the *Defpara* menu. The results of all module calculations are displayed in the MATLAB Command Window.

B. BISTATIC RANGE PRODUCT ANALYSIS

Selecting the *Range Calculation* menu, the program computed the following results for the three range related parameters, as shown in Figure 4.1. Selecting one of the four system analysis buttons on the menu will generate the respective plot as shown in Figure 4.2. When the plot detection contour button is selected, the program generates a plot for the oval of Cassini for constant SNR at 93 dB, 96 dB, 98.65 dB and 102 dB, as shown in Figure 4.3.

Bidarsa - Range Calculation:

Bistatic Maximum Range Product R_tR_r is 6.7706e+006 km^2.

Bistatic Radar constant K is 4.5841e+016.

Maximum Range-Unambiguous PRF is 1.8299e+000 kHz.

Figure 4.1 Range calculation menu output.

From Equations (2.48) and (2.46), it is interesting to note that in a bistatic radar system, increasing the bistatic angle, β, will increase the maximum range-unambiguous PRF, which will in turn, reduce the bistatic radar maximum unambiguous range.

Figure 4.3 resembles Figure 2.3 in Reference [3]. With the system parameters used in this analysis, the lemniscate ($L = 2\sqrt{\kappa}$) occurs when the SNR is 98.65 dB; two separate ellipses enclosing the transmitter and receiver ($L > 2\sqrt{\kappa}$) are observed at a SNR of 102 dB; two single continuous ellipses ($L < 2\sqrt{\kappa}$) are observed at SNRs of 93 dB and 96 dB.

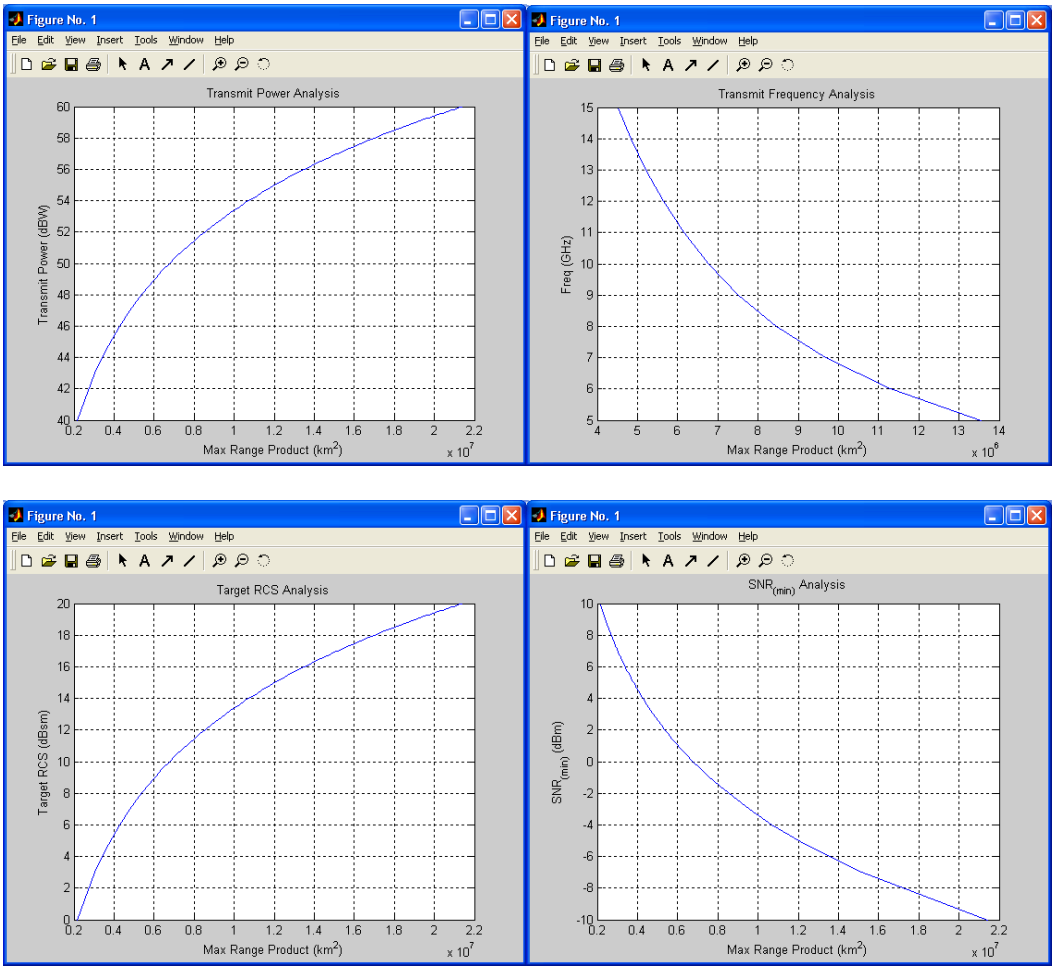

Figure 4.2 System analysis: transmit power (top left), frequency (top right), target RCS (bottom left) and SNR (bottom right).

Reference [3] also suggests a few possible military applications for each of the operating regions.

1. For $L > 2\sqrt{\kappa}$, with a small oval around the receiver, i.e., the receiver-centered region, the system can be used for

   a. Air-to-ground operations using a stand-off transmitter and penetrating aircraft with silent receiver.

   b. A semiactive homing missile.

2. For $L > 2\sqrt{\kappa}$, with a small oval around the transmitter, i.e., the transmitter-centered region, the system can be used for

   a. Monitoring of activity near a non-cooperative transmitter.

3. For $L < 2\sqrt{\kappa}$, with a single continuous ellipse, the system can be used for

   a. Medium-range air defense.

   b. Missile tracking from ground-based sites.

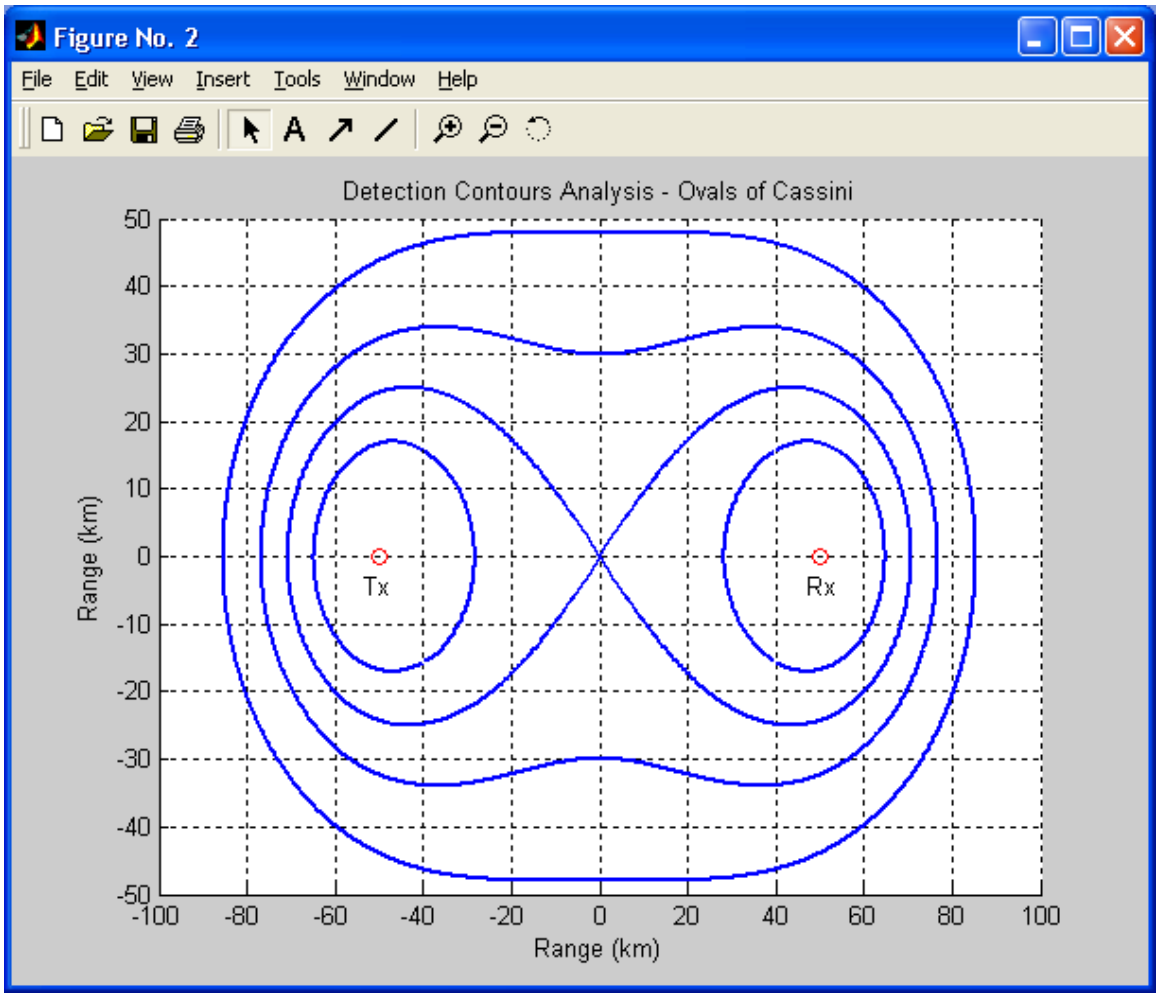

Figure 4.3 Constant SNR oval of Cassini plot.

C. TARGET LOCATION ANALYSIS

In the *Target Location* menu, test values for the range-sum and angle θ_R as shown in Table 2, are used for the evaluation. The program computed the results for the three target location related parameters, as shown in Figure 4.4.

Range-sum $(R_T + R_R)$	350 km
Angle θ_R	30 degrees

Table 2. Test values used for target location analysis.

```
Bidarsa - Target Location:

Theta_R Selected

Distance of Target to Transmitter is 2.0938e+002 km.

Distance of Target to Receiver      is 1.4063e+002 km.

The Beta Angle calculated           is 24.43 deg.
```

Figure 4.4 Target location menu output.

The user can toggle the angle θ entry and verify from the results that the isorange contour has to be an ellipse, rather than the tangent approximation. The possibility of using either θ_T or θ_R for computation offers more flexibility in the analysis.

D. COVERAGE AREA ANALYSIS

In the *Coverage Area* menu, the test values for the transmitter, receiver, and target altitude are shown in Table 3. The program computed the following results in determining the region of operation and the coverage areas, as shown in Figure 4.5. A system analysis of the transmitter-receiver-target height requirement versus baseline range is also generated as shown in Figure 4.6.

Target height	30 km
Transmitter antenna height	10 km
Receiver antenna height	10 km

Table 3. Test values used for coverage area analysis.

Bidarsa - Coverage Area:

Target is in the Single Cosite Region.

The Detection-constrained coverage area is 2.1271e+007 km^2.

Rx and Tx is in LOS.

The LOS-constrained coverage area is 3.7384e+006 km^2.

Figure 4.5 Coverage area menu output.

In this analysis, Figure 4.6 shows that, for a baseline range L of 100 km, an adequate LOS is established for a target height > 700 m, only if the transmitting or receiving antenna is > 100 m. If the baseline is decreased to 50 km (0.5L), an adequate LOS is established for a target height > 700 m, when the antenna is near zero altitude. However, if the range is increased to 1200 km (1.2L), to achieve an adequate LOS for target height > 700 m, the antenna will need to be raised to at least 180 m.

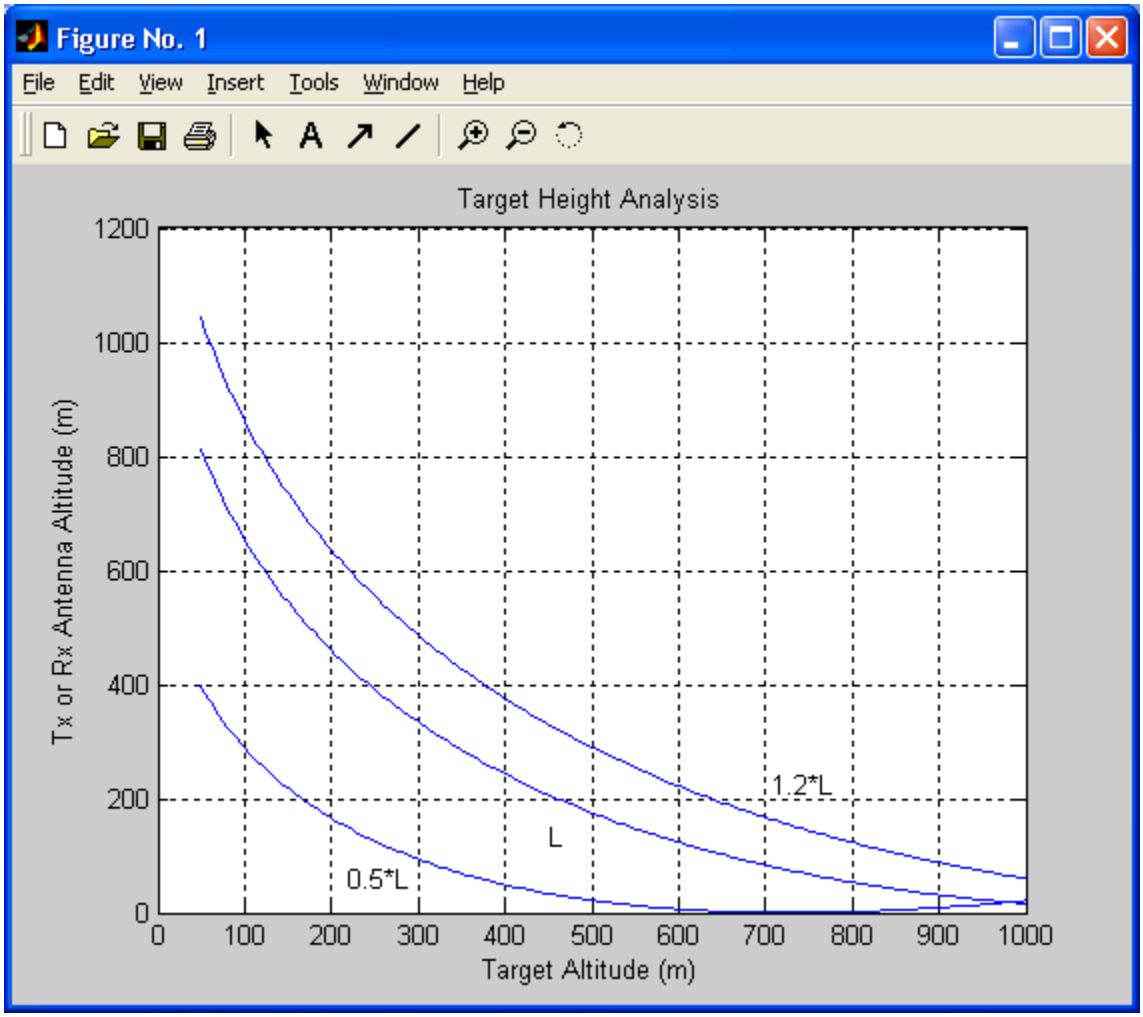

Figure 4.6 Transmitter-target-receiver height analysis.

E. BISTATIC FOOTPRINT ANALYSIS

In the *Bistatic Footprint & Clutter Area* menu, the radar parameters as shown in Table 4 are used for the evaluation. The program determines if the bistatic footprint is limited by beamwidth or pulsewidth, as shown in Figure 4.7. The range cell size and maximum error were also generated.

Receiver-target range	100 km
Transmitter-to-target range	50 km
Receiver antenna beamwidth	10 degrees
Transmitter antenna beamwidth	10 degrees
Transmitter pulsewidth	30 μsec

Table 4. Test parameters for bistatic footprint analysis.

Bidarsa - Bistatic Footprint:

This is a Beamwidth Limited case.

The Bistatic Footprint is 8.7711e+002 sqr-km.

Range Cell Size is 4.5172e+003 m.

Maximum Error is 2.2164e-002 m.

Figure 4.7 Bistatic footprint menu output.

It can be gathered from the Equations (2.42) to (2.45) that the radar pulsewidth plays a bigger role than the beamwidth in determining whether the bistatic footprint is limited by beamwidth or pulsewidth. This effect can be verified using this module.

It is also verified in this module that variation in the antenna beamwidth will only affect the beamwidth limited bistatic footprint area; it has a small effect on the range cell size and does not affect the maximum error calculated.

F. DOPPLER RELATION ANALYSIS

In the *Doppler Relation* menu, the test values shown in Table 5 are used for the evaluation. The program computes the bistatic doppler shift as shown in Figure 4.8. A system analysis for doppler frequency shift versus different target aspect angle and bistatic angle is generated as shown in Figure 4.9.

Target velocity	300 m/s
Target velocity aspect angle	80 degrees

Table 5. Test parameters used for doppler relationship analysis.

Bidarsa - Doppler Relation:

Bistatic Doppler Shift is 3.4597e+001 kHz.

Figure 4.8 Doppler Relation menu output.

An analysis of Figure 4.9 illustrates the following observations:

1. For all bistatic angles, β, if the target velocity aspect angle, δ, is between $-90°$ and +90°, there will be a positive bistatic target doppler.
2. For all bistatic angles, β, if the target aspect angle is normal to the bistatic bisector, $\delta = \pm 90°$, there will be zero bistatic target doppler.
3. For all bistatic angles, $\beta < 120°$, if the target aspect angle is collinear with the bistatic bisector, a maximum bistatic target doppler will occur when $\delta = 0°$ and a minimum will occur when $\delta = 180°$.

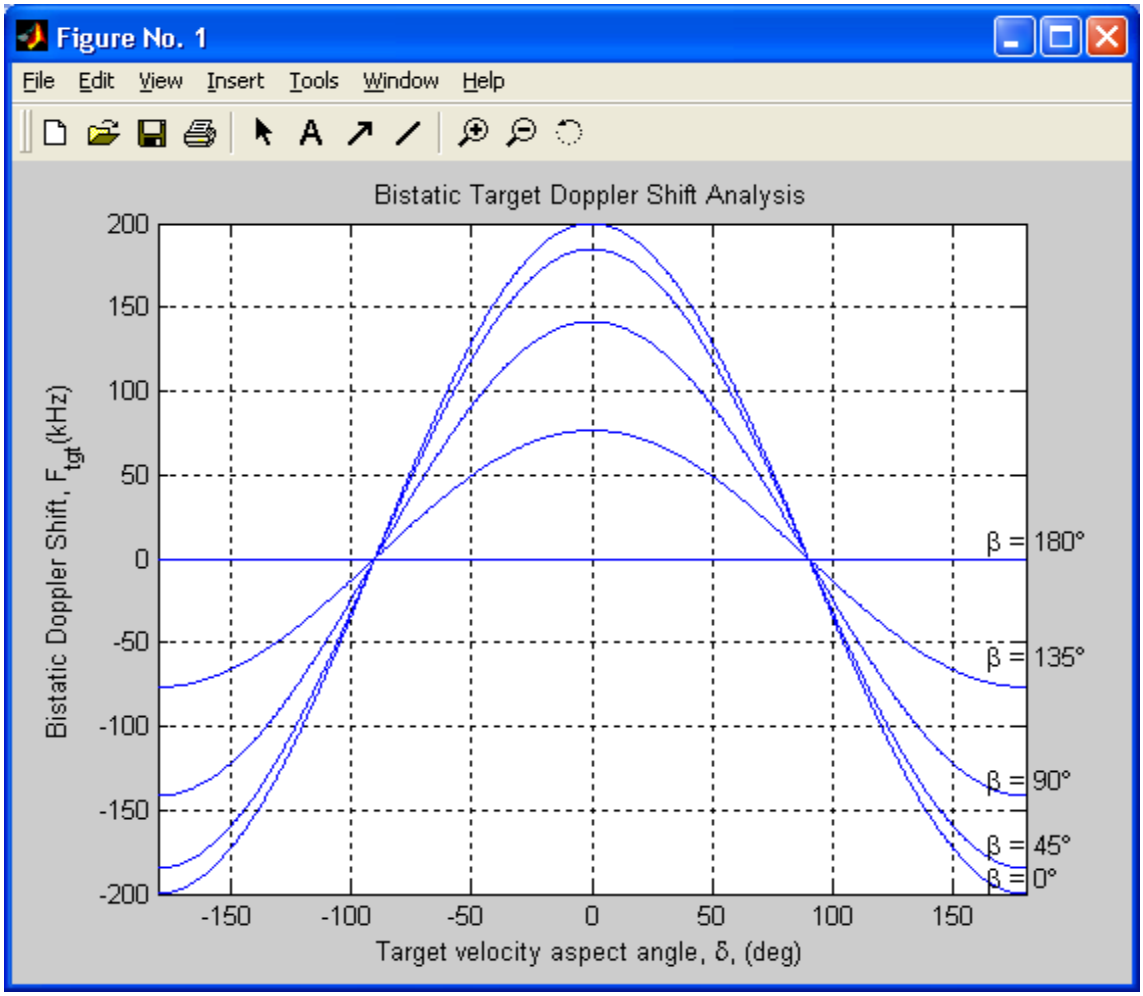

Figure 4.9 Doppler frequency shift analysis with different target aspect angle and bistatic angle.

G. ELECTRONIC COUTNERMEASURES ANALYSIS

In the *EW Effects* menu, the jammer specifications shown in Table 6 are used for the evaluation. The program computes the bistatic maximum range product with jamming as shown in Figure 4.10. A system analysis for the burn-through ranges with different jammer transmit powers and target RCS is generated in Figure 4.11.

Jammer transmitted power	30 dBW
Jammer antenna gain	5 dB
Jammer propagation factor	1 dB
Jammer bandwidth	1000 MHz
Jammer-receiver range	100 km
Jammer system loss	3 dB

Table 6. Test parameters used for electronic countermeasures analysis

Bidarsa - EW Effects:

Bistatic Max Range Product with Jamming $(R_t R_r)_j$ is 3.6418e+008 km^2.

Figure 4.10 EW Effects menu output.

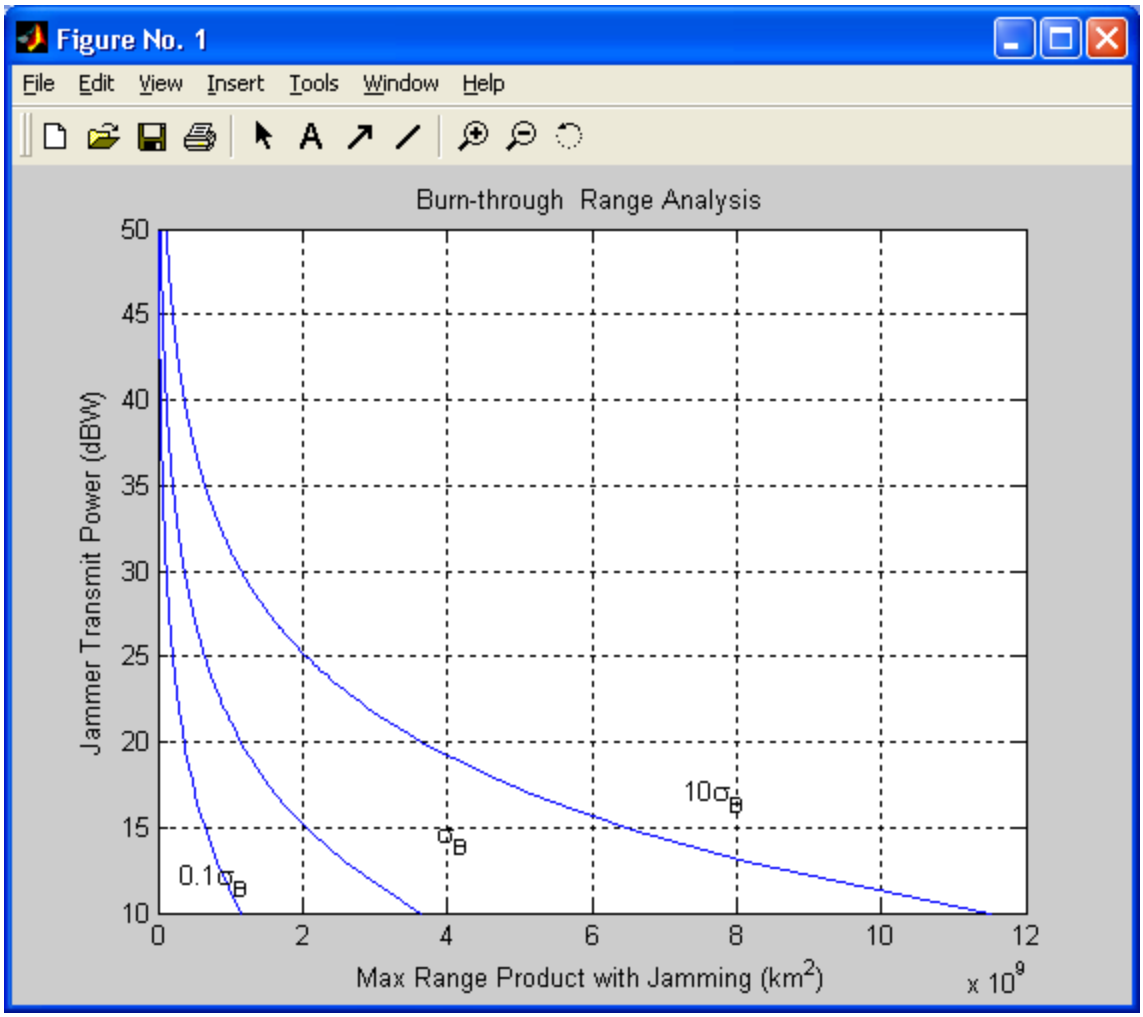

Figure 4.11 Burn-through range analysis with different target RCS.

H. SUMMARY

A set of radar parameters was used to demonstrate the simulation and analysis capability provided by the *Bidarsa* program. The simulation results from the different modules were discussed along with their respective plots. The *Bidarsa* program offers flexibility for the user to study how the individual parameters will affect the overall system performance. Chapter V presents more examples to illustrate how the *Bidarsa* program can be used to study specific bistatic configurations and scenarios.

V. SAMPLE SYSTEMS ANALYSIS

A. OVERVIEW

Chapter IV has shown the various components of the software simulation and analysis offered by the *Bidarsa* program. In this chapter, the use of the *Bidarsa* program is further illustrated with two specific bistatic radar applications.

B. MONITORING OF SHIPPING CHANNEL

Chapter 13 of Reference [3] has provided a scenario whereby an aircraft uses a bistatic radar system to monitor the shipping across a channel. A 1-GHz, 1-W transmitting buoy is dropped in a shipping channel such that ships must pass within 1 km of it. The aircraft is flying directly overhead listening for the echoes with a receiving antenna gain of 20 dB. The buoy has a unity antenna gain and the aircraft receiver requires a detection signal strength of -130 dBW. The bistatic RCS of a ship is 30 dBsm, and all the losses and multipath effects are neglected.

The above parameters were entered into the *Bidarsa* program for the analysis. The results are tabulated as shown in Table 7.

Bistatic maximum range product, $R_t R_r$	6.7359 km^2
Bistatic radar constant, K	4.5372e-006
Distance of target-to-transmitter	1 km
Distance of target-to-receiver	6.7359 km
Beta angle	82.17 degrees

Table 7. Results for shipping-channel example.

The above bistatic radar result was used to compare its performance to an aircraft using monostatic radar in the same scenario. The transmitting antenna is assumed to

have a unity gain for consistency. The monostatic radar will give a maximum detection range of only 2.595 km, due to the factor of R^4 in the monostatic radar range equation. The bistatic radar system in this application has a range advantage over the monostatic radar.

This comparison has implied that by using the bistatic radar configuration, the aircraft could fly at a much higher altitude to monitor the shipping channel without being detected. On the other hand, if the monostatic radar uses a higher gain antenna, it will make up for the monostatic R^4 factor and require less transmitting power for the same range performance. However, the monostatic radar configuration, being an active system, might possibly give away the position of the aircraft.

The transmitting power, system frequency, target RCS, and SNR_{min} versus maximum range product are shown in Figure 5.1.

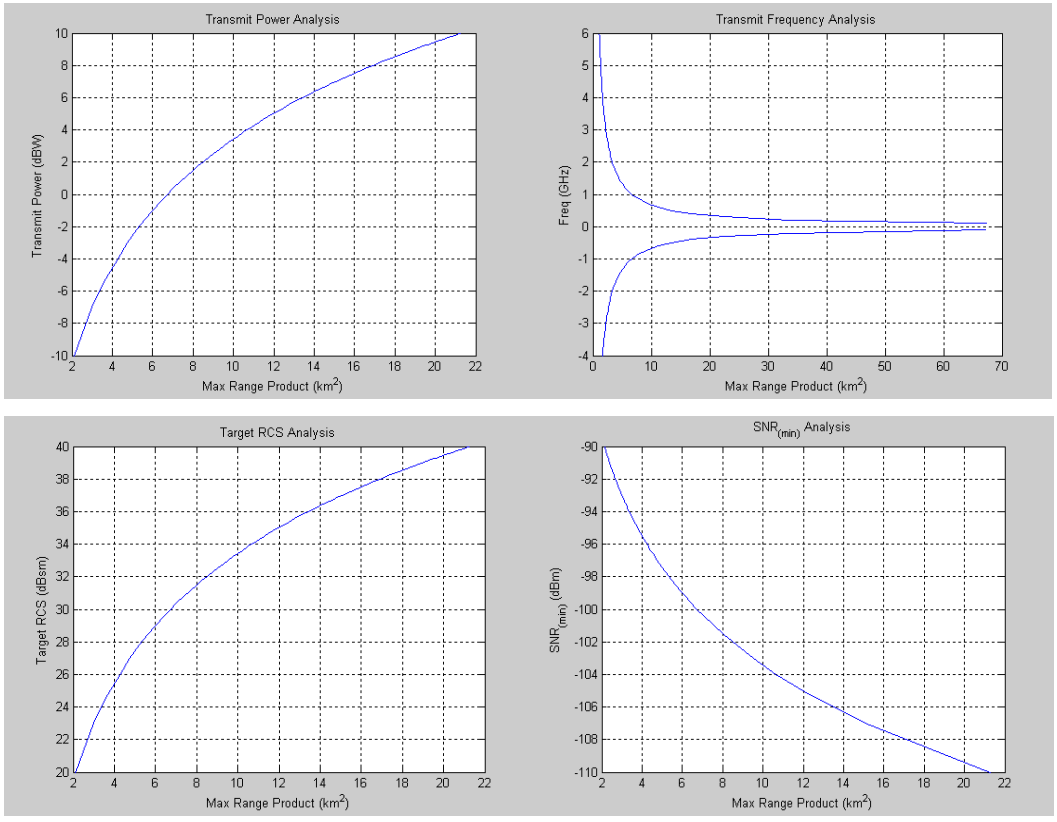

Figure 5.1 System analysis for shipping-channel example: transmit power (top left), frequency (top right), target RCS (bottom left) and SNR (bottom right).

Depending on the requirements of the aircraft operation profile, Figure 5.1 illustrates the tradeoffs in the system design parameters, so as to achieve a higher probability of detection and aircraft survivability.

C. NPS BISTATIC RADAR EXPERIMENT

An experiment was conducted at the Naval Postgraduate School (NPS) in 1997 to investigate the feasibility of using direct television broadcast satellites and ground-based bistatic receivers to detect targets [4]. With the Hughes Direct Broadcast Satellite (DBS-1) used to illuminate the target, receive signals from the direct path antenna and bistatic receive antenna were compared and processed to detect the presence of targets. The system configuration is shown in Figure 5.2.

The system parameters used for the experiment are as shown in Table 8. The parameters were entered into the Bidarsa program for the analysis. The resulting bistatic maximum range product is 74543 km^2, and the estimated RR is approximately 2 km. This is in agreement with the result obtained in the experiment conducted.

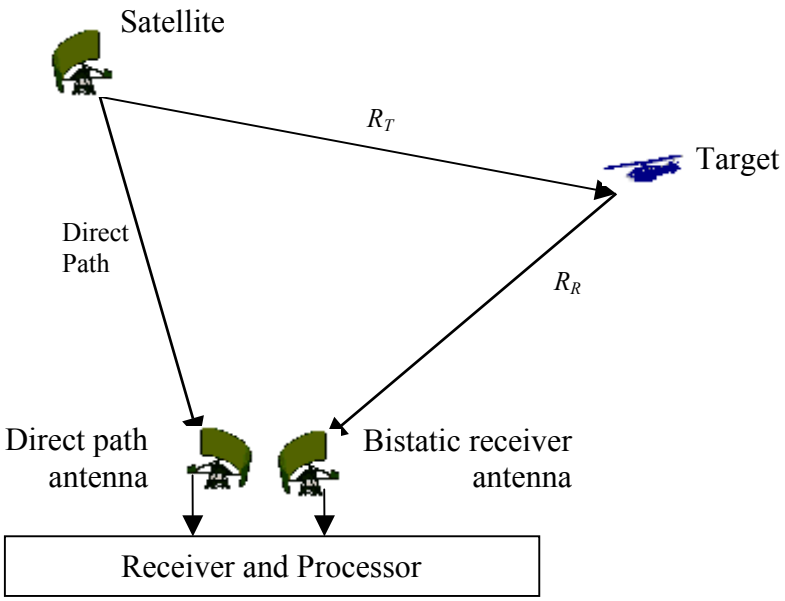

Figure 5.2 NPS bistatic radar experiment configuration.

Effective radiated power of DBS-1, $P_T G_T$	53 dBW
Transmitting frequency, f	12.7 GHz
Effective aperture of receiver antenna, A_e	0.16 m^2
Processor gain of radar, G_P	61 dB
Receiver bandwidth, B_n	45 MHz
Minimum SNR required for detection, $(S/N)_{min}$	13 dB
Distance between transmitter and receiver, L	37000 km
Target bistatic RCS, σ_B	30 dBsm
Two way Losses, $L_t L_r$	6 dB

Table 8. System parameters for the NPS bistatic radar experiment.

The experimental result confirmed the feasibility of the concept, and suggested several improvement factors [4] that could employed to increase the detection range. They are:

1. Using of low-noise receiver to improve the SNR.
2. Increase the receiver antenna area and gain.
3. Increase the effective radiated power on the transmitting satellite.
4. Using multiple receivers to increase the processing gain.

It is important to take note that in the bistatic-radar-satellite-hitchhiker system, each 6-dB increase in SNR doubles the receiver-to-target range, R_R. This is much better than the monostatic system, whereby it takes a 12-dB increase in SNR to double the detection range.

D. SUMMARY

In this chapter, two different applications were presented and the *Bidarsa* program was used to carry out the system performance prediction. The results were analyzed and compared with the monostatic radar system. Recommendations were also discussed to improve the system performance. The final chapter will present a overall summary of this thesis and suggest possible follow up work.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. SUMMARY AND CONCLUSIONS

A. SUMMARY

Bistatic radar offers many advantages over monostatic radar in certain applications, since the receiving antenna is at a distant site with reference to the transmitting antenna. This thesis achieved the objective of developing a software model for bistatic radar systems. The software allows a user to examine the various system design parameters and evaluate the system performance tradeoffs, under different conditions and scenarios, in a simulated environment.

MATLAB GUI is chosen as the software to develop the *Bidarsa* simulation program because it is an effective interactive tool for both numerical computation and data visualization. It also allows a very user-friendly interface to be designed and operated.

The *Bidarsa* program is easy to learn and navigate with Help menus designed to guide the user in every module. It is suitable to be used by any engineering student or professionals who are interested in the bistatic radar field.

With the current operation trends toward a lower signature target and the third resurgence for bistatic radar using Passive Coherent Location [8], the bistatic radar offers a much greater potential now than at any time before. The *Bidarsa* program will help engineering students to understand the fundamental design principles of bistatic radar, and provide a convenient tool for the professional in this field to analyze the system performance. The software is available free of charge and can be downloaded from www.nps.navy.mil/jenn.

B. RECOMMENDATIONS

While this thesis focused on the effort to develop a software model to evaluate the essential performance of a bistatic radar system, there are other areas that could be examined to provide a more advanced analysis to predict the bistatic radar performance. They include:

1. Examine the effects of bistatic RCS for surface clutter, which consists of both ground and sea echoes.

2. Clutter tuning concepts that cater to a moving transmitting and receiver.

3. Pulse chasing concepts to reduce the complexity and cost of multibeam bistatic receivers.

4. Passive Coherent Location (PCL) considerations [8], such as the CELLDAR concept shown in Figure 6.1. CELLDAR is a passive system based on the bistatic radar principle. It works by using existing transmissions made by digital cell phone base stations. When a target object enters the detection region, these cell phone transmissions are reflected by it and detected at the CELLAR antennas.

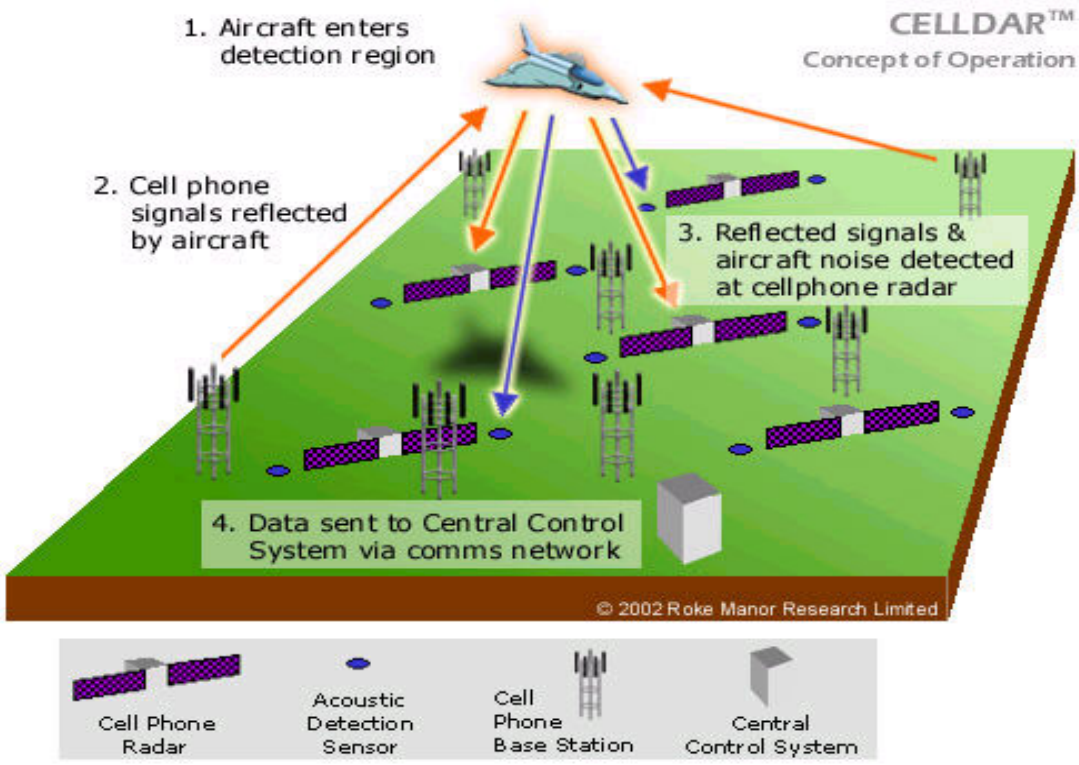

Figure 6.1 CELLDAR concept of operation proposed by Roke Manor. (From Ref. [9].)

APPENDIX

A. MATLAB CODE FOR *BIDARSA.M*

This appendix details the *Bidarsa.m* code developed to allow a user to select one of the three group functions: (1) specify the system parameters, (2) carry out a basic system performance analysis, or (3) carry out an extended system performance analysis.

```
function varargout = Bidarsa(varargin)
% BIDARSA M-file for Bidarsa.fig
%      BIDARSA, by itself, creates a new BIDARSA or raises the existing
%      singleton*.
%      H = BIDARSA returns the handle to a new BIDARSA or the handle to
%      the existing singleton*.
%      BIDARSA('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in BIDARSA.M with the given input arguments.
%      BIDARSA('Property','Value',...) creates a new BIDARSA or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Bidarsa_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Bidarsa_OpeningFcn via varargin.
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help Bidarsa
% Last Modified by GUIDE v2.5 20-Oct-2003 17:08:59
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Bidarsa_OpeningFcn, ...
                   'gui_OutputFcn',  @Bidarsa_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(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 Bidarsa is made visible.
function Bidarsa_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
clear all;
clc;


% --- Outputs from this function are returned to the command line.
function varargout = Bidarsa_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;


%**********************************************************************
% --- Executes on button press in pushbutton00.
function pushbutton00_Callback(hObject, eventdata, handles)
Defpara

% --- Executes on button press in pushbutton11.
function pushbutton11_Callback(hObject, eventdata, handles)
RngCal

% --- Executes on button press in pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)
TgtLoc

% --- Executes on button press in pushbutton13.
function pushbutton13_Callback(hObject, eventdata, handles)
CovAre

% --- Executes on button press in pushbutton14.
function pushbutton14_Callback(hObject, eventdata, handles)
FotPrt

% --- Executes on button press in pushbutton21.
function pushbutton21_Callback(hObject, eventdata, handles)
DopRel

% --- Executes on button press in pushbutton31.
function pushbutton31_Callback(hObject, eventdata, handles)
EWEff
%**********************************************************************
```

```
% --- Executes on button press in Close.
function Close_Callback(hObject, eventdata, handles)
close all hidden

% --- Executes on button press in Help.
function Help_Callback(hObject, eventdata, handles)
yHlpMain
%**************************************************************************
```

B. MATLAB CODE FOR *DEFPARA.M*

This appendix details the *Defpara.m* code developed to provide a convenient way for user to specify the bistatic radar's transmitter and receiver design parameters, the target information and the environmental information.

```
function varargout = Defpara(varargin)
% DEFPARA M-file for Defpara.fig
%      DEFPARA, by itself, creates a new DEFPARA or raises the existing
%      singleton*.
%      H = DEFPARA returns the handle to a new DEFPARA or the handle to
%      the existing singleton*.
%      DEFPARA('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in DEFPARA.M with the given input arguments.
%      DEFPARA('Property','Value',...) creates a new DEFPARA or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Defpara_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Defpara_OpeningFcn via varargin.
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help Defpara
% Last Modified by GUIDE v2.5 24-Oct-2003 01:25:10
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Defpara_OpeningFcn, ...
                   'gui_OutputFcn',  @Defpara_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(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 Defpara is made visible.
function Defpara_OpeningFcn(hObject, eventdata, handles, varargin)
```

```
handles.output = hObject;
guidata(hObject, handles);
clc;


% --- Outputs from this function are returned to the command line.
function varargout = Defpara_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;


% --- Executes during object creation, after setting all properties.
function Pt_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Pt_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Gt_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Gt_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Fq_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Fq_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function SN_input_CreateFcn(hObject, eventdata, handles)
if ispc
```

```
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function SN_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Gr_input_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Gr_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Ts_input_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Ts_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Bn_input_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Bn_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Br_input_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
```

```
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Br_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Sm_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Sm_input_Callback(hObject, eventdata, handles)


function Ba_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Ba_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Ft_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Ft_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Fr_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function Fr_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Lt_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Lt_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Lr_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Lr_input_Callback(hObject, eventdata, handles)


%**************************************************************************
% --- Executes on button press in Update.
function Update_Callback(hObject, eventdata, handles)
global Pt_00 Gt_00 Fq_00 SN_00 Pt_00 Gr_00 Ts_00 Bn_00 Br_00...
      Sm_00 Ba_00 Ft_00 Fr_00 Lt_00 Lr_00 Lambda_00
clc;
Pt_input = eval(get(handles.Pt_input,'String'));
Gt_input = eval(get(handles.Gt_input,'String'));
Fq_input = eval(get(handles.Fq_input,'String'));
SN_input = eval(get(handles.SN_input,'String'));
Gr_input = eval(get(handles.Gr_input,'String'));
Ts_input = eval(get(handles.Ts_input,'String'));
Bn_input = eval(get(handles.Bn_input,'String'));
Br_input = eval(get(handles.Br_input,'String'));
Sm_input = eval(get(handles.Sm_input,'String'));
Ba_input = eval(get(handles.Ba_input,'String'));
Ft_input = eval(get(handles.Ft_input,'String'));
Fr_input = eval(get(handles.Fr_input,'String'));
Lt_input = eval(get(handles.Lt_input,'String'));
Lr_input = eval(get(handles.Lr_input,'String'));
```

```
Pt_00 = 10^(Pt_input/10);
Gt_00 = 10^(Gt_input/10);
Fq_00 = Fq_input*1e9;
SN_00 = (10^((SN_input-30)/10));
Ba_00 = deg2rad(Ba_input);
Gr_00 = 10^(Gr_input/10);
Ts_00 = Ts_input;
Bn_00 = Bn_input*1e6;
Br_00 = Br_input;
Sm_00 = 10^(Sm_input/10);
Ft_00 = 10^(Ft_input/10);
Fr_00 = 10^(Fr_input/10);
Lt_00 = 10^(Lt_input/10);
Lr_00 = 10^(Lr_input/10);

Lambda_00 = 3e8/(Fq_00);                                %System Wavelength
mb0 = msgbox(['     New bistatic radar parameters loaded.     '],'Result');
%**************************************************************************


% --- Executes on button press in Print.
function Print_Callback(hObject, eventdata, handles)
printdlg('-setup',handles.figure1)

% --- Executes on button press in Close.
function Close_Callback(hObject, eventdata, handles)
delete(handles.figure1)

% --- Executes on button press in Help.
function Help_Callback(hObject, eventdata, handles)
yHlpDefpara
%**************************************************************************
```

C. MATLAB CODE FOR *RNGCAL.M*

This appendix details the *Rngcal.m* code developed to compute the bistatic maximum range product, bistatic radar constant, maximum range-unambiguous PRF, plotting of the detection contour, and carry out system analysis on four design parameters.

```
function varargout = RngCal(varargin)
% RNGCAL M-file for RngCal.fig
%      RNGCAL, by itself, creates a new RNGCAL or raises the existing
%      singleton*.
%      H = RNGCAL returns the handle to a new RNGCAL or the handle to
%      the existing singleton*.
%      RNGCAL('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in RNGCAL.M with the given input arguments.
%      RNGCAL('Property','Value',...) creates a new RNGCAL or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before RngCal_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to RngCal_OpeningFcn via varargin.
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help RngCal
% Last Modified by GUIDE v2.5 06-Nov-2003 08:56:10
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @RngCal_OpeningFcn, ...
                   'gui_OutputFcn',  @RngCal_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(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 RngCal is made visible.
function RngCal_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
clc;
axes(handles.axes1);
[x1,map] = imread('z-Range.jpg','jpg');
image(x1);
colormap(map);
set(handles.axes1,'Visible','off');
global PowP FreP RCSP SNRP
PowP = 0;
FreP = 0;
RCSP = 0;
SNRP = 0;


% --- Outputs from this function are returned to the command line.
function varargout = RngCal_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;


% --- Executes on mouse press over figure background, over a disabled or
% --- inactive control, or over an axes background.
function figure1_WindowButtonDownFcn(hObject, eventdata, handles)


% --- Executes on key press over figure1 with no controls selected.
function figure1_KeyPressFcn(hObject, eventdata, handles)


%**************************************************************************
% --- Executes on button press in Calculate.
function Calculate_Callback(hObject, eventdata, handles)
global Pt_00 Gt_00 Fq_00 SN_00 Pt_00 Gr_00 Ts_00 Bn_00 Br_00...
      Sm_00 Ba_00 Ft_00 Fr_00 Lt_00 Lr_00 Lambda_00 RtRr11 BRP11...
      PowP FreP RCSP SNRP ConP
clc;
k = 1.38065*10^(-23);
Num11 = (Pt_00*Gt_00*Gr_00*Lambda_00^2*Sm_00*Ft_00^2*Fr_00^2);
Den11 = ((4*pi)^3*k*Ts_00*Bn_00*SN_00*Lt_00*Lr_00);
RtRr11 = ((Num11/Den11)^(0.5))/1000;                %Bistatic Max Range Product in km
Den12 = ((4*pi)^3*k*Ts_00*Bn_00*Lt_00*Lr_00);
BRP11 = (Num11/Den12);                              %Bistatic Radar Parameter in km
PRFu11 = 3e8/((Br_00^2+(2*(RtRr11*1000)*(1+cos(Ba_00))))^0.5);
PRFu12 = PRFu11/1000;
```

```
fprintf('\nBidarsa - Range Calculation:\n');
fprintf('Bistatic Maximum Range Product RtRr is %-8.4e km^2.\n',RtRr11);
fprintf('Bistatic Radar constant        K    is %-8.4e.\n\n',BRP11);
fprintf('Maximum Range-Unambiguous PRF       is %-8.4e kHz.\n\n',PRFu12);


%*****************************************************************************
close(figure(1));
%Plotting Transmit Power Analysis
if (PowP==1)
   Pt_11   = (Pt_00/10):(Pt_00/10):(Pt_00*10);
   Num111  = (Gt_00.*Pt_11*Gr_00*Lambda_00^2*Sm_00*Ft_00^2*Fr_00^2);
   Den111  = ((4*pi)^3*k*Ts_00*Bn_00*SN_00*Lt_00*Lr_00);
   RtRr111 = (((Num111./Den111).^(0.5))/1000);    %Bistatic Max Range Product in km
   Pt_112  = 10.*log10(Pt_11);
   figure(1);
   plot(RtRr111,Pt_112);
   title('Transmit Power Analysis');
   xlabel('Max Range Product (km^2)');
   ylabel('Transmit Power (dBW)');
   grid on;
end

%Plotting Tansmit Frequency Analysis
if (FreP==1)
   Fq_12     = (Fq_00-5e9):(Fq_00/10):(Fq_00+5e9);
   Lambda_12 = 3e8./Fq_12;
   Num121    = (Pt_00*Gt_00*Gr_00.*Lambda_12.^2*Sm_00*Ft_00^2*Fr_00^2);
   Den121    = ((4*pi)^3*k*Ts_00*Bn_00*SN_00*Lt_00*Lr_00);
   RtRr121   = (((Num121./Den121).^(0.5))/1000);   %Bistatic Max Range Product in km
   Fq_121    = Fq_12./1e9;
   figure(1);
   plot(RtRr121,Fq_121);
   title('Transmit Frequency Analysis');
   xlabel('Max Range Product (km^2)');
   ylabel('Freq (GHz)');
   grid on;
end

%Plotting Target RCS Analysis
if (RCSP==1)
   Sm_13   = (Sm_00/10):(Sm_00/10):(Sm_00*10);
   Num131  = (Pt_00*Gt_00*Gr_00*Lambda_00^2.*Sm_13*Ft_00^2*Fr_00^2);
   Den131  = ((4*pi)^3*k*Ts_00*Bn_00*SN_00*Lt_00*Lr_00);
   RtRr131 = (((Num131./Den131).^(0.5))/1000);     %Bistatic Max Range Product in km
```

```
   Sm_121  = 10.*log10(Sm_13);
   figure(1);
   plot(RtRr131,Sm_121);
   title('Target RCS Analysis');
   xlabel('Max Range Product (km^2)');
   ylabel('Target RCS (dBsm)');
   grid on;
end

%Plotting Receiver SNR Analysis
if (SNRP==1)
   SN_14   = (SN_00/10):(SN_00/10):(SN_00*10);
   Num141  = (Pt_00*Gt_00*Gr_00*Lambda_00^2*Sm_00*Ft_00^2*Fr_00^2);
   Den141  = ((4*pi)^3*k*Ts_00*Bn_00.*SN_14*Lt_00*Lr_00);
   RtRr141 = (((Num141./Den141).^(0.5))/1000);    %Bistatic Max Range Product in km
   SN_141  = (10.*log10(SN_14))+30;               %Converting dB to dBm
   figure(1);
   plot(RtRr141,SN_141);
   title('SNR_{(min)} Analysis');
   xlabel('Max Range Product (km^2)');
   ylabel('SNR_{(min)} (dBm)');
   grid on;
end


%**********************************************************************
% --- Executes on button press in Pow_plot_input.
function Pow_plot_input_Callback(hObject, eventdata, handles)
set(hObject, 'Value', 1);                          %Turn this button on
set(handles.Fre_plot_input, 'Value', 0);           %Turn off all the buttons
set(handles.RCS_plot_input, 'Value', 0);           %Turn off all the buttons
set(handles.SNR_plot_input, 'Value', 0);           %Turn off all the buttons
global PowP FreP RCSP SNRP
PowP = 1;
FreP = 0;
RCSP = 0;
SNRP = 0;

% --- Executes on button press in Fre_plot_input.
function Fre_plot_input_Callback(hObject, eventdata, handles)
set(hObject, 'Value', 1);                          %Turn this button on
set(handles.Pow_plot_input, 'Value', 0);           %Turn off all the buttons
set(handles.RCS_plot_input, 'Value', 0);           %Turn off all the buttons
set(handles.SNR_plot_input, 'Value', 0);           %Turn off all the buttons
global PowP FreP RCSP SNRP
PowP = 0; %get(handles.Pow_plot_input, 'Value');
```

```
FreP = 1; %get(hObject,'Value');
RCSP = 0;
SNRP = 0;

% --- Executes on button press in RCS_plot_input.
function RCS_plot_input_Callback(hObject, eventdata, handles)
set(hObject, 'Value', 1);                          %Turn this button on
set(handles.Pow_plot_input, 'Value', 0);           %Turn off all the buttons
set(handles.Fre_plot_input, 'Value', 0);           %Turn off all the buttons
set(handles.SNR_plot_input, 'Value', 0);           %Turn off all the buttons
global PowP FreP RCSP SNRP
PowP = 0;
FreP = 0;
RCSP = 1;
SNRP = 0;

% --- Executes on button press in SNR_plot_input.
function SNR_plot_input_Callback(hObject, eventdata, handles)
set(hObject, 'Value', 1);                          %Turn this button on
set(handles.Pow_plot_input, 'Value', 0);           %Turn off all the buttons
set(handles.Fre_plot_input, 'Value', 0);           %Turn off all the buttons
set(handles.RCS_plot_input, 'Value', 0);           %Turn off all the buttons
global PowP FreP RCSP SNRP
PowP = 0;
FreP = 0;
RCSP = 0;
SNRP = 1;

% --- Executes on button press in Con_plot_input.
function Con_plot_input_Callback(hObject, eventdata, handles)
close(figure(2));
Calculate_Callback;
Cass


%*****************************************************************
% plot ovals of cassini for bistatic radar SNR
function Cass
global BRP11 Br_00
rad=pi/180;
i=0;
L=Br_00;
K=BRP11;
delt=.1;
a0 = 10*log10(16*K/(L^4));
a1 = round(a0);
```

```
am6 = a1-6;
am3 = a1-3;
ap3 = a1+3;
snr=[am6 am3 a0 ap3];

for isnr=1:length(snr)
  SNR=10^(snr(isnr)/10);
  i=i+1; k=0;
% setup for b~=a
  a=L/2; b4=K/SNR; b=b4^(.25);

if b>=a
  for thd=0:delt:360
    thr=thd*rad;
    % b>a is one curve
    if b>a
      k=k+1;
      T(k)=thr;
      Rsq=a^2*(cos(2*thr)+sqrt(cos(2*thr)^2-1+(b/a)^4));
      r1(k)=sqrt(Rsq);
      r2(k)=0;
    end
    % b=a is lemniscate
    if b==a,
       k=k+1;
       T(k)=thr;
       Rsq=a^2*2*cos(2*thr); r1(k)=sqrt(Rsq); r2(k)=0;
    end
  end
  figure(2)
  if i==1, hold on; plot(-a,0,'or',a,0,'or'); end
  X=r1.*cos(T); Y=r1.*sin(T);
  plot(X,Y);
end
end
clear r1 r2 T X Y
% b<a is two curves
% this covers b<a in which case 8 separate curves are generated
if b<a
  for curv=1:4
    th1=[0 90 180 270];                          %lower limit
    th2=[90 180 270 360];                        %upper limit
    k=0;
    % decrease angle increment for these small curves
    for thd=th1(curv):delt/10:th2(curv)
      thr=thd*rad;
```

```
      R1sq=a^2*(cos(2*thr)+sqrt(cos(2*thr)^2-1+(b/a)^4));
      R2sq=a^2*(cos(2*thr)-sqrt(cos(2*thr)^2-1+(b/a)^4));
      R1=sqrt(R1sq); R2=sqrt(R2sq);
      if imag(R1)==0 & imag(R2)==0                    %R1~= conj(R2)
        k=k+1;
        T(k)=thr;
        r1(k)=(R1); r2(k)=(R2);
      end
    end
    figure(2); X=r1.*cos(T); Y=r1.*sin(T);
    plot(X,Y);
    clear X Y
    if i==1, hold on; end
    X=r2.*cos(T); Y=r2.*sin(T);
    plot(X,Y);
    grid on;
    clear r1 r2 T X Y
  end
end
title('Detection Contours Analysis - Ovals of Cassini');
xlabel('Range (km)');
ylabel('Range (km)');
axes('position', [0 0 1 1],'visible','off');
text(0.31,0.48,'Tx');
text(0.70,0.48,'Rx');
hold off
fprintf('Ovals of Cassini illustrate the contour plots for S/N at:\n');
fprintf('%-4.2fdB, %-4.2fdB, %-4.2fdB & %-4.2fdB.\n\n',am6, am3, a0, ap3);
%*********************************************************************


% --- Executes on button press in Print.
function Print_Callback(hObject, eventdata, handles)
printdlg('-setup',handles.figure1)

% --- Executes on button press in Close.
function Close_Callback(hObject, eventdata, handles)
delete(handles.figure1)
close(figure(1))
close(figure(2))

% --- Executes on button press in Help.
function Help_Callback(hObject, eventdata, handles)
yHlpRngCal
%*********************************************************************
```

D. MATLAB CODE FOR *TGTLOC.M*

This appendix details the *TgtLoc.m* code developed to compute the location of the target.

```
function varargout = TgtLoc(varargin)
% TGTLOC M-file for TgtLoc.fig
%      TGTLOC, by itself, creates a new TGTLOC or raises the existing
%      singleton*.
%      H = TGTLOC returns the handle to a new TGTLOC or the handle to
%      the existing singleton*.
%      TGTLOC('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TGTLOC.M with the given input arguments.
%      TGTLOC('Property','Value',...) creates a new TGTLOC or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before TgtLoc_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to TgtLoc_OpeningFcn via varargin.
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help TgtLoc
% Last Modified by GUIDE v2.5 29-Oct-2003 15:06:14
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @TgtLoc_OpeningFcn, ...
                   'gui_OutputFcn',  @TgtLoc_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(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 TgtLoc is made visible.
function TgtLoc_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
```

```
guidata(hObject, handles);
global TR TT
axes(handles.axes1);
[x1,map] = imread('z-TgtLoc.jpg','jpg');
image(x1);
colormap(map);
set(handles.axes1,'Visible','off');
TR = 1;
TT = 0;


% --- Outputs from this function are returned to the command line.
function varargout = TgtLoc_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;


% --- Executes during object creation, after setting all properties.
function BR_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function BR_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function TR_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function TR_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function TA_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function TA_input_Callback(hObject, eventdata, handles)


% --- Executes on button press in Theta_R_input.
function Theta_R_input_Callback(hObject, eventdata, handles)
set(hObject, 'Value', 1);                          %Turn this button on
set(handles.Theta_T_input, 'Value', 0);            %Turn off all the buttons
global TR TT
TR = get(hObject,'Value');
TT = get(handles.Theta_T_input, 'Value');


% --- Executes on button press in Theta_T_input.
function Theta_T_input_Callback(hObject, eventdata, handles)
set(hObject, 'Value', 1);                          %Turn this button on
set(handles.Theta_R_input, 'Value', 0);            %Turn off all the buttons
global TR TT
TR = get(handles.Theta_R_input, 'Value');
TT = get(hObject,'Value');


%**********************************************************************
% --- Executes on button press in Calculate.
function Calculate_Callback(hObject, eventdata, handles)
global Br_00 TR TT
clc;
TR_input = eval(get(handles.TR_input,'String'));
TA_input = eval(get(handles.TA_input,'String'));

BR_21 = Br_00;                                     %Base Range in km
TR_21 = TR_input;
TA_21 = deg2rad(TA_input);

Elli  = BR_21/TR_21;
Num21 = BR_21*(1 - Elli^2);
Den21 = 2*Elli*(1 + Elli * sin(TA_21));
Rr21  = Num21/Den21;                               %Tgt Range to Tx in km
Num22 = BR_21*(Elli^2 + 1 + (2*Elli*sin(TA_21)));
Rt21  = Num22/Den21;                               %Tgt Range to Rx in km

Den23 = 2*Elli*(1 - Elli * sin(TA_21));
Rt22  = Num21/Den23;
Num24 = BR_21*(Elli^2 + 1 - (2*Elli*sin(TA_21)));
Rr22  = Num24/Den23;

fprintf('\nBidarsa - Target Location:\n');
```

```
if TR>=TT
   Rt23 = Rt21;
   Rr23 = Rr21;
   Bet_21 = asin(BR_21/Rt23*cos(TA_21));
   fprintf('Theta_R Selected\n');
else
   Rt23 = Rt22;
   Rr23 = Rr22;
   Bet_21 = asin(BR_21/Rr23*cos(TA_21));
   fprintf('Theta_T Selected\n');
end
fprintf('Distance of Target to Transmitter is %-8.4e km.\n',Rt23);
fprintf('Distance of Target to Receiver    is %-8.4e km.\n',Rr23);

Bet_22 = rad2deg(Bet_21);
fprintf('The Beta Angle calculated is %-0.2f deg.\n\n', Bet_22);
%**************************************************************************


% --- Executes on button press in Print.
function Print_Callback(hObject, eventdata, handles)
printdlg('-setup',handles.figure1)

% --- Executes on button press in Close.
function Close_Callback(hObject, eventdata, handles)
delete(handles.figure1)

% --- Executes on button press in Help.
function Help_Callback(hObject, eventdata, handles)
yHlpTgtLoc
%**************************************************************************
```

E. MATLAB CODE FOR *COVARE.M*

This appendix details the *CovAre.m* code developed to evaluate the coverage areas of both detection-constrained and LOS-constrained cases.

```
function varargout = CovAre(varargin)
% COVARE M-file for CovAre.fig
%      COVARE, by itself, creates a new COVARE or raises the existing
%      singleton*.
%      H = COVARE returns the handle to a new COVARE or the handle to
%      the existing singleton*.
%      COVARE('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in COVARE.M with the given input arguments.
%      COVARE('Property','Value',...) creates a new COVARE or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before CovAre_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to CovAre_OpeningFcn via varargin.
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help CovAre
% Last Modified by GUIDE v2.5 23-Oct-2003 11:49:58
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @CovAre_OpeningFcn, ...
                   'gui_OutputFcn',  @CovAre_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(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 CovAre is made visible.
function CovAre_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
```

```
guidata(hObject, handles);
axes(handles.axes1);
[x1,map] = imread('z-Centerr.jpg','jpg');
image(x1);
colormap(map);
set(handles.axes1,'Visible','off');

axes(handles.axes2);
[x2,map] = imread('z-Centert.jpg','jpg');
image(x2);
colormap(map);
set(handles.axes2,'Visible','off');

axes(handles.axes3);
[x3,map] = imread('z-Centerc.jpg','jpg');
image(x3);
colormap(map);
set(handles.axes3,'Visible','off');

axes(handles.axes4);
[x4,map] = imread('z-LOS.jpg','jpg');
image(x4);
colormap(map);
set(handles.axes4,'Visible','off');


% --- Outputs from this function are returned to the command line.
function varargout = CovAre_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;


% --- Executes during object creation, after setting all properties.
function Htg_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Htg_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Htx_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
```

```
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Htx_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Hrx_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Hrx_input_Callback(hObject, eventdata, handles)


%**************************************************************************
% --- Executes on button press in Calculate.
function Calculate_Callback(hObject, eventdata, handles)
global RtRr11 Br_00
clc;
Htg_input = eval(get(handles.Htg_input,'String'));
Htx_input = eval(get(handles.Htx_input,'String'));
Hrx_input = eval(get(handles.Hrx_input,'String'));

Htg_31 = Htg_input;                          %Height in km
Htx_31 = Htx_input;
Hrx_31 = Hrx_input;
BMRP31 = RtRr11;                             %Bistatic Max Range Parameter in km
BR31   = Br_00;                              %Base Range in km
TEST = 2*sqrt(BMRP31);
TEST2 = abs(BR31-TEST);
TEST3 = TEST2/BR31;


%**************************************************************************
fprintf('\nBidarsa - Coverage Area:\n');
if  (TEST3<=0.1)                             %Check if L~=2sqrt(k)
   Ab31 = pi*BMRP31;                         %Area in km^2
   fprintf('Target is in the Lemniscate Region.\n');
elseif (BR31<TEST)
   Ab31 = pi*BMRP31*(1 - (BR31^4/BMRP31^2/64) - (3*BR31^8/BMRP31^4/16384));
   fprintf('Target is in the Single Cosite Region.\n');
else (BR31>TEST)
```

```
   Ab31 = (2*pi*BMRP31^2/BR31^2)*(1 + (2*BMRP31^2/BR31^4) + ...
     (12*BMRP31^4/BR31^8) + (100*BMRP31^6/BR31^12));
   fprintf('Target is in the Two separate Ellipses Region.\n');
end
fprintf('The Detection-constrained coverage area is %-8.4e km^2.\n\n',Ab31);


%**********************************************************************
Rr31 = 130*(sqrt(Htg_31) + sqrt(Hrx_31));   %Range in km
Rt31 = 130*(sqrt(Htg_31) + sqrt(Htx_31));
Theta_r = 2*acos((Rr31^2 - Rt31^2 + BR31^2)/(2*Rr31*BR31));
Theta_t = 2*acos((Rt31^2 - Rr31^2 + BR31^2)/(2*Rt31*BR31));

if  (Rt31>=(BR31+Rr31))
   Ac31 = pi*Rr31^2;                          %Area in km^2
   fprintf('Tx coverage includes all Rx coverage circle.\n');
elseif  (Rr31>=(BR31+Rt31))
   Ac31 = pi*Rt31^2;
   fprintf('Rx coverage includes all Tx coverage circle.\n');
elseif  ((Rr31+Rt31)>=BR31)                   %Area in km^2
   Ac31 = 0.5*((Rr31^2*(Theta_r - sin(Theta_r))) + ...
            (Rt31^2*(Theta_t - sin(Theta_t))));
   fprintf('Rx and Tx is in LOS.\n');
else
   Ac31 = 0
   fprintf('No LOS between Rx and Tx.\n')
end
fprintf('The LOS-constrained coverage area is %-8.4e km^2.\n\n',Ac31);


%**********************************************************************
close(figure(1));
%Plotting Transmitter-Target-Receiver Height Analysis
BMRP32 = RtRr11*1e3;                          %Bistatic Max Range Parameter in m
BR32   = Br_00*1e3;                           %Base Range in m
for L= [(0.5*BR32) BR32 (1.2*BR32)]
   n=0;
for ht=50:5:1000                              %target heigth in m
   n=n+1;
   hrm=( ((BMRP32+L^2/4)^0.5+L/2)/130-sqrt(ht*1e3) )^2;
   hr=hrm/1000;
   Ht(n)=ht;
   Hr(n)=hr;
end
figure(1)
plot(Ht,Hr,'-'), hold on
```

```
title('Target Height Analysis');
xlabel('Target Altitude (m)')
ylabel('Tx or Rx Antenna Altitude (m)')
grid on
end
axes('position', [0 0 1 1],'visible','off');
text(0.30,0.15,'0.5*L');
text(0.48,0.20,'L');
text(0.68,0.26,'1.2*L');
hold off
%**************************************************************************


% --- Executes on button press in Print.
function Print_Callback(hObject, eventdata, handles)
printdlg('-setup',handles.figure1)

% --- Executes on button press in Close.
function Close_Callback(hObject, eventdata, handles)
delete(handles.figure1)
close(figure(1))

% --- Executes on button press in Help.
function Help_Callback(hObject, eventdata, handles)
yHlpCovAre
%**************************************************************************
```

F. MATLAB CODE FOR *FOTPRT.M*

This appendix details the *ForPrt.m* code developed to evaluate the bistatic footprint and clutter area.

```
function varargout = FotPrt(varargin)
% FOTPRT M-file for FotPrt.fig
%      FOTPRT, by itself, creates a new FOTPRT or raises the existing
%      singleton*.
%      H = FOTPRT returns the handle to a new FOTPRT or the handle to
%      the existing singleton*.
%      FOTPRT('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in FOTPRT.M with the given input arguments.
%      FOTPRT('Property','Value',...) creates a new FOTPRT or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before FotPrt_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to FotPrt_OpeningFcn via varargin.
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help FotPrt
% Last Modified by GUIDE v2.5 20-Oct-2003 17:10:38
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @FotPrt_OpeningFcn, ...
                   'gui_OutputFcn',  @FotPrt_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(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 FotPrt is made visible.
function FotPrt_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
```

```
guidata(hObject, handles);
axes(handles.axes1);
[x,map] = imread('z-BWLimit.jpg','jpg');
image(x);
colormap(map);
set(handles.axes1,'Visible','off');

axes(handles.axes2);
[y,map] = imread('z-PWLimit.jpg','jpg');
image(y);
colormap(map);
set(handles.axes2,'Visible','off');


% --- Outputs from this function are returned to the command line.
function varargout = FotPrt_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;


% --- Executes during object creation, after setting all properties.
function Htx_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Htx_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Hrx_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Hrx_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Rr_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
```

```
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Rr_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Rt_input_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Rt_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Rx_BW_input_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Rx_BW_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Tx_BW_input_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Tx_BW_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function PusWid_input_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function PusWid_input_Callback(hObject, eventdata, handles)


%**********************************************************************
% --- Executes on button press in Calculate.
function Calculate_Callback(hObject, eventdata, handles)
global Ba_00 Br_00
clc;
Rrx_input = eval(get(handles.Rr_input,'String'));
Rtx_input = eval(get(handles.Rt_input,'String'));
RBW_input = eval(get(handles.Rx_BW_input,'String'));
TBW_input = eval(get(handles.Tx_BW_input,'String'));
PuW_input = eval(get(handles.PusWid_input,'String'));

Bet_41 = Ba_00;                                    %Beta Angle in rad
Rrx_41 = Rrx_input;
Rtx_41 = Rtx_input;
RBW_41 = deg2rad(RBW_input);
TBW_41 = deg2rad(TBW_input);
PuW_41 = PuW_input*10^-6;

Num41 = Rrx_41*RBW_41*Rtx_41*TBW_41;
Den41 = sin(Bet_41);

Num42 = 3e8*PuW_41*Rrx_41*RBW_41;
Den42 = 2*(cos(Bet_41/2))^2;

DelR  = (3e8*PuW_41)/(2*cos(Bet_41/2));
RrDr  = Rrx_41*RBW_41;

CeS41 = 3e8*PuW_41/(2*cos(Ba_00/2));

Br41  = Br_00*1e3;
Aa41  = (Rrx_41+Rtx_41)*1e3/2;
Ap41  = Aa41 + (3e8*PuW_41/2);
Bb41  = (Aa41^2 - (Br41^2)/4)^(0.5);
Bp41  = (Ap41^2 - (Br41^2)/4)^(0.5);

Emax41 = ((Aa41*(Ap41-Aa41))/(Bb41*(Bp41-Bb41)))-1;

fprintf('\nBidarsa - Bistatic Footprint:\n');
if DelR < RrDr
   Ac41 = Num42/Den42;
   fprintf('This is a Pulsewidth Limited case.\n');
else
```

```
   Ac41 = Num41/Den41;
   fprintf('This is a Beamwidth Limited case.\n');
end
fprintf('The Bistatic Footprint is %-8.4e sqr-km.\n\n',Ac41);

fprintf('Range Cell Size       is %-8.4e m.\n',CeS41);
fprintf('Maximum Error         is %-8.4e m.\n\n',Emax41);
%*********************************************************************


% --- Executes on button press in Print.
function Print_Callback(hObject, eventdata, handles)
printdlg('-setup',handles.figure1)

% --- Executes on button press in Close.
function Close_Callback(hObject, eventdata, handles)
delete(handles.figure1)

% --- Executes on button press in Help.
function Help_Callback(hObject, eventdata, handles)
yHlpFotPrt
%*********************************************************************
```

G. MATLAB CODE FOR *DOPREL.M*

This appendix details the *DopRel.m* code developed to examine the doppler shift of a moving target.

```
function varargout = DopRel(varargin)
% DOPREL M-file for DopRel.fig
%      DOPREL, by itself, creates a new DOPREL or raises the existing
%      singleton*.
%      H = DOPREL returns the handle to a new DOPREL or the handle to
%      the existing singleton*.
%      DOPREL('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in DOPREL.M with the given input arguments.
%      DOPREL('Property','Value',...) creates a new DOPREL or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before DopRel_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to DopRel_OpeningFcn via varargin.
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help DopRel
% Last Modified by GUIDE v2.5 02-Nov-2003 11:11:15
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @DopRel_OpeningFcn, ...
                   'gui_OutputFcn',  @DopRel_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(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 DopRel is made visible.
function DopRel_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
```

```
guidata(hObject, handles);
axes(handles.axes1);
[x1,map] = imread('z-DopRel.jpg','jpg');
image(x1);
colormap(map);
set(handles.axes1,'Visible','off');


% --- Outputs from this function are returned to the command line.
function varargout = DopRel_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;


% --- Executes during object creation, after setting all properties.
function Vel_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Vel_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Del_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Del_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function TA_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function TA_input_Callback(hObject, eventdata, handles)
```

```
%*********************************************************************
% --- Executes on button press in Calculate.
function Calculate_Callback(hObject, eventdata, handles)
global Ba_00 Lambda_00
clc;
Vel_input = eval(get(handles.Vel_input,'String'));
Del_input = eval(get(handles.Del_input,'String'));

Vel_51 = Vel_input;
Del_51 = deg2rad(Del_input);
DoS_51 = (2*Vel_51/Lambda_00)*(cos(Del_51)*cos(Ba_00/2));
DoS_52 = DoS_51/1000;                                   %in kHz
fprintf('\nBidarsa - Doppler Relation:\n');
fprintf('Bistatic Doppler Shift is %-8.4e kHz.\n\n',DoS_52);


%*********************************************************************
close(figure(1));
%Plotting Doppler Shift Analysis
for Ba_51  = 0:(pi/4):pi;
   Del_52 = -pi:(pi/360):pi;
   DoS_53 = (2*Vel_51/Lambda_00).*cos(Del_52).*cos(Ba_51/2);
   figure(1);
   DoS_54 = DoS_53./1e3;
   Del_53 = 1.*rad2deg(Del_52);
   plot(Del_53,DoS_54);
   title('Bistatic Target Doppler Shift Analysis');
   xlabel('Target velocity aspect angle, \delta, (deg)');
   ylabel('Bistatic Doppler Shift, F_{tgt}(kHz)');
   xlim([-180 180]);
   grid on;
   hold on
end
axes('position', [0 0 1 1],'visible','off');
text(0.87,0.54,'\beta = 180\circ');
text(0.87,0.40,'\beta = 135\circ');
text(0.87,0.25,'\beta = 90\circ');
text(0.87,0.17,'\beta = 45\circ');
text(0.87,0.13,'\beta = 0\circ');
hold off
%*********************************************************************


% --- Executes on button press in Print.
function Print_Callback(hObject, eventdata, handles)
printdlg('-setup',handles.figure1)
```

```
% --- Executes on button press in Close.
function Close_Callback(hObject, eventdata, handles)
delete(handles.figure1)
close(figure(1))

% --- Executes on button press in Help.
function Help_Callback(hObject, eventdata, handles)
yHlpDopRel
%*****************************************************************
```

H. MATLAB CODE FOR *EWEFF.M*

This appendix details the *EWEff.m* code developed to evaluate the bistatic radar performance under different jammer environments.

```
function varargout = EWEff(varargin)
% EWEFF M-file for EWEff.fig
%      EWEFF, by itself, creates a new EWEFF or raises the existing
%      singleton*.
%      H = EWEFF returns the handle to a new EWEFF or the handle to
%      the existing singleton*.
%      EWEFF('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in EWEFF.M with the given input arguments.
%      EWEFF('Property','Value',...) creates a new EWEFF or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before EWEff_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to EWEff_OpeningFcn via varargin.
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help EWEff
% Last Modified by GUIDE v2.5 23-Oct-2003 14:10:38
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @EWEff_OpeningFcn, ...
                   'gui_OutputFcn',  @EWEff_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(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 EWEff is made visible.
function EWEff_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
```

```
guidata(hObject, handles);
axes(handles.axes1);
[x1,map] = imread('z-EWEff.jpg','jpg');
image(x1);
colormap(map);
set(handles.axes1,'Visible','off');


% --- Outputs from this function are returned to the command line.
function varargout = EWEff_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;


% --- Executes during object creation, after setting all properties.
function Pj_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Pj_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Gj_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Gj_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Fj_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Fj_input_Callback(hObject, eventdata, handles)
```

```
% --- Executes during object creation, after setting all properties.
function Bn_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Bn_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Lj_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Lj_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Rj_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Rj_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Bj_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Bj_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Sm_input_CreateFcn(hObject, eventdata, handles)
```

```
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Sm_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Lr_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Lr_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Lt_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Lt_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Fr_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Fr_input_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function Ft_input_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
```

```
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Ft_input_Callback(hObject, eventdata, handles)


%**********************************************************************
% --- Executes on button press in Calculate.
function Calculate_Callback(hObject, eventdata, handles)
global Pt_00 Gt_00 Fq_00 SN_00 Pt_00 Gr_00 Ts_00 Bn_00 Sm_00...
      Ft_00 Fr_00 Lt_00 Lr_00 Lambda_00 RtRr11
clc;
Pj_input = eval(get(handles.Pj_input,'String'));
Gj_input = eval(get(handles.Gj_input,'String'));
Fj_input = eval(get(handles.Fj_input,'String'));
Bj_input = eval(get(handles.Bj_input,'String'));
Rj_input = eval(get(handles.Rj_input,'String'));
Lj_input = eval(get(handles.Lj_input,'String'));

Pj_61 = 10^(Pj_input/10);
Gj_61 = 10^(Gj_input/10);
Fj_61 = 10^(Fj_input/10);
Bj_61 = Bj_input*10^6;
Rj_61 = Rj_input*10^3;
Lj_61 = 10^(Lj_input/10);

Num61 = Pt_00*Gt_00*(Ft_00)^2*(Fr_00)^2*Lj_61*Bj_61*Sm_00*(Rj_61)^2;
Dem61 = Pj_61*Gj_61*(Fj_61)^2*Lt_00*Lr_00*Bn_00*4*pi*SN_00;
RtRrJ1 = (Num61/Dem61)^(0.5);
fprintf('\nBidarsa - EW Effects:\n');
fprintf('Bistatic Max Range Product with Jamming (RtRr)j is  %-8.4e km^2.\n\n',RtRrJ1);


%**********************************************************************
close(figure(1));
%Plotting EW Effects Analysis
for Sm_611 = [(Sm_00/10) Sm_00 (Sm_00*10)]
   Pj_611  = (Pj_61/100):(Pj_61/100):(Pj_61*100);
   Num611  = Pt_00*Gt_00*(Ft_00)^2*(Fr_00)^2*Lj_61*Bj_61.*Sm_611*(Rj_61)^2;
   Dem611  = Gj_61.*Pj_611*(Fj_61)^2*Lt_00*Lr_00*Bn_00*4*pi*SN_00;
   RtRrJ61 = ((Num611./Dem611).^(0.5));
   Pj_612  = 10.*log10(Pj_611);
   figure(1);
   plot(RtRrJ61,Pj_612);
   title('Burn-through  Range Analysis');
```

```
   xlabel('Max Range Product with Jamming (km^2)');
   ylabel('Jammer Transmit Power (dBW)');
   grid on;
   hold on
end
axes('position', [0 0 1 1],'visible','off');
text(0.15,0.15,'0.1\sigma_{B}');
text(0.38,0.20,'\sigma_{B}');
text(0.60,0.25,'10\sigma_{B}');
hold off
%**************************************************************************


% --- Executes on button press in Print.
function Print_Callback(hObject, eventdata, handles)
printdlg('-setup',handles.figure1)

% --- Executes on button press in Close.
function Close_Callback(hObject, eventdata, handles)
delete(handles.figure1)
close(figure(1))

% --- Executes on button press in Help.
function Help_Callback(hObject, eventdata, handles)
yHlpEWEff
%**************************************************************************
```

LIST OF REFERENCES

1. M. I. Skolnik, *Introduction to Radar Systems*, 3rd Edition, McGraw-Hill, New York, 2001.
2. S. Kingsley and S. Quegan, *Understanding Radar Systems*, SciTech, Raleigh, NC, 1992.
3. N. J. Willis, *Bistatic Radar*, 2nd Edition, Technology Service Corporation, Silver Spring, MD, 1995.
4. R. Bennstein and N. J. Willis, unpublished report, Bistatic Radar Project, Naval Postgraduate School, Monterey, CA.
5. *IEEE Standard Radar Definitions*, IEEE Std 686-1997, September 16, New York, 1997.
6. M. I. Skolnik, *Radar Handbook*, 2nd Edition, McGraw-Hill, New York, 1990.
7. D. C. Jenn, unpublished notes, Naval Postgraduate School, Monterey, CA.
8. N. J. Willis, "Bistatic Radars and Their Third Resurgence", *2002 IEEE Radar Conference*, Long Beach, CA, April 2002.
9. URL: http://www.roke.co.uk/sensors/stealth/cell_phone_radar_concept.asp, November 2003.
10. D. C. Schleher, *MTI and Pulse Doppler Radar*, Artech House, Norwood, MA, 1991.
11. H. D. Griffiths, "Bistatic Radar – Principles and Practice", *Microwave Conference*, Brazil, 1993, SMO International, Volume 2, August 2-5, 1993, page 519-526.
12. H. D. Griffith, C. J. Baker, J. Baubert, N. Kitchen, and M. Treagust, "Bistatic Radar Using Satellite-Borne Illuminators", *Radar 2002*, Edinburgh, UK, October 2002, page 1-5.

13. B. R. Mahafza, *Radar Systems Analysis and Design Using Matlab*, Chapman & Hall, London, 2000.

14. P. Marchand and O. T. Holland, *Graphics and GUIs with Matlab*, 3rd Edition, Chapman & Hall, London, 2003.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor John P. Powers, Chairman, Code EC/Po
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

4. Professor David C. Jenn, Code EC/Jn
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

5. Professor D. Curtis Schleher, Code EC/Sc
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

6. Professor Jeffrey Knorr, Code EC/Ko
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

7. Nicholas J. Willis
Carmel, California

8. Professor Yeo Tat Soon
Director, Temasek Defence Systems Institute
Singapore

9. Teo Ching Leong
Singapore Technologies Marine
Singapore