Software Simulation of Synthetic Aperture Radar

by

P. J. Gołda

B.Sc.(Eng), University of Cape Town (1993)

Submitted to the Department of Electrical Engineering
in fulfillment of the requirements for the degree of

Master of Science in Engineering

at the

UNIVERSITY OF CAPE TOWN

September 1997

© University of Cape Town

The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

University of Cape Town

Declaration

I declare that this dissertation is my own work. It is being submitted for the Degree of Master of Science in Engineering at the University of Cape Town. It has not been submitted before for any degree or examination at this or any other university.

Mr Peter John Gołda

Acknowledgements

I would like to thank my supervisor, Associate Professor M.R. Inggs, firstly for proposing research into this exciting field of software engineering, and secondly for his unfailing assistance throughout the period of development.

Special thanks go to Jasper Horrell, of the Radar Remote Sensing Group, for his help with the theory and its implementation, the testing he performed when using my software for his research work, and the constructive criticism that he gave freely. Similarly, thanks go to Gernot Hassenpflug, Richard Lord and Rolf Langefelder who also used the software for their research and provided useful feedback. Big thanks go to Paul Archer of ISIS for spending many hours proof-reading this dissertation.

Further, thanks go to all those posting on the `comp.lang.c` Internet news group for their thorough help during all stages of design and implementation.

Final thanks go to my then girlfriend, and now my wife, Jia, who would not let a weekend go by without pointing out ever so forcefully that my dissertation was yet to be completed.

Terms of Reference

This thesis report is in fulfilment of the requirements for the degree of Master of Science in Electrical Engineering. The thesis was supervised by Associate Professor Michael R. Inggs of the Radar Remote Sensing Group (RRSG), and involved the development of Synthetic Aperture Radar (SAR) simulation software, from now on referred to as **RadSim**. Associate Professor Inggs' specific instructions given in December 1993 were as follows:

- RadSim source code shall comply with the ANSI C standard. This is to allow for execution on different C platforms.
- Each processing module and function within RadSim shall be as independent of the other modules as possible. This is to allow for uncomplicated modification and addition of processing modules and functions later on.
- All parameters for the simulation shall be given in a plain ASCII command file, after which no further user interface shall be specified other than error, information or confirmation messages and their handling.
- Provision must be made for the definition of the simulation geometry which, amongst others, shall allow for the specification of -
  - the aircraft altitude and ground speed;
  - any number of one dimensional point targets whose positions shall be defined in three dimensional space;
  - both sinusoidal and random aircraft motion perturbation data defined using the three dimensions and a squint angle.
- Provision must be made for the definition of the radar parameters and behaviour which, amongst others, shall allow for the specification of -

  - whether monochromatic pulsing or linear FM chirp pulsing is to be used for transmission;
  - standard antenna gain pattern parameters, including the ability to specify the antenna pointing direction.
  - standard mathematical complex array manipulation functions such as Fourier transforms, inverse Fourier transforms, matched filtering (range compression), multiplication, amplification, and analogue to digital conversion.
  - analogue to digital conversion parameters allowing for 4- or 8-bit conversion, as well as sampling rate and sensitivity specification.

- At any point in the processing, it must be possible to write any data array to file, both in a human readable ASCII format and a machine readable binary stream.

- Above all, RadSim must be flexible, fast, simple to use and clearly coded.

Synopsis

The purpose of this report is to set out the results of the development of SAR simulation software. The aim of the thesis was to develop such software so that it provides the necessary functionality but is still flexible and simple to use. It addition it must be developed such that it may be compiled and run on as many platforms as possible and future functionality may be added with ease. All this in order to enable other RRSG members to obtain known simulated SAR data for the purpose of testing SAR processing algorithms.

First, the introductory theory of SAR was studied [7, 11, 12], and once understood, a top level algorithm was designed [8, 13]. The required functionality was specified in more detail and with further study of SAR theory a detailed algorithm design took place [10, 17, 19, 21].

Thereafter, the ANSI C programming standard [2] was studied as well as numerous documentation detailing C programming styles, commenting habits, and software design and implementation [4, 5, 18]. Once completed, RadSim was specified in detail and designed.

The key design decision was to couple the simulator **engine** and the offered functionality loosely. This resulted in the engine being the controlling body of the simulation, responsible for software initialisation, the reading and parsing of the simulation command file and the consequent setting up of the simulation parameters. After this, the engine's responsibility would lie in calling independent functionality modules in the order they were called in the simulation command file and with the appropriate parameters. Each functionality module was then designed to be as independent of other modules as possible and completely separate from the simulation engine.

The implementation of the above design now began and, as with any software project, numerous problems were encountered, especially when it came to the correct implementation of complex functionality such as Fourier transforms, pulse generation and the calculation of

the return pulse. However, debugging was made simple due to the above mentioned loose coupling of the simulation engine and the functionality, which allowed for each simulation function to be tested independently of most other code modules.

After implementation and debugging, thorough and controlled testing began. Although all tests were passed, it was found that due to the representation of analogue waveforms using a finite number of finite precision elements, the resulting waveforms appeared severely quantised in phase and magnitude. This is a problem that cannot be alleviated completely due to the inherent digital computing platform; however a number of steps were taken to improve the results. First, the key mathematical formulas were rearranged so as to minimise the occurrences where very small numbers interact with very large numbers, thus improving result precision. With the same thought, the units of some parameters in the command file were changed so as to scale them down - for example from hertz to gigahertz - such that these small and large number interactions were again minimised. The most important breakthrough was the computation of the return waveform's in-phase and quadrature components **after** the completion of the calculation of the final phase of the pulse, that is the initial phase of the pulse at some point plus the phase change due to range. Before, the initial pulse was separated into its two components and thus its phase was quantised. Onto this quantised phase, a quantised version of the phase change due to range was added. The collective result of these quantisations had disastrous effects. The new method of quantising the phase after its final value had been calculated removed these additional quantisation layers and produced greatly improved results. Finally, this problem's effects are further diminished when the number of sample points that make up all waveforms is increased, as with any system where sampling is employed.

After several months of the use of RadSim by other members of the RRSG and some minor bug fixes and adjustments [6, 8, 14, 16], the software was deemed complete. It was concluded that all requirements had been met and that a working SAR simulator had been produced and that it could be used to simulate both ideal situation and *real life* situations reliably. The software has already been instrumental in a number of postgraduate theses and has helped a number of students to understand SAR.

A number of persons have expressed wishes for additional functionality, such as the speci-

fication of moving targets, of user defined antenna gain patters and aircraft motion offsets, and finally support for stepped frequency pulsing [6, 8, 14, 16]. Thus, it has been recommended that these improvements be made as part of future work. In addition, it may be a worthwhile exercise to convert the software to C++ based on a well planned object model, as now almost all platforms support ANSI C++. Further, a more user friendly interface to RadSim should be useful, either using XMotif for UNIX platforms or ObjectWindows for Windows 95 PC platforms.

Abbreviations and Mathematical Symbols

Abbreviations

A2D	Analogue to Digital Converter
ASCII	American Standard Code for Information Interchange
CFFT	Complex Fast Fourier Transform
FAQ	Frequently Asked Questions
FFT	Fast Fourier Transform
GUI	Graphical User Interface
I	In-phase Waveform Component
PRF	Pulse Repetition Frequency
PRI	Pulse Repetition Interval
Q	Quadrature Waveform Component
RRSG	Radar Remote Sensing Group
SAR	Synthetic Aperture Radar
SGI	Simulation Group Identifier

Mathematical Symbols

α_a	Antenna azimuth angle.
α_{a_a}	Target azimuth angle with respect to aircraft.
α_{a_t}	Target azimuth angle.

α_e	Antenna elevation angle.
α_{e_a}	Target elevation angle with respect to aircraft.
α_{e_t}	Target elevation angle.
α_l	Antenna look angle.
α_m	Total motion offset value in squint angle.
α_{m_a}	Sinusoidal motion offset maximum amplitude in squint angle
α_{m_f}	Sinusoidal motion offset frequency in squint angle.
$\alpha_{m_{max}}$	Random motion offset maximum allowed +/- variation in squint angle.
α_{m_o}	Sinusoidal motion offset "DC offset" in squint angle.
$\alpha_{m_{rnd}}$	Random motion offset value in squint angle.
$\alpha_{m_{sin}}$	Sinusoidal motion offset value in squint angle.
α_s	Aircraft squint angle.
λ_{centre}	Wavelength of operational centre frequency of radar transceiver.
ϕ_{chirp}	Phase angle of linear chirp pulse.
ϕ_{delay}	Phase angle shift associated with t_{delay}.
θ_a	Antenna azimuth beamwidth.
θ_e	Antenna elevation (range) beamwidth.
a_{rcs_t}	Radar cross-section of target.
B_{start}	Bin number in range array for A2D to begin sampling from.
B_{num}	Number of bins for the A2D to process after bin B_{start}.
c	Speed of light constant equal to $3.0 \cdot 10^8\, m/s$.
d	Total distance to be covered by aircraft during simulation.
d_{pri}	Distance covered during each pulse repetition interval.
f_{BW}	Bandwidth of linear chirp pulse.
f_{centre}	Centre frequency of linear chirp pulse.
f_{chirp}	Frequency of linear chirp pulse.
f_{prf}	Pulse repetition frequency.
f_{radar}	Operational centre frequency of radar transceiver.
f_{sim}	Simulation frequency (sampling rate in range direction).
G_a	Amplifier gain factor in decibels.
G_{ant}	Unitless antenna power gain factor.

G_{sin_a}	Unitless sinusoidal antenna voltage gain factor in azimuth angle.
G_{sin_e}	Unitless sinusoidal antenna voltage gain factor in elevation angle.
H	Matched filter waveform.
h	Aircraft ideal altitude above ground level.
i	Imaginary number indicator, $\sqrt{-1}$.
L_{txrx}	Transmission, propagation and reception losses.
N_{bin}	Total number of range bins to be simulated.
N_{bits}	Number of bits of precision of A2D converter.
N_{delay}	Total number of range bins corresponding to t_{delay}.
N_{pri}	Total number of pulse repetition intervals to be simulated.
N_r	Number of complex pairs required to store chirp envelope t_{rise}.
N_{rl}	Number of complex pairs required to store chirp envelope $t_{rise} + t_{length}$.
N_{rlf}	Number of complex pairs required to store chirp envelope $t_{rise} + t_{length} + t_{fall}$.
N_{tar}	Number of point targets to be processed for that SGI.
n_{bin}	Index counter where $0 \leq n_{bin} < N_{bin}$.
n_{bin_f}	Same as n_{bin} but specifically for a frequency domain waveform.
n_{bin_t}	Same as n_{bin} but specifically for a time domain waveform.
P_r	Power in received pulse.
P_t	Power in transmitted pulse.
p_{chirp}	Linear chirp pulse complex waveform.
$p_{chirp_{env}}$	Linear chirp pulse envelope waveform.
p_{chirp_I}	In-phase component of p_{chirp}.
p_{chirp_Q}	Quadrature component of p_{chirp}.
p_{mono}	Monochromatic pulse complex waveform.
p_{mono_I}	In-phase component of p_{mono}.
p_{mono_Q}	Quadrature component of p_{mono}.
p_{return}	Linear chirp waveform p_{chirp} after transmission and return.
R_{max}	Slant range to the end of the swath.
R_{min}	Slant range to the beginning of the swath.
R_o	Slant range offset for simulation geometry.
r_a	Range to target with respect to aircraft.

r_t	Range to target.
r_{total}	Total slant range span covered in the range direction by an array.
S_{sc}	Voltage scaling factor of received waveform relative to the one originally transmitted.
S_{rate}	A2D converter sampling rate in units of bins per sample.
t	Variable of continuous time.
t_{bin}	Duration of a range bin.
t_{delay}	Total time delay between pulse transmission and its return.
t_{fall}	Fall time of linear chirp pulse envelope.
t_{length}	Length of linear chirp pulse envelope at peak amplitude.
t_{mid}	Half of the total linear chirp envelope ($t_{pulse}/2$).
t_o	Time offset corresponding to R_o.
t_{pri}	Duration of a pulse repetition interval.
t_{pulse}	Total length of linear chirp envelope ($t_{rise} + t_{length} + t_{fall}$).
t_{rise}	Rise time of linear chirp pulse envelope.
t_{total}	Total time span covered in the range direction by an array.
t_{width}	Linear chirp envelope width ($t_{rise}/2 + t_{length} + t_{fall}/2$).
V_{lsb}	Value of least significant bit of A2D in volts.
V_{peak}	Peak amplitude of generated linear chirp pulse envelope.
V_r	Received waveform represented as a voltage.
V_{r_I}	In-phase component of V_r in volts.
V_{r_Q}	Quadrature component of V_r in volts.
V_t	Transmitted waveform represented as a voltage.
v	Aircraft ideal velocity along ideal flight path.
W	Hanning window scaling constant.
w	Hanning window function in the time domain.
X_n	Some input array or waveform.
X_{n_I}	In-phase component of X_n.
X_{n_Q}	Quadrature component of X_n.
x_a	x-axis co-ordinate of a point target with respect to aircraft.
$x_{current}$	Current aircraft position along the x-axis during simulation.
x_m	Total motion offset value along x-axis.

x_{m_a}	Sinusoidal motion offset maximum amplitude along x-axis.
x_{m_f}	Sinusoidal motion offset frequency along x-axis.
$x_{m_{max}}$	Random motion offset maximum allowed +/- variation along x-axis.
x_{m_o}	Sinusoidal motion offset "DC offset" along x-axis.
$x_{m_{rnd}}$	Random motion offset value along x-axis.
$x_{m_{sin}}$	Sinusoidal motion offset value along x-axis.
x_{start}	Starting position of the aircraft along the x-axis.
x_t	x-axis co-ordinate of a point target.
Y_n	Some output array or waveform.
Y_{n_I}	In-phase component of Y_n.
Y_{n_Q}	Quadrature component of Y_n.
y_a	y-axis co-ordinate of a point target with respect to aircraft.
y_m	Total motion offset value along y-axis.
y_{m_a}	Sinusoidal motion offset maximum amplitude along y-axis.
y_{m_f}	Sinusoidal motion offset frequency along y-axis.
$y_{m_{max}}$	Random motion offset maximum allowed +/- variation along y-axis.
y_{m_o}	Sinusoidal motion offset "DC offset" along y-axis.
$y_{m_{rnd}}$	Random motion offset value along y-axis.
$y_{m_{sin}}$	Sinusoidal motion offset value along y-axis.
y_t	y-axis co-ordinate of a point target.
z_a	z-axis co-ordinate of a point target with respect to aircraft.
z_m	Total motion offset value along z-axis.
z_{m_a}	Sinusoidal motion offset maximum amplitude along z-axis.
z_{m_f}	Sinusoidal motion offset frequency along z-axis.
$z_{m_{max}}$	Random motion offset maximum allowed +/- variation along z-axis.
z_{m_o}	Sinusoidal motion offset "DC offset" along z-axis.
$z_{m_{rnd}}$	Random motion offset value along z-axis.
$z_{m_{sin}}$	Sinusoidal motion offset value along z-axis.
z_t	z-axis co-ordinate of a point target.

Contents

Declaration i

Acknowledgements iii

Terms of Reference v

Synopsis vii

Abbreviations and Mathematical Symbols xi

Abbreviations xi

Mathematical Symbols xi

Table of Contents xvii

List of Figures xxiii

List of Tables xxv

1 Introduction 1

2 Overview of SAR and Proposal of Simulation Algorithm 5

2.1 Introduction 6

2.2 Typical Hardware Implementation 7

2.3 The Proposed Simulation Algorithm 9

2.4 Simulation Geometry 10

2.5 Pulse Generation 14

2.6 Amplification and Multiplication 17

2.7 Fourier Transformations 18

2.8 Matched Filtering 19

2.9 Calculation of the Return Waveform 20

2.10 Analogue to Digital Conversion 23

3 Software Design and Implementation **25**

3.1 Introduction 25

3.2 Top-Level Design 26

3.3 The Command File 27

3.4 The Simulation Engine 30

3.5 The Individual Simulation Functions 33

3.5.1 Function `GENERAL` 33

3.5.2 Function `TARGET` 34

3.5.3 Function `MOTIONRND` 36

3.5.4 Function `MOTIONSIN` 37

3.5.5 Function `ADD` 39

3.5.6 Function `GEOMETRY` 40

3.5.7 Function PULSEGEN 42

3.5.8 Function MATCHFILT 44

3.5.9 Function FFT 46

3.5.10 Function AMPLIFY 47

3.5.11 Function MULTIPLY 48

3.5.12 Function A2D 48

3.5.13 Function RETURNL 50

3.5.14 Function ENDLOOP 54

3.5.15 Function WRITE 55

3.6 Formats of Output Data Files 56

3.6.1 Arrays of Type 'A' - Amplitude 56

3.6.2 Arrays of Type 'D' - Digitised 57

3.6.3 Arrays of Type 'F' - Frequency 58

3.6.4 Arrays of Type 'G' - Geometry 59

3.6.5 Arrays of Type 'M' - Motion 60

3.6.6 Arrays of Type 'T' - Time 61

4 Simulation Software Testing **63**

4.1 Introduction to the Simulation Function Tests 63

4.2 Test of the GENERAL, MOTIONSIN and WRITE Functions 65

4.3 Test of the MOTIONRND and ADD Functions 69

4.4 Test of the TARGET and GEOMETRY Functions 73

4.5 Test of the PULSEGEN Function 77

4.6 Test of the FFT and MATCHFILT Functions 81

4.7 Test of the AMPLIFY and MULTIPLY Functions 85

4.8 Test of the A2D Function 91

4.9 Test of the RETURNL and ENDLOOP Functions 95

4.10 Portability Tests 101

4.10.1 Borland C++ 3.1 102

4.10.2 Borland C++ 4.02 102

4.10.3 GNU C 102

4.10.4 GNU C++ 103

4.10.5 cc 103

4.10.6 CC 103

5 Sample Simulations **105**

5.1 Introduction to the Simulations 105

5.2 Simple Simulation of One Point Target 107

5.3 Simulation With Noisy Flight Path 115

5.4 Simulation of Three Point Targets 121

5.5 Simulation With Directional Antenna 127

6 Conclusions and Recommendations **129**

6.1 Conclusions 129

6.2 Recommendations 130

A User's Manual and Function Reference **133**

A.1 Introduction 133

A.2 Running RadSim 134

A.3 Function Reference with Examples 135

A.3.1 A2D 135

A.3.2 ADD 136

A.3.3 AMPLIFY 137

A.3.4 ENDLOOP 138

A.3.5 FFT 139

A.3.6 GENERAL 140

A.3.7 GEOMETRY 141

A.3.8 MATCHFILT 142

A.3.9 MOTIONRND 143

A.3.10 MOTIONSIN 144

A.3.11 MULTIPLY 145

A.3.12 PULSEGEN 146

A.3.13 RETURNL 147

A.3.14 TARGET 149

A.3.15 WRITE 150

B Contents of Accompanying Computer Disk **153**

B.1 Contents of `MATHCAD.ZIP` 154

B.2 Contents of `SIM1.ZIP` 154

B.3 Contents of `SIM2.ZIP` 155

B.4 Contents of SIM3.ZIP 155

B.5 Contents of SIM4.ZIP 155

B.6 Contents of SOURCE.ZIP 155

B.7 Contents of TESTS.ZIP 157

B.8 Contents of THESIS.ZIP 157

Bibliography **159**

List of Figures

2.1 Block Diagram of Typical SAR Hardware Implementation 8

2.2 Proposed SAR Simulation Algorithm 9

2.3 Classic SAR Geometry 11

2.4 Cartesian Axes Definition 12

2.5 Simplified Slant Range Geometry 14

3.1 Two Main Parts of the RadSim Design 26

3.2 Flowchart of Function `main()` - the RadSim Engine 31

3.3 Sinusoidal Antenna Gain Pattern 53

4.1 Simulated Sinusoidal Motion Offsets for the x-, y- and z- Axes 67

4.2 Simulated x-Axis Random Motion Offsets 71

4.3 Result of Simulated Summation of Squint Angle Motion Offsets 72

4.4 Simulated Range Curvature for Target 1 75

4.5 Envelope of Simulated Chirp Waveform 80

4.6 In-phase and Quadrature Components of Simulated Chirp Waveform 80

4.7 Overlaid Plots of Radar Chirp Pulse and Ideal Matched Filter 83

4.8 Radar Chirp Pulse After Scaling 88

4.9 Result of Convolution of Radar Chirp Pulse and its Ideal Matched Filter . . . 89

4.10 Digitised Version of Radar Chirp Pulse Waveform 93

4.11 Magnitude of Return Pulse at PRI 7 99

5.1 Sim 1 : Manipulated Range Compressed Image - PRIs 0-3999, Bins 0-127 . . 111

5.2 Sim 1 : Original Range Compressed Image - PRIs 1750-2249, Bins 0-255 . . . 112

5.3 Sim 1 : Manipulated Azimuth Compressed Image - PRIs 1900-2099, Bins 0-127113

5.4 Sim 1 : Original Azimuth Compressed Image - PRIs 1750-2249, Bins 0-255 . 114

5.5 Sim 1 : Cut Through Bin 27 - PRIs 1900-2099 114

5.6 Sim 2 : Manipulated Range Compressed Image - PRIs 0-3999, Bins 0-127 . . 117

5.7 Sim 2 : Original Range Compressed Image - PRIs 1750-2249, Bins 0-255 . . . 117

5.8 Sim 2 : Manipulated Azimuth Compressed Image - PRIs 1900-2099, Bins 0-127118

5.9 Sim 2 : Original Azimuth Compressed Image - PRIs 1750-2249, Bins 0-255 . 119

5.10 Sim 2 : Cut Through Bin 27 - PRIs 1900-2099 120

5.11 Sim 3 : Manipulated Range Compressed Image - PRIs 0-3999, Bins 0-127 . . 124

5.12 Sim 3 : Original Range Compressed Image - PRIs 1750-2249, Bins 0-255 . . . 125

5.13 Sim 3 : Manipulated Azimuth Compressed Image - PRIs 1700-2099, Bins 0-127125

5.14 Sim 3 : Original Azimuth Compressed Image - PRIs 1700-2199, Bins 0-255 . 126

5.15 Sim 4 : Manipulated Range Compressed Image - PRIs 0-3999, Bins 0-127 . . 128

List of Tables

3.1 The Six Simulation Array Types 28

3.2 Parameter Details for the `GENERAL` Function 34

3.3 Parameter Details for the `TARGET` Function 35

3.4 Parameter Details for the `MOTIONRND` Function 36

3.5 Parameter Details for the `MOTIONSIN` Function 38

3.6 Parameter Details for the `ADD` Function 39

3.7 Parameter Details for the `GEOMETRY` Function 40

3.8 Parameter Details for the `PULSEGEN` Function 43

3.9 Parameter Details for the `MATCHFILT` Function 44

3.10 Parameter Details for the `FFT` Function 46

3.11 Parameter Details for the `AMPLIFY` Function 47

3.12 Parameter Details for the `MULTIPLY` Function 48

3.13 Parameter Details for the `A2D` Function 49

3.14 Parameter Details for the `RETURNL` Function 51

3.15 Parameter Details for the `WRITE` Function 55

3.16 Units For 'A'-Type Output Files 57

3.17 Units For 'D'-Type Output Files 58

3.18 Units For 'F'-Type Output Files 59

3.19 Units For 'G'-Type Output Files 60

3.20 Units For 'M'-Type Output Files 61

3.21 Units For 'T'-Type Output Files 62

Chapter 1

Introduction

In recent years, one of the main activities of the RRSG has been to devise, improve and test new algorithms for the processing of raw SAR data. Obtaining suitable raw data for testing purposes has been difficult, and raw SAR data from other commercial and research entities is often limited in volume and represents a single recording geometry. Such data is also composed of the results of a real flight recording session. This is often unwanted as new algorithms and ideas should preferably undergo first tests using ideal situation data. Only once the algorithms are debugged and shown to perform correctly, should the tests move onto the processing of real life data.

For the purpose of obtaining ideal case data, an antiquated VAX-VMS based simulator [3] was used. Although technically it was a very detailed and flexible simulator, it was not suitable for the simulation of SAR. In the case of SAR simulation, the geometry must be simulated for a rather large number of Pulse Repetition Intervals (PRIs) where for each PRI the simulation geometry shifts slightly to reflect the flight of the aircraft. The old simulator was unable to perform this task and so a whole separate simulation had to be run for each PRI. This resulted in many hours of processing time, and also a separate data file for each PRI. These files then had to be downloaded to a PC platform and joined into one data file before further SAR processing could take place.

Thus, the main aim of this research work and this dissertation was to alleviate the above problems and to describe how this was achieved. Note that since most of the RRSG members

of the time were well versed with the old simulator, it was decided to use a similar user interface for the new simulator [3]. Hence the concept of the command file with simulation function calls and parameters.

In summary, this dissertation discusses RadSim, the SAR simulation software package that was developed. This highly configurable and flexible software is able to simulate a number of different aspects of SAR, right from theoretical ideals all the way to the simulation of the effects of perturbations in the aircraft flight path. Further, the software offers a much needed improvement over the VAX-VMS based simulator of the past with regard to SAR simulation.

Chapter 2 of this dissertation provides an overview of the theory and algorithms used in the software package. After introducing the fundamental concepts behind SAR, the SAR simulation geometry is explained together with definitions of numerous concepts and quantities required for the understanding of the task in hand. The algorithm for the simulation is then presented.

Thereafter, Chapter 3 deals with the software specification, design and implementation. After the introduction, each major section and function of the software is specified, the design issues are discussed and the method of implementation is mentioned - this beginning with the simulation engine and going onto dealing with each simulation function in turn. The chapter ends with a description of the formats of output files generated by the software.

Further, Chapter 4 presents the results of detailed and stringent tests of each software component. The results of each simulation function which have been written to file are directly compared to the results of the same mathematical calculations performed using the MathCAD package. The chapter concludes with a discussion of portability levels.

As a final set of tests and as examples, Chapter 5 shows the results of four full SAR simulations. The first example shows the results of a most simple simulation of a single point target in ideal conditions. The second is the same as the first with the exception that aircraft motion errors are introduced. The third introduces two more point targets. The last simulation replaces the omnidirectional antenna with one that has a sinusoidal gain pattern. Each set of simulation results is discussed in detail.

Finally, Chapter 6 draws conclusions and offers recommendations.

The dissertation ends with a two appendices, the first providing a summarised user's manual for the simulation software and the second discussing the contents of the accompanying computer disk.

It was decided that inclusion of a printed version of the source code as a further appendix would not be instructive. However, all of the source code is present on the accompanying PC $3\frac{1}{2}$ inch disk. The disk also contains an electronic version of this document as well as all of the MathCAD sheets generated during testing as described in Chapter 4. The disk also contains all input files, output files and radar images used within the four simulations as described in Chapter 5.

Chapter 2

Overview of SAR and Proposal of Simulation Algorithm

In order to understand the design and operation of the SAR simulation software, a rather detailed knowledge of SAR theory and mathematical properties is required. Since the scope of discussion must be limited, this chapter attempts only to provide a basis of the ideas involved within the SAR simulator.

The introduction below provides an overall view of SAR, with mention of its history and applications. The section after that discusses a typical hardware implementation of a SAR system - it is such a system that RadSim attempts to imitate. Then a stylised block diagram of the proposed simulation algorithm is presented, and each section thereafter deals with some aspect of this proposed algorithm. First, the SAR simulation geometry is described, followed by discussions of pulse generation, amplification and multiplication, Fourier transformations, matched filtering, calculation of the return waveform and finally analogue to digital conversion. For each, the key mathematical relationships are presented.

However, before going any further, it is important to lay some ground rules and state important assumptions that affect this entire research work.

Conventions and Assumptions

All power is assumed to dissipate into 1 ohm loads, this reducing the power - voltage relationship from

$$V = \sqrt{P/R} \quad \text{to} \quad V = \sqrt{P} \quad .$$

Other than this, all components in the system are assumed to have no resistance, impedance or capacitance.

All interconnects between hardware components are assumed to be perfectly matched such that power is transferred without loss from one component to the next[1].

Further, the system is assumed to be noiseless. Therefore, none of the various sources of noise in a typical radar system are taken into account - including thermal noise[2].

Finally, all arrays that represent waveforms are deemed to represent voltage and not power.

2.1 Introduction

SAR began to emerge as an efficient and flexible imaging technique approximately thirty years ago. It is used to image large terrestrial areas from airborne or spaceborne platforms. The main advantages at the time were that radar was not limited to daylight hours and could be operated even with cloud cover. Signal processing was then applied to obtain high resolution images [10].

Recently, the uses of SAR have expanded greatly after it was determined that different sensor frequencies produced imaging that showed different aspects of what was being imaged. For example, high frequencies in the C-Band or above are used to image flora above ground level, while lower frequencies result in ground penetration whereby information may be obtained with regard to mineral deposits. This could also be used by the military to image landmines or other such underground installations. Today SAR is used in a wide variety of applications such as the monitoring of population spread, deforestation, ice cover, oil-slicks

[1]The simulator does however cater for the simulation of a lumped radar pulse transmission and reception loss.

[2]The simulator does however cater for the simulation of erratic aircraft motion.

and fishing. Further, it is used for mining prospecting as mentioned above, the building of topographic maps, and the monitoring of the deformation of the earth's crust after seismic activity [7, 10].

In the past, signal processing was performed using intricate arrays of optical lenses. Luckily, this was soon replaced by digital technology, which opened up an entire new field for signal processing researchers. However, digital processing systems have some disadvantages in that sampling must be performed above the Nyquist rate to avoid aliasing [21]. This implies that high speed data sampling, recording and processing systems are necessary, thus increasing the costs immensely. Therefore, test data for new processing algorithms is expensive to obtain and is limited in volume.

2.2 Typical Hardware Implementation

Figure 2.1 on the following page shows a block diagram of a typical hardware implementation of a SAR system [7]. Since the software simulation algorithm attempts to mimic such an implementation, it is instructive to discuss this block diagram before the proposed algorithm is presented.

Description of the block diagram begins towards the top right side where the coherent oscillator waveform is mixed with a linear chirp[3]. This resulting waveform is then mixed up to transmission frequency with the stable local oscillator. Further, this waveform is amplified and cropped by a rectangular pulse envelope. As may be seen, the making of the chirp pulse and the rectangular pulse envelope is triggered by the radar PRF generator thus synchronising the process. This chirp pulse is then switched by the duplexer into the transmission antenna. The reflected waveform, whose envelope and phase has been distorted by objects on the ground, is then received some time later by the same antenna and switched by the duplexer into a low noise amplifier. The waveform is now mixed back down to baseband and amplified once again. The channel is then mixed down and split into its in-phase and quadrature components. This is done so that after storage, the relative

[3]A linear chirp is a waveform whose frequency changes linearly with time. It is dealt with in more detail later in this chapter.

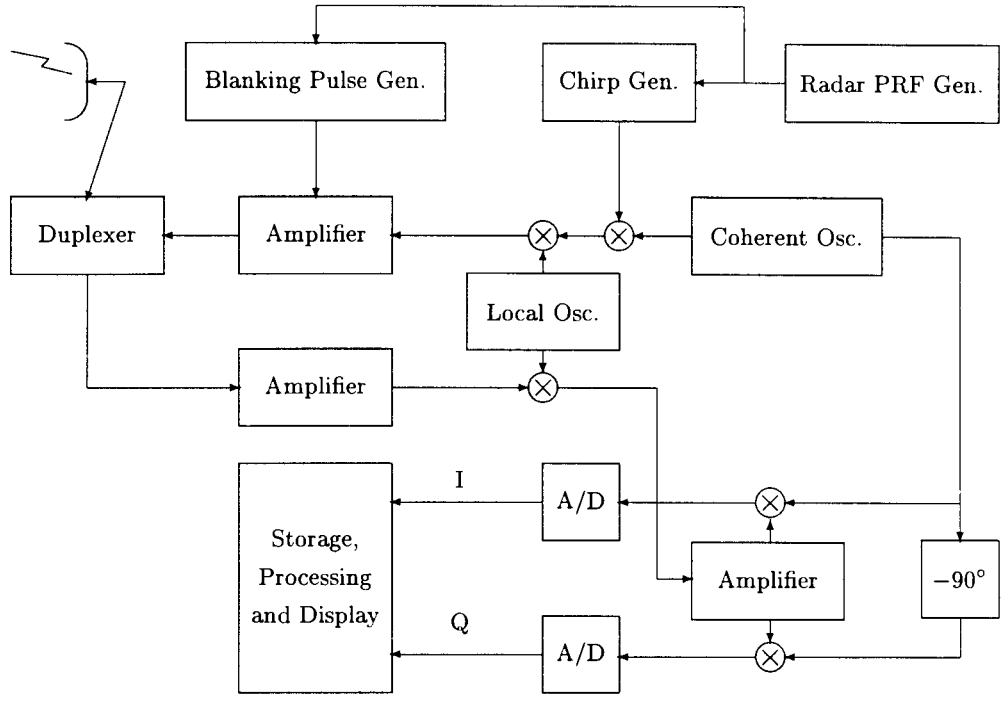

Figure 2.1: Block Diagram of Typical SAR Hardware Implementation

phase of the signal may be determined at any point. Note that in SAR signal processing, it is the phase that carries the most information and must thus be available for later processing. The phase information is also required for azimuth compression. These two I and Q channels are then converted to a digital stream and written to digital mass storage media for further processing.

The block diagram has been simplified and does not show the fact that usually after reception, the waveform is range compressed using a matched filter corresponding to the chirp pulse that was transmitted. Further, the data rates after the A/D converters are often too high for direct storage. To counter this, presumming is employed to bring down the data rate by summing 2^n consecutive received pulses together. The value of n is usually kept between 1 and 8, where the exact value used clearly depends on the speed of available recording equipment.

After storage the data is processed with appropriate software algorithms such as multi-look

azimuth compressors and various digital filters, thus yielding the final radar images.

2.3 The Proposed Simulation Algorithm

Figure 2.2 below shows a block diagram of the proposed simulation algorithm [8, 12, 13].

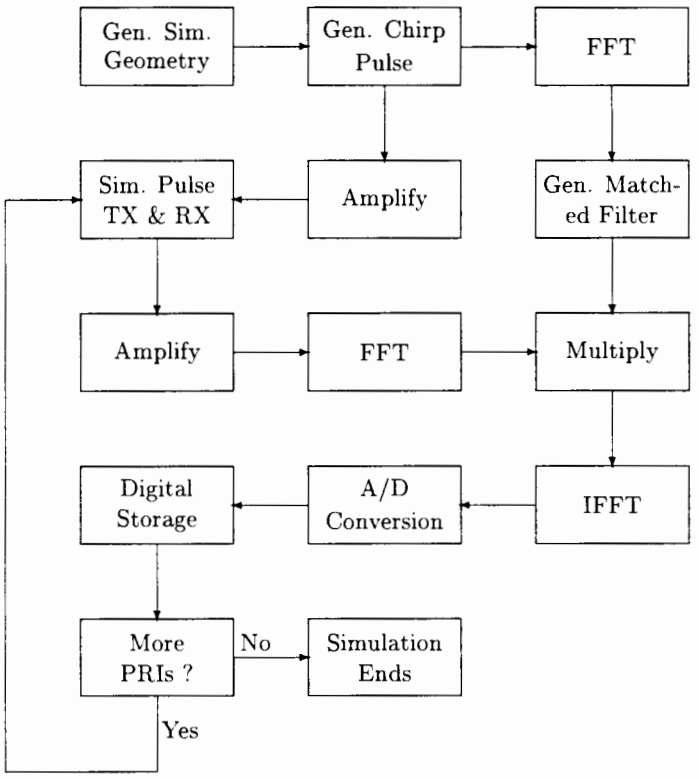

Figure 2.2: Proposed SAR Simulation Algorithm

The main difference between this algorithm and the hardware implementation example of the previous section is that the stable local oscillator and the I and Q channel splitting is not present. The major function of the stable local oscillator, as mentioned before, is to mix the waveforms up to transmission frequency. This is not necessary in the simulation since no physical transmission is taking place and thus all calculations are performed at baseband. With hindsight, this results in great simplification of the software algorithms. With further

hindsight, all of the waveforms must be stored as arrays of complex pairs from the start to prevent loss of phase information during the simulation calculations. Thus there is no need to concern ourselves with the signal to I and Q channel splitting section of the hardware implementation, since the I and Q channels are available already. This having been said, the algorithm may be summarised as follows:

The simulation geometry (positions of the targets, positions of the aircraft, aircraft velocity and flight path) is generated first. Then the chirp pulse is generated, and as may be seen, it is used for two purposes. Along the first branch, it is Fourier transformed into the frequency domain and modified to become the matched filter for itself. Along the other branch, it is amplified. After that, the transmission and return is simulated[4] and the resulting waveform is again amplified. This is then Fourier transformed into the frequency domain and multiplied with the matched filter waveform. The result is then inverse Fourier transformed back into the time domain, digitised and stored. The algorithm then returns to the transmission and return simulation stage as long as there are more pulses to simulate. Note that for each new pulse, the geometry adjusts so as to simulate the aircraft moving forward. When there are no more pulses to be dealt with, the simulation ends.

2.4 Simulation Geometry

In order to understand most of mathematics behind airborne SAR, and the simulation program in particular, it is imperative to understand how the imaging takes place, and the geometry that governs it. To this end, Figure 2.3 on the following page shows the classic SAR geometry set-up [7].

In the figure, the aircraft is travelling at some altitude above ground level, h, and with a constant velocity, v, along the **flight path** vector. Parallel to this flight path vector and along the ground surface is the **nadir track**. The SAR transmitter and receiver is directed to look out at an angle away from normal to the ground as indicated by the **look angle**, α_l. The look angle is related to the elevation angle[5] of the antenna, α_e, in that $\alpha_e = \alpha_l - 90°$.

[4] It is assumed that the distance travelled by the aircraft between transmission of the radar pulse and its subsequent reception is negligible.

[5] A negative elevation angle indicates an angle below the horizontal plane of the aircraft.

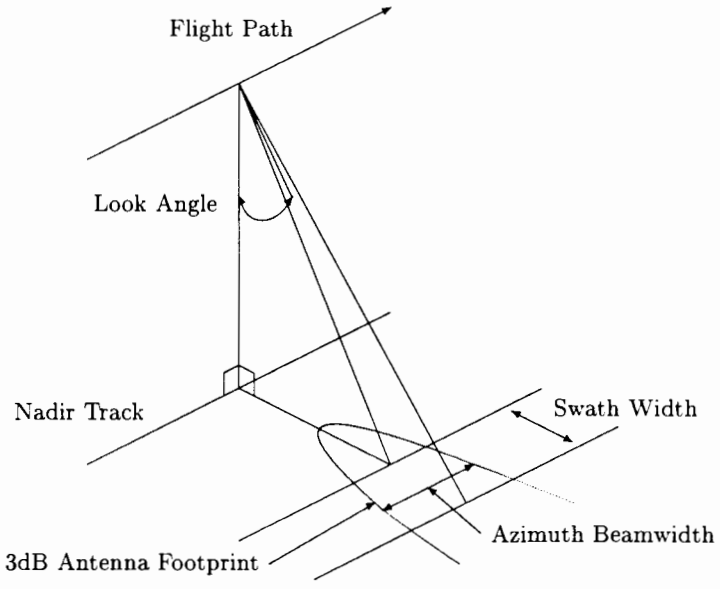

Figure 2.3: Classic SAR Geometry

Not shown in the diagram however is a possible **squint angle**, α_s, which is the angle between the ideal flight path vector and the direction in which the aircraft is pointing[6]. The **azimuth beamwidth**, θ_a, together with the **swath width**, or **range beamwidth**, θ_r, trace out the area of interest. Note that the swath width and azimuth beamwidth are predefined values, and usually do not coincide with the conical section of the **3dB antenna footprint**[7] as the magnitude of the return pulse from farther reaches of the footprint is too weak both due to increased range and increased angles of incidence to the antenna.

Further, as the aircraft flies along, radar pulses are transmitted at regular intervals of time, each interval referred to as the **pulse repetition interval**, or PRI, or t_{pri} and its inverse is the **pulse repetition frequency**, or PRF, or f_{prf}. Clearly, the PRF must be high enough so that successive antenna footprints overlap, thus imaging a continuous swath on the ground[8].

[6]The aircraft may not be pointing in the direction it is flying due to cross-winds. Also, the antenna may not be installed perpendicularly to the aircraft's longitudinal axis. A negative squint indicates a clockwise squint.

[7]That is the area mapped out on the ground by the 3dB azimuth and range beamwidths **of the antenna itself**. In other words, it is the area where the antenna gain response is equal to or more than -3dB. See Figure 3.3 in the next chapter for an example antenna gain response.

[8]There is also a requirement that the PRF be more than twice the maximum Doppler frequency resulting from the movement of the aircraft and/or the whole or part of the ground plane. However, this does not

To clarify matters before going onto the next point in this section dealing with point target definitions, the set of Cartesian axes used is defined in Figure 2.4 below.

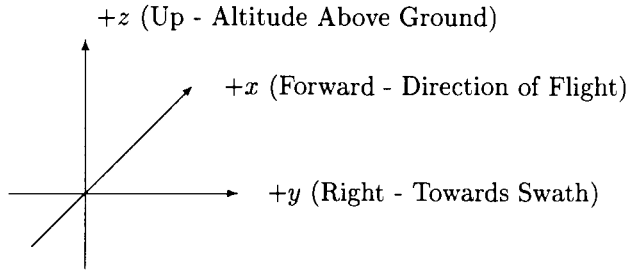

Figure 2.4: Cartesian Axes Definition

The x-axis is always taken to lie along the ideal aircraft flight path, with the forward sense corresponding to $+x$. The y-axis is then taken to be the horizontal axis, with the right sense corresponding to $+y$. The z-axis is then the vertical axis with $+z$ being the upward sense. The origin of this set of axes is placed exactly in the middle of the ideal flight path. Thus, if the total distance to be covered by the aircraft is d, then the aircraft is seen to travel from $-d/2$ to $+d/2$.

Point targets are defined by their three Cartesian co-ordinates in accordance to the definitions of the previous paragraph as x_t, y_t, z_t. In spherical co-ordinates, each point target has a range, r_t, an elevation, α_{e_t}, and an azimuth angle[9], α_{a_t}.

To end this section off, all of the ideas described in this section thus far are formally defined in mathematical terms. The following given constants are defined first:

- f_{prf} - Given pulse repetition frequency.
- N_{pri} - Total number of pulse repetition intervals to be simulated.
- v - Aircraft ideal velocity along ideal flight path.
- h - Aircraft ideal altitude above ground level.

apply here as during the simulation the ground plane is assumed to be stationary and the movement of the aircraft between radar pulse transmission and subsequent reception is assumed to be zero.

[9]The azimuth angle lies in the xy plane, with the $+y$-axis as $0°$, and the $+x$-axis as $90°$.

- α_e - Antenna elevation angle.
- α_s - Aircraft squint angle.
- θ_a - Antenna azimuth beamwidth.
- θ_e - Antenna elevation (range) beamwidth.
- x_t - x-axis co-ordinate of a point target.
- y_t - y-axis co-ordinate of a point target.
- z_t - z-axis co-ordinate of a point target.

Based on the above, the duration of each PRI, t_{pri} is

$$t_{pri} = 1/f_{prf} \tag{2.1}$$

and the distance covered during one PRI, d_{pri} is

$$d_{pri} = vt_{pri}. \tag{2.2}$$

Therefore, the total length of the ideal flight path, d, must be

$$d = (N_{pri} - 1)d_{pri}. \tag{2.3}$$

Finally, the spherical co-ordinates of a target are $(r_t, \alpha_{a_t}, \alpha_{e_t})$ where

$$r_t = \sqrt{x_t^2 + y_t^2 + z_t^2}, \tag{2.4}$$

$$\alpha_{a_t} = \tan^{-1}(x_t/y_t), \tag{2.5}$$

$$\alpha_{e_t} = -\sin^{-1}(z_t/r_t). \tag{2.6}$$

2.5 Pulse Generation

Before discussing pulse generation, it is imperative to introduce some concepts that govern the lengths of arrays used to represent analogue waveforms, and the sampling rate involved. For this purpose, Figure 2.5 below illustrates a simplified slant range geometry.

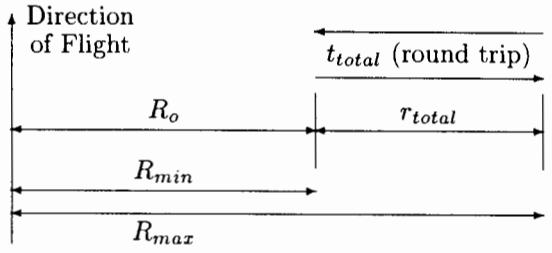

Figure 2.5: Simplified Slant Range Geometry

In the figure, the aircraft is flying towards the top of the page, with the ground below it and the antenna looking out to the right and downwards into the page. R_{min} defines the slant range to the beginning of the swath while R_{max} defines the end of the swath. Further, R_o defines the slant range from the antenna to the start of the swath, and r_{total} defines the slant swath width[10]. The round trip delay that corresponds to the swath width for a waveform that propagates at a speed negligibly close to the speed of light is t_{total} and is defined as follows [11],

$$t_{total} = \frac{2r_{total}}{c} \tag{2.7}$$

where c is the speed of light in a vacuum. This means that the returning waveform must be sampled for a period of t_{total} seconds in order to sample the entire swath width. This sampling must of course occur after a delay corresponding to the range R_o. Supposing now that one wishes to use a sampling rate of f_{sim} hertz when storing this t_{total} seconds worth of analogue waveform in a computer system, then this translates to an array of $N_{bin} = t_{total} f_{sim}$ complex pairs, or so-called **range bins**.

In the context of the needs of the simulator, these relationships are formally defined as follows:

[10]From now on, wherever swath width is mentioned, the slant swath width is implied.

- f_{sim} - Given sampling rate to be used within the simulation for the representation of baseband analogue waveforms in the time or frequency domains, here called the **simulation frequency**.
- N_{bin} - Given total number of complex pairs, or bins, to be used to represent the analogue waveform.

Therefore, the time span covered by each bin is

$$t_{bin} = 1/f_{sim} \tag{2.8}$$

and the total time and range coverage is

$$t_{total} = N_{bin}/f_{sim} \quad \text{and} \tag{2.9}$$

$$r_{total} = c(N_{bin}/f_{sim})/2 \tag{2.10}$$

respectively.

Going back to the original topic, the linear chirp pulse is generated in two stages. First the pulse envelope is generated according to given parameters. From that, the phase information is added by appropriate calculations and splitting of the envelope into in-phase and quadrature components.

First the given parameters are defined:

- t_{rise} - Pulse envelope rise time.
- t_{fall} - Pulse envelope fall time.
- t_{length} - Pulse length where amplitude is at its peak.
- V_{peak} - Peak amplitude.
- f_{BW} - Chirp bandwidth.

From these, the following intermediate parameters are derived:

$$t_{pulse} = t_{rise} + t_{length} + t_{fall} \tag{2.11}$$

$$t_{width} = t_{rise}/2 + t_{length} + t_{fall}/2 \tag{2.12}$$

$$t_{mid} = t_{pulse}/2 \tag{2.13}$$

$$N_r = t_{rise}/t_{bin} \tag{2.14}$$

$$N_{rl} = (t_{rise} + t_{length})/t_{bin} \tag{2.15}$$

$$N_{rlf} = (t_{rise} + t_{length} + t_{fall})/t_{bin} \tag{2.16}$$

Having laid down the mathematical foundation, the pulse envelope, $p_{chirp_{env}}$ is simply defined as

$$p_{chirp_{env}}[n_{bin}] = \begin{cases} V_{peak}(t_{bin}/t_{rise})n_{bin} & \text{if } t_{rise} > 0.0 \text{ and } 0 \leq n_{bin} < N_r \\ V_{peak} & \text{if } N_r \leq n_{bin} < N_{rl} \\ V_{peak}(t_{bin}/t_{fall})(N_{rlf} - n_{bin}) & \text{if } t_{fall} > 0.0 \text{ and } N_{rl} \leq n_{bin} < N_{rlf} \\ 0 & \text{if } N_{rlf} \leq n_{bin} < N_{bin} \end{cases} \tag{2.17}$$

The phase calculations are more complex. The frequency of the chirp pulse at time t is [12]

$$f_{chirp}(t) = f_{centre} + f_{BW}(t/t_{width}) \quad \text{for } -t_{mid} \leq t \leq t_{mid} \tag{2.18}$$

where f_{centre} is the centre frequency of the chirp pulse. Note however that all simulation calculations are performed at baseband, implying that $f_{centre} = 0$. Now the phase angle, $\phi_{chirp}(t)$, corresponding to $f_{chirp}(t)$, is defined as [12]

$$\phi_{chirp}(t) = 2\pi \int f_{chirp}(t)\, dt. \tag{2.19}$$

Therefore,

$$\phi_{chirp}(t) = \pi f_{BW}(t^2/t_{width}) \quad \text{for } -t_{mid} \leq t \leq t_{mid} \tag{2.20}$$

remembering that f_{centre} has been set to zero and ignored.

What is required now is a conversion from the continuous time variable to the discrete n_{bin} counter. The important issue to note that in Equation 2.20 the time variable runs from $-t_{mid}$ to t_{mid} while the index n_{bin} runs from 0 to $N_{bin} - 1$. This gives rise to an offset subtracted from n_{bin} and results in the following:

$$\phi_{chirp}[n_{bin}] = \pi f_{BW}((n_{bin}t_{bin} - t_{mid})^2/t_{width}) \quad \text{for } 0 \leq n_{bin} < N_{bin} \tag{2.21}$$

Now that one knows the amplitude of the pulse, $p_{chirp_{env}}$ and the phase angle ϕ_{chirp}, the complete chirp waveform is constructed as

$$p_{chirp}[n_{bin}] = p_{chirp_{env}}[n_{bin}]e^{j\phi_{chirp}[n_{bin}]}. \tag{2.22}$$

Expanding this to a form more convenient for computer calculation using the identity [17]

$$\cos(\omega t) + j\sin(\omega t) = e^{j\omega t} \tag{2.23}$$

one obtains

$$\begin{aligned} p_{chirp_I}[n_{bin}] &= p_{chirp_{env}}[n_{bin}]\cos(\phi_{chirp}[n_{bin}]) \\ p_{chirp_Q}[n_{bin}] &= p_{chirp_{env}}[n_{bin}]\sin(\phi_{chirp}[n_{bin}]) \end{aligned} \tag{2.24}$$

which gives the in-phase and quadrature channels of the chirp pulse.

2.6 Amplification and Multiplication

At certain stages in the algorithm, it is required that an array representing an analogue waveform, be it in the time or frequency domain, is amplified by a constant gain, or multiplied by another array of the same type. Amplification occurs before transmission and after reception. Multiplication of two waveforms in the frequency domain is used, instead of convolution in the time domain, to filter one waveform by the other.

Amplification is very simple. Assuming the input waveform, in volts, is stored within the simulator as some array X_n, which consists of N_{bin} complex pairs, then the amplified result

array, Y_n, also in volts, is

$$Y_n[n_{bin}] = 10^{G_a/20} X_n[n_{bin}] \tag{2.25}$$

where G_a is the gain factor given in decibels, and n_{bin} is the complex array index.

Multiplication, on the other hand, requires more thought as one is dealing with complex arrays. Once again, assuming that the first waveform to be multiplied, in volts, is stored within the simulator as some array X_{n1}, which consists of N_{bin} complex pairs, and the second array to be multiplied is similarly stored as X_{n2}, then the product array, Y_n, also defined to be in volts, is

$$\begin{aligned} Y_{n_I}[n_{bin}] &= X_{n1_I}[n_{bin}]X_{n2_I}[n_{bin}] - X_{n1_Q}[n_{bin}]X_{n2_Q}[n_{bin}] \\ Y_{n_Q}[n_{bin}] &= X_{n1_I}[n_{bin}]X_{n2_Q}[n_{bin}] + X_{n1_Q}[n_{bin}]X_{n2_I}[n_{bin}] \end{aligned} \tag{2.26}$$

where the subscripts I and Q are used to indicate the in-phase and quadrature component of the complex pair respectively.

2.7 Fourier Transformations

Fourier transformations are used to transform an analogue waveform, stored in an array of complex pairs, either from the time domain to the frequency domain, or the other way round. The need for this arises from the need to filter the returned waveform with another filter waveform. In the time domain this requires convolution, which is more difficult to implement and more processor hungry than the equivalent Fourier transform, multiplication, and inverse Fourier transform [17, 19].

Thus, given an input waveform, in volts, in the time domain, which is stored within the simulator as some array X_n, which consists of N_{bin} complex pairs, then the transformed output array, Y_n, also in volts but in the frequency domain is [19]

$$Y_n[n_{bin_f}] = \sum_{n_{bin_t}=0}^{N_{bin}-1} X_n[n_{bin_t}] e^{-j(\pi/N_{bin}) n_{bin_t} n_{bin_f}} \tag{2.27}$$

where the subscripts t and f on n_{bin} are used to indicate that the two indexes are independent, one indexing the time domain complex array, the other indexing the frequency domain

complex array.

Similarly, an inverse transformation where X_n represents a waveform in the frequency domain, and Y_n is the output array in the time domain is [19]

$$Y_n[n_{bin_t}] = \frac{1}{N_{bin}} \sum_{n_{bin_f}=0}^{N_{bin}-1} X_n[n_{bin_f}] e^{+j(\pi/N_{bin}) n_{bin_t} n_{bin_f}} \tag{2.28}$$

Note that the scaling factor $1/N_{bin}$ ensures amplitude equality between an original waveform, and the inverse transform of its transform [17].

2.8 Matched Filtering

There is a requirement that a filter be derived that matches the waveform that is transmitted by the radar. The transmitted waveform is an array of complex values. By definition, the ideal matched filter for a complex time domain waveform is a complex conjugate of that waveform reversed in time [21]. Thus, if the input waveform, X_n, in volts, is N_{bin} complex pairs long, then the ideal matched filter, H is [21]

$$H[n_{bin_t}] = X_n^*[-n_{bin_t}] \tag{2.29}$$

where the asterisk indicates the complex conjugate of X_n.

However, noting that the matched filter waveform is required in the frequency domain, as discussed in the previous section, so that it can be multiplied with the transform of the simulated return waveform so as to perform range compression, one makes use of the identity [21]

$$f^*(t) = F^*(-\omega) \tag{2.30}$$

such that

$$H[n_{bin_f}] = X_n^*[n_{bin_f}] \tag{2.31}$$

where X_n has been transformed into the frequency domain and the resulting matched filter is also in the frequency domain.

This simplification reduces the number of computations required to calculate the matched filter in the frequency domain, thus clearly reducing processing time.

2.9 Calculation of the Return Waveform

This is the most mathematically complex of all the sections. Given the chirp pulse, together with the simulation geometry for that particular PRI, the transmission of the pulse, its interaction with defined point targets, and then its return must be simulated. There are three items that must be calculated - first the time delay between the transmission of the pulse and its consequent reception, second the amplitude of the returned pulse, and thirdly the relative phase of the returned pulse. These three issues are dealt with in order on the following page.

The time delay from the moment that the pulse was transmitted to the moment it is received again is

$$t_{delay} = 2r_t/c \tag{2.32}$$

Note the factor of 2 because t_{delay} represents the time to the target and back[11]. Recall that r_t is the range to the target in question. Finally, the number of range bins that t_{delay} corresponds to is

$$N_{delay} = \text{round}(t_{delay}/t_{bin}) \tag{2.33}$$

Now for the amplitude of the returned pulse. Recalling the assumptions made at the beginning of this chapter, the amplitude of the returned pulse is affected only by a range dependent system scaling factor that also accounts for lumped losses and gains. Thus, the relationship between the transmitted waveform and the received one is

$$V_r = V_t S_{sc} \tag{2.34}$$

where V_t is the transmitted waveform, V_r is the received waveform and S_{sc} is the already introduced voltage system scaling factor. A meaningful formula for S_{sc} is derived from the

[11] Recall that it is assumed that the distance travelled by the aircraft during this time is negligible.

standard radar equation [10, 11, 21]

$$P_r = P_t \frac{G_{ant}^2 \lambda_{centre}^2 a_{rcs_t}}{(4\pi)^3 r_t^4 L_{txrx}} \tag{2.35}$$

where P_t is the power in the transmitted waveform, P_r is the power in the received waveform, G_{ant} is the antenna gain (squared since the same antenna is used for transmission and reception), a_{rcs_t} is the target radar cross-section, L_{txrx} are the combined transmission, propagation (two way) and reception losses and any other losses that one may wish to include, r_t is the range to the target, and finally λ_{centre} is the wavelength corresponding to the radar operational centre frequency. Until now, the radar centre frequency has been kept at zero for all intents and purposes and all relevant calculations have been performed at baseband. This is the only time that the centre frequency is required as it affects the gain of the system. Note that

$$\lambda_{centre} = c/f_{centre} \tag{2.36}$$

Since 1 ohm loads are assumed throughout, Equation 2.35 may be rewritten in terms of V_r and V_t as follows

$$\frac{V_r^2}{V_t^2} = \frac{G_{ant}^2 \lambda_{centre}^2 a_{rcs_t}}{(4\pi)^3 r_t^4 L_{txrx}} \tag{2.37}$$

Taking the square root of the equation gives the desired value for the voltage system scaling factor which is now formally defined as

$$S_{sc} = \sqrt{\frac{G_{ant}^2 c^2 a_{rcs_t}}{(4\pi)^3 r_t^4 f_{centre}^2 L_{txrx}}} \tag{2.38}$$

Therefore, the amplitude of the returned pulse is defined to be its original amplitude multiplied by S_{sc}. Splitting the received waveform into I and Q channels effectively also splits the power evenly between the two channels and thus does not affect the derived value of S_{sc}. This fact is derived as follows: The total power contained within the I and Q channels must equal the received power such that

$$V_{r_I}^2 + V_{r_Q}^2 = P_r = S_{sc}^2 P_t \tag{2.39}$$

Since the power is split evenly between the channels

$$\begin{aligned} V_{r_I}^2 &= P_r/2 = S_{sc}^2 P_t/2 \\ V_{r_Q}^2 &= P_r/2 = S_{sc}^2 P_t/2 \end{aligned} \tag{2.40}$$

and thus

$$\begin{aligned} |V_{r_I}| &= \sqrt{P_r/2} = S_{sc}\sqrt{P_t/2} \\ |V_{r_Q}| &= \sqrt{P_r/2} = S_{sc}\sqrt{P_t/2} \end{aligned} \tag{2.41}$$

which is the desired result as S_{sc} is still the scaling factor that is applied to both channels.

Finally the phase shift, ϕ_{delay}, is defined as [11]

$$\phi_{delay} = 4\pi r_t/\lambda_{centre} \tag{2.42}$$

This is converted to

$$\phi_{delay} = 2\pi(t_{delay} f_{centre}) \tag{2.43}$$

which is more suitable since one assumes that f_{centre} is given. Note also that for computational purposes to ensure that overflows do not occur, the factor inside the brackets, that is $t_{delay} f_{centre}$ would always be kept modulo 1 as the factor of 2π in front ensures cyclic behaviour.

Bringing all of the above together, if the transmitted waveform is p_{chirp}, then the return p_{return} is

$$p_{return}[n_{bin}] = p_{chirp}[n_{bin} - N_{delay}] e^{j\phi_{chirp}[n_{bin}-N_{delay}]} \cdot S_{sc} e^{-j\phi_{delay}}. \tag{2.44}$$

which is simplified to

$$p_{return}[n_{bin}] = S_{sc} p_{chirp}[n_{bin} - N_{delay}] e^{j(\phi_{chirp}[n_{bin}-N_{delay}]-\phi_{delay})}. \tag{2.45}$$

The above is split into its in-phase and quadrature components in the same way as p_{chirp} was in Section 2.5 using the identity shown in Equation 2.23.

The extension of the above for a case where there is more then one target is trivial as the

effects are purely additive as in a real radar system [8, 12]. Equation 2.45 is processed for each target, giving one p_{return} array for each target. All of these arrays are then added to yield the final return waveform.

2.10 Analogue to Digital Conversion

In a real SAR system, the in-phase and quadrature parts of the return waveform are converted to a digital stream for storage on mass media so that post-processing may be performed. As this is an important stage of a typical SAR system, it must be simulated by RadSim.

Supposing that the analogue return waveform, in volts, is stored within the simulator as some array X_n, which consists of N_{bin} complex pairs. The digitised integer array Y_n is then [8]

$$Y_n[n_{bin}] = \begin{cases} \text{floor}(X_n[n_{bin}]/V_{lsb} + 0.5) & \text{for } X_n[n_{bin}] \geq 0 \\ \text{ceil}(X_n[n_{bin}]/V_{lsb} - 0.5) & \text{for } X_n[n_{bin}] < 0 \end{cases} \tag{2.46}$$

where V_{lsb} is the given constant value of the least significant bit of the A2D converter, also in volts, and n_{bin} is the complex array index. As generally understood, the floor() function returns the largest integer smaller than or equal to its argument, and the ceil() function returns the smallest integer larger than or equal to its argument. The above equation is not quite complete, as a real A2D converter does not represent negative values and the number of bits using which the result is represented is limited. The appropriate modifications to the equation, together with a detailed explanation are given in Chapter 3 in the subsection dealing with the A2D simulation function.

Chapter 3

Software Design and Implementation

In this chapter the RadSim design is presented in detail and, where applicable, information as to implementation methods is offered and relevant mathematical formulae are given and explained.

The introduction below offers a very brief overview of the system, and reasons for the specific choice of platform and language. In the next section, the design is dealt with in more detail. This is followed by the definition of the so-called **command file**. After that, the simulation **engine**, which is responsible for parsing the command file and managing the simulation, is dealt with in detail. Each function offered by RadSim is then discussed in turn. The chapter ends with a description of all output data file formats.

3.1 Introduction

The idea for this project came from the results of an undergraduate thesis by Liz Keay [12]. Miss Keay's software was able to simulate SAR under very specific conditions and with limited options. Her thesis was sufficient to illustrate the principles involved, and thus lay the foundations for and inspired the need for a more robust, flexible and advanced simulator.

At first, it was thought that Miss Keay's software could be used as the basis for the new simulator. After some time spent developing however, it was found that her design methods were lacking in solidity and did not provide the room needed for the expansion and improvement of the system. It was thus decided to begin the RadSim software project from scratch, but at the same time keeping in mind the valuable lessons learned from Miss Keay's simulator [13].

It was further decided to retain the **C** language as the implementation language for RadSim. The C language, at the time that the decision was made, was the most widely used development language world-wide. It was also the most portable, and a great deal of time was spent studying the ANSI C standard [2, 4, 5, 18] so as to ensure a maximum portability of RadSim across all ANSI C platforms. The C language is also fast and simple, not requiring the overheads of 4GL data processing systems such as Khoros or IDL. Such 4GL systems are also often proprietary and specific to one platform and thus make portability almost impossible. The C language was also the common denominator within the RRSG, as few people had migrated to Object Orientation and/or C++ at the time. This allowed other members of the group to study the software, understand it and use it better for their various purposes [8].

3.2 Top-Level Design

The design is very clearly separated into two parts. The first is the simulation engine, which parses the command file, sets up the simulation and controls it. The second part consists of all the separate simulation functions. This is illustrated in Figure 3.1 below.

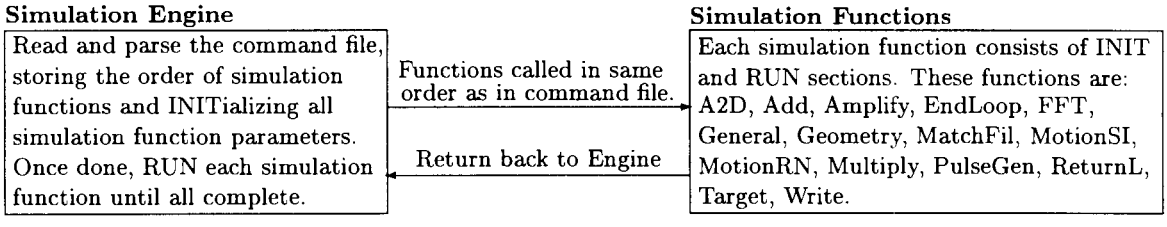

Figure 3.1: Two Main Parts of the RadSim Design

In the first iteration of the design process, the various functions were called immediately as

they were parsed from the command file. The impracticality of this was soon found, for if the command file contained an error right at the end, the user would have to wait through almost the entire simulation until the error was detected and RadSim aborted. Thus, this method, which can be likened to an interpreter, was replaced by the two part design. Here, the simulation engine was used to first parse the command file in its entirety, and set up the simulation. Only then, if the command file was found to be error free, would the relevant functions be called by the engine. This may be likened more to a compilation and execution system.

This also resulted in a significant error handling simplification because almost all error handling could now be performed by the parsing section of the engine, leaving the simulation functions cleaner and leaner. This saw a corresponding increase in performance, as most of the processing time is clearly spent within these various simulation functions.

This separation of the engine and the functional part of the simulation also meant that simulation functions were now technically completely independent of each other. They could thus be made more generic and flexible and designed with clearly specified inputs and outputs. This may be likened to the C++ idea of encapsulation. Later sections of this chapter should illustrate this clearly.

Before going on to a more detailed discussion of the RadSim engine, the command file must be dealt with so that one has an idea of the basic requirements of the parsing subsystem and the simulation set-up that must be performed.

3.3 The Command File

The command file is a plain ASCII file, that may be likened to a program. As the name implies, the file consists of a number of commands. Each command takes the following general form:

```
! A comment about this command
$COMMAND_NAME OUT_PARM_1      ! A comment about this variable
     IN_PARM_1                ! A comment about this parameter
     IN_PARM_n                ! A comment about this parameter
```

The command name and all the parameters themselves may be separated by white space, newlines, tabs, commas, any combination of these four, and comments. As may be seen, the output variable is listed first, while the input parameters are last. Usually, there is only one output variable, and commands are written in that visual format to ensure good readability. Clearly, the exclamation mark is the comment field delimiter and comments are always taken to run to the end of the line. Further, the dollar sign character, '$', always precedes a command name to distinguish it from any other text that may follow in the inputs section. The command file parser is case insensitive.

An important issue that must be dealt with here is that of array types. RadSim must be able to store and manage arrays that represent different intermediate results. There are arrays that store waveform data in time or frequency domains using floating point complex number pairs, or digitised integer complex pairs, or magnitude information only using real floating point elements. There are arrays that store the simulation geometry where for each target its three dimensional position is given for each simulation iteration. Finally, there are arrays that store aircraft motion perturbation offset data which is used to simulate real conditions of flight.

There are six array types in total, each one specified using a single letter code. Table 3.1 below details these six types.

Table 3.1: The Six Simulation Array Types

Type	**Meaning and Use**
'A'	Amplitude - one column array with waveform magnitude information only stored as integer values.
'D'	Digitised - two column array with waveform complex pairs which have been digitised to pairs of integer values.
'F'	Frequency - two column array with waveform complex pairs in the frequency domain stored as pairs of floating point values.
'G'	Geometry - three column array with simulation geometry data as floating point values. For each target for each PRI there is the range to target, target azimuth angle and target elevation angle.
'M'	Motion - four column array with aircraft motion perturbation data as floating point values. For each PRI the aircraft offset in Cartesian co-ordinates and squint angle is given.
'T'	Time - two column array with waveform complex pairs in the time domain stored as pairs of floating point values.

Clearly, more than one unique array of each type may be required during the simulation. Thus, each array type code is followed by an identifier, which is an integer value between 1 and 256.

Finally, each simulation command file must begin with a call to the **GENERAL** function which sets up global parameters for that simulation. Since there may be more than two of these calls in one command file, each call must be given a **Simulation Group Identifier**, or SGI, which is a single letter from 'A' to 'Z'.

The following is an example of an abridged command file which should illustrate the use of array type codes and their identifiers as well as the concept of the SGI.

```
! Abridged demo command file

$GENERAL  A      ! General simulation data with SGI=A
     64          ! [ ]   Number of Pulse Repetition Intervals to Simulate
     50.00       ! [Hz]  Pulse Repetition Frequency
    256          ! [ ]   Number of range bins to simulate
      2.00       ! [GHz] Simulation frequency

$GEOMETRY G1     ! Calculate simulation geometry data and put into array G1
      A          ! Use which GENERAL data ?
     M1          ! Use which MOTION data ?
   3000.00       ! [m]   Aircraft altitude
    100.00       ! [m/s] Aircraft velocity

$GEOMETRY G2     ! Calculate simulation geometry data and put into array G2
      A          ! Use which GENERAL data ?
     M2          ! Use which MOTION data ?
   4000.00       ! [m]   Aircraft altitude
    150.00       ! [m/s] Aircraft velocity

$WRITE G1 ASC G1.ASC     ! Print G1 to file G1.ASC in ASCII format
$WRITE G2 BIN G2.ASC     ! Print G2 to file G2.ASC in raw binary format
```

3.4 The Simulation Engine

This section discusses the RadSim simulation engine which is the central controlling module of the system. The engine is responsible for the parsing of the command file, the setting up of parameter structures, the allocation of necessary memory and the execution of the simulation via calls to the appropriate simulation functions. Great emphasis was placed on the detection of all possible errors **before** the simulation proper actually begins. As mentioned previously, the simulation may be a very time consuming process and it is irritating for the user to have to re-run the simulation when an error occurs towards the end of the run. However, the main emphasis lay in the design of this module such that the addition and modification of the separate simulation function calls may be as simple as possible - this is achieved by ensuring that the simulation functions are independent with well defined parameters and behaviour, and that the engine is as generic as possible in the way it parses the command file and then calls the simulation functions.

For each function call found in the command file, the relevant parameters, whether given in the command file or derived internally are stored in linked lists[1], where one list is dedicated to one type of simulation function. The use of lists is clear because the same simulation function, with the same or different parameters, may be called more than once during the simulation and so each call must be separately stored uniquely in order. Linked lists provide a dynamic manner of achieving this[2]. The simulation engine thus keeps a pointer to each list[3], and inserts a new entry into the appropriate list with the appropriate parameters for each command found in the command file. There is also a main list, which simply keeps track of the order in which the various commands were set out in the command file, and which item in which linked list containing simulation function parameters belongs to which call.

Warning, error and information reporting is handled in a central messaging system[4]. If an

[1] Refer to the `global.h` source code file where all of the parameters for each simulation function are defined within structures.

[2] Refer to the `linklist.h`, `linklist.c` and `macros.h` source code files for detail. Note how use is made of the `#define` macros to simulate C++ templates so that only one set of linked list handling functions may be used for all the different data structures to be stored in the these lists [5], i. e. one `typedef`'ed structure per simulation function as per `global.h`.

[3] Refer to `engine.h`, `engine.c`, `common.h` and `common.c` for details of memory management.

[4] Refer to `message.h` and `message.c` source code files.

error state is detected, or a warning or information message needs to be communicated to the user, the messaging system is invoked with a unique message identification number and, if necessary, other situational parameters, like a line number, used to make up a meaningful message. The messaging system constructs the message and displays it to the user, very much like a compiler.

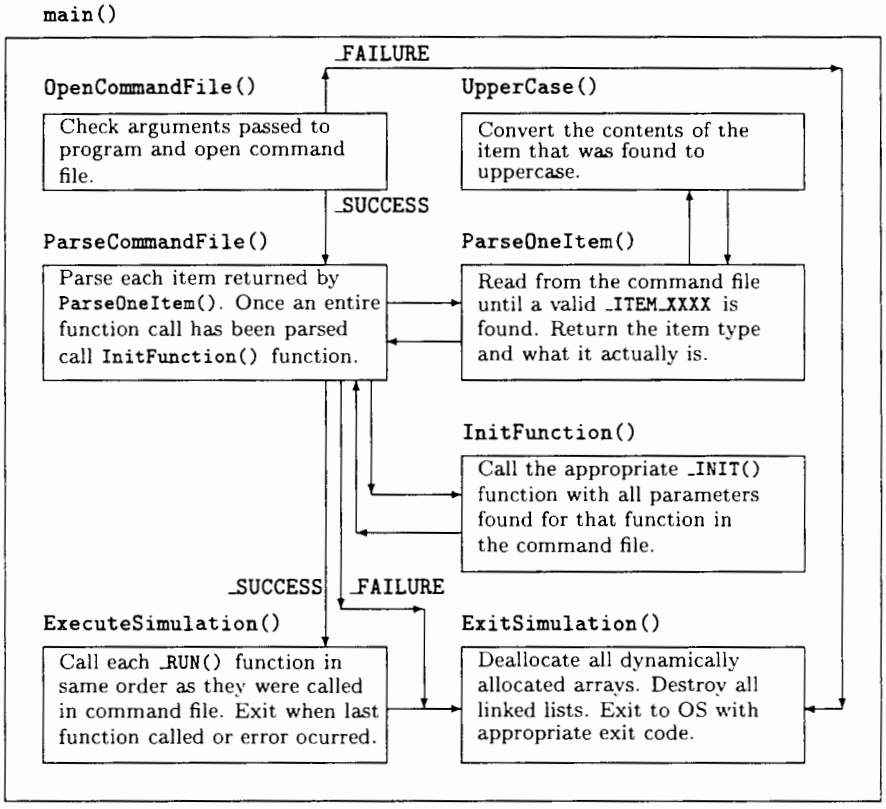

Figure 3.2: Flowchart of Function `main()` – the RadSim Engine

Figure 3.2 above shows a simplified flowchart of the operation of the RadSim engine. Note that each simulation function consists of two C functions called `_INIT()` and `_RUN()`[5]. The `_INIT()` function always takes all parameters found for that simulation function in the command file, derives any other necessary parameters and adds the lot onto the end of the corresponding linked list, and adds the actual simulation call to the main simulation call list. The corresponding `_RUN()` function is then called later during the simulation proper at the

[5]These are macro names that are expanded by the parser to unique names for each function during compilation.

correct point. It must be noted that most stringent error and parameter bounds checking is performed within the _INIT() functions. This results in almost all possible errors being caught during the command file parsing stage, thus allowing the _RUN() functions to assume a valid set of parameters and to concentrate on processing the data.

All in all, the simulation engine consists of four functions, not including main(), that are worth mentioning here. Each of these functions is now briefly described :-

Function ParseOneItem() : This function runs through the command file, ignoring white space, tabs, commas and comments. When End-Of-Line, or End-Of-File is detected, the function returns to ParseCommandFile(). When a simulation function call is found, its name is parsed and returned. When a simulation function parameter is found, its name and value is parsed and returned. Full error handling is in place and the parsing is case insensitive.

Function ParseCommandFile() : This function successively calls ParseOneItem() until End-Of-File is found. For each End-Of-Line, the line number counter (used for error reporting) is incremented. For each new simulation function call, InitFunction() for the previous simulation function is called with all found parameters, after which the parameter counter is reset. Also, the new simulation function call is added to the main call list so as to allow the engine to keep track of which simulation functions were called and when. For each new parameter, the parameter in question is added to the parameter list and the parameter counter is incremented.

Function InitFunction() : Called by ParseCommandFile(), this function determines which simulation function is concerned and thus calls the correct _INIT() function passing along the list of found parameters.

Function ExecuteSimulation() : Once the command file has been parsed in full, control is passed to this function which iterates through the main call list. The correct simulation _RUN() function is thus called in the correct order, together with pointers to required parameter items residing in relevant linked lists.

3.5 The Individual Simulation Functions

Having discussed the format of the command file, and the workings of the RadSim engine, the individual simulation functions can now be presented. Each function is dealt with separately, and in detail. The sections all begin with the command file format of the function in question, a description of the parameters and their allowable values, a description of its purpose, and notes about implementation with mathematical formulae where applicable. Note that the order of presentation is not in alphabetical, but rather in logical order. Before beginning however, some notation conventions must be clarified:

- Floating point values are shown by `f.fff`.
- Integer values are shown by `iii`.
- Text fields are shown by `ttt`.
- Input arrays are shown by `Xn`.
- Output arrays are shown by `Yn`.
- The simulation group identifier is shown by `A`.

3.5.1 Function GENERAL

The command file format for this function is shown below. This is followed by Table 3.2 which formally describes each parameter.

```
$GENERAL A
       iii    ! [ ]   Number of PRIs to simulate
    ff.fff    ! [Hz]  Pulse Repetition Frequency
       iii    ! [ ]   Number of range samples to simulate
    ff.fff    ! [GHz] Simulation frequency
```

There are certain constant quantities that are used throughout the simulation, so instead of specifying these each time they are needed, the `GENERAL` function was designed. The

Table 3.2: Parameter Details for the GENERAL Function

Symbol	Unit	Range	Description
SGI	n/a	'A' to 'Z'	Simulation Group Identifier
N_{pri}	n/a	> 0	Number of PRIs to simulate
f_{prf}	Hz	> 0.0	Pulse Repetition Frequency
N_{bin}	n/a	> 0	Number of range bins to simulate
f_{sim}	GHz	> 0.0	Simulation frequency

internal data parameters of this function are accessible to all other functions in the same simulation group and so global simulation behaviour is defined.

All of the parameters for this function have already been defined in Chapter 2, hence the discussion here is kept to a minimum. In a standard simulation, N_{pri} determines how many times the inner processing loop is to execute. Recall that one iteration of the loop processes one range return for one specific PRI. The f_{prf} is used by the GEOMETRY function to calculate the incremental distance between each PRI, as will be illustrated later. N_{bin} determines the number of elements, or bins, in all of the data arrays used within that simulation group, while f_{sim} is actually the sampling frequency for the N_{bin} bins. Internally, the GENERAL function derives two further quantities. These are the time span covered by each range bin, t_{bin}, and the pulse repetition interval, t_{pri}. Equations 2.8 and 2.1 respectively are used to calculate these values. These equations are shown below again for clarity.

$$t_{bin} = 1/f_{sim} \qquad \text{and} \qquad t_{pri} = 1/f_{prf} \, .$$

Note that t_{bin} is now in microseconds, while t_{pri} is in seconds.

Finally, the _RUN part of this function does nothing, as this function has no processing purpose other than to provide the above global parameters to other functions.

3.5.2 Function TARGET

The command file format for this function is shown on the next page. This is followed by Table 3.3 which formally describes each parameter.

```
$TARGET  A
    ff.fff    ! [m] x direction offset from mid path
    ff.fff    ! [m] y ground range at closest approach
    ff.fff    ! [m] z height above ground level
```

Table 3.3: Parameter Details for the `TARGET` Function

Symbol	Unit	Range	Description
SGI	n/a	'A' to 'Z'	Simulation Group Identifier
x_t	m	real	x direction offset from mid path
y_t	m	real	y ground range at closest approach
z_t	m	real	z height above ground level

This function is used to define one point target in the simulation plane, and is usually called more than once in the command file.

As discussed in Chapter 2, the x_t, y_t and z_t parameters describe the Cartesian plane location of the target in the simulation world. It is very important to note that the origin of this set of axes is at the **middle** of the ideal flight path at **ground level**. Thus, if the co-ordinates of the target are $(0, 0, 0)$, then the aircraft will pass directly over the target on the ground exactly halfway through the simulated ideal flight path[6], or in mathematical terms, when the current simulation PRI counter, n_{pri} is equal to $(N_{pri} - 1)/2 + 1$ if N_{pri} is an odd positive integer.

At this point refer back to Figure 2.4 which shows the axis definitions for RadSim. According to these, a positive x_t represents a target further away from the starting position of the aircraft than one with a negative x_t. A positive y_t places the target to the right of the aircraft, if one is facing in the direction of the flight path. A positive z_t raises the target is above ground since the origin is at ground level.

As with the `GENERAL` function, the `_RUN` part of this function does nothing, as this function is meant to define target parameters only.

[6]The calculation of the aircraft flight path is discussed in the `GEOMETRY` function section.

3.5.3 Function MOTIONRND

The command file format for this function is shown below. This is followed by Table 3.4 which formally describes each parameter.

```
$MOTIONRND Yn    ! Create array Yn and place random motion data in it
         A       ! [ ]   Use which GENERAL parameters ?
     ff.fff      ! [m]   Maximum allowed +/- variation along x-axis
     ff.fff      ! [m]   Maximum allowed +/- variation along y-axis
     ff.fff      ! [m]   Maximum allowed +/- variation along z-axis
     ff.fff      ! [deg] Maximum allowed +/- variation in squint angle
```

Table 3.4: Parameter Details for the MOTIONRND Function

Symbol	Unit	Range	Description
Y_n	n/a	'M'	Output array, of type 'motion'
SGI	n/a	'A' to 'Z'	Simulation Group Identifier
$x_{m_{max}}$	m	real	Maximum allowed +/− variation along x-axis
$y_{m_{max}}$	m	real	Maximum allowed +/− variation along y-axis
$z_{m_{max}}$	m	real	Maximum allowed +/− variation along z-axis
$\alpha_{m_{max}}$	degrees	0.0 to 90.0	Maximum allowed +/− variation in squint angle

This function is used to generate pseudo-random offsets from the ideal flight path in Cartesian co-ordinates, and in squint angle. The generated offsets may then be passed onto the GEOMETRY function which uses these to perturb the calculated straight line flight path, and the squint angle of the aircraft. This function was designed with post-simulation testing of motion compensation algorithms in mind, and for the quantification of the effects of motion 'noise' on SAR processing algorithms.

Taking the x-axis parameter as an example, the random value is constructed as follows:

$$x_{m_{rnd}}[n_{pri}] = x_{m_{max}}\text{rand}() \qquad \text{where} \quad -1.0 \leq \text{rand}() \leq 1.0 \ .$$

The calculation is the same for the y- and z-axis and squint angle. The four results are obviously combined to form an offset vector whose origin is the position of the aircraft at the PRI iteration, n_{pri}, to which the vector applies. Consequently, it is clear that the output array, Y_n, must consist of N_{pri} rows, and four columns, thus giving the x-axis, y-axis, z-axis

and squint angle offset for each PRI within the simulation geometry. Note that the default squint angle of the aircraft is zero, and positive values indicate an anticlockwise squint. Further, the squint angle offset is given in degrees, but internally it is converted to radians, and this is the implied unit whenever future references are made to this parameter. The value given for any of these four parameters represents the one sided maximum, and so the total variation that may occur is twice that value - that is for example from $-x_{m_{max}}$ to $+x_{m_{max}}$ along the x-axis.

The _RUN part of this function first creates the output array, Y_n, of $4N_{pri}$ floating point elements. The library function **rand()** is then seeded with the current clock time, and a macro is defined that returns a random number in the range -1.0 to 1.0. A loop then iterates N_{pri} times, within which each of the four parameters is multiplied by a different random number as per the macro defined. The four results are then stored at the correct row in the output array.

3.5.4 Function MOTIONSIN

The command file format for this function is shown below. This is followed by Table 3.5 which formally describes each parameter.

```
$MOTIONSIN Yn    ! Create array Yn and place sinusoidal motion data in it
         A       ! [ ]           Use which GENERAL parameters ?
     ff.fff      ! [m]           Sine wave amplitude along x-axis
     ff.fff      ! [cycles/PRI]  Sine wave frequency along x-axis
     ff.fff      ! [m]           Sine wave "DC" offset along x-axis

     ff.fff ff.fff ff.fff ! Same for y-axis
     ff.fff ff.fff ff.fff ! Same for z-axis
     ff.fff ff.fff ff.fff ! Same for squint angle
```

Table 3.5: Parameter Details for the `MOTIONSIN` Function

Symbol	Unit	Range	Description
Y_n	n/a	'M'	Output array, of type 'motion'
SGI	n/a	'A'-'Z'	Simulation Group Identifier
x_{m_a}	m	real	Sine wave amplitude along x-axis
x_{m_f}	cycles/PRI	real	Sine wave frequency along x-axis
x_{m_o}	m	real	Sine wave "DC" offset along x-axis
y_{m_a}	m	real	Sine wave amplitude along y-axis
y_{m_f}	cycles/PRI	real	Sine wave frequency along y-axis
y_{m_o}	m	real	Sine wave "DC" offset along y-axis
z_{m_a}	m	real	Sine wave amplitude along z-axis
z_{m_f}	cycles/PRI	real	Sine wave frequency along z-axis
z_{m_o}	m	real	Sine wave "DC" offset along z-axis
α_{m_a}	deg	-90.0 to 90.0	Sine wave amplitude in squint angle
α_{m_f}	cycles/PRI	real	Sine wave frequency in squint angle
α_{m_o}	deg	-90.0 to 90.0	Sine wave "DC" offset in squint angle

Note that there is an additional constraint in that

$$|\alpha_{m_a}| + |\alpha_{m_o}| \leq 90.0 \ .$$

This function has exactly the same purpose as the `MOTIONRND` function except that the generated motion offsets are of a sinusoidal nature, rather than a random one. A sinusoidal generation function was chosen mainly for its versatility and simplicity. Unlike the `MOTIONRND` function, this function also produces predictable results, which should prove useful when debugging motion compensation algorithms.

Taking the x-axis parameter as an example, the sine value is constructed as follows:

$$x_{m_{sin}}[n_{pri}] = x_{m_a} \sin(2\pi x_{m_f} n_{pri}) + x_{m_o} \ .$$

The calculation is the same for the y- and z-axis and squint angle. The four results are obviously combined to form an offset vector whose origin is the position of the aircraft at the PRI iteration, n_{pri}, to which the vector applies. As with the `MOTIONRND` function, the output array, Y_n, must consist of N_{pri} rows, and four columns; the default squint angle of the aircraft is zero, and positive values indicate an anticlockwise squint; and finally two of the squint angle parameters, α_{m_a} and α_{m_o}, are given in degrees, but internally they are

converted to radians, and this is the unit meant whenever future references are made to these parameters.

The _RUN part of this function first creates the output array, Y_n, of $4N_{pri}$ floating point elements. Macros are then defined for the evaluation of the sine formula. A loop then iterates N_{pri} times, within which the sine formula is evaluated for each of the four parameter sets. The four results are then stored at the correct position in the output array.

3.5.5 Function ADD

The command file format for this function is shown below. This is followed by Table 3.6 which formally describes each parameter.

```
$ADD Yn        ! Create array Yn and place sum of the two input arrays in it
       Xn1     ! [ ] First array for addition
       Xn2     ! [ ] Second array for addition
```

Table 3.6: Parameter Details for the ADD Function

Symbol	Unit	Range	Description
Y_n	n/a	'A', 'D', 'F', 'G', 'M' or 'T'	Output array, of any type
X_{n_1}	n/a	'A', 'D', 'F', 'G', 'M' or 'T'	Input array 1, of any type
X_{n_2}	n/a	'A', 'D', 'F', 'G', 'M' or 'T'	Input array 2, or any type

Note that there are additional constraints in that the two input arrays must be of the same type and must have been generated by functions using the same SGI[7]. The output array must also be of the same type as the two input arrays.

The purpose if this function is to add together two arrays of the same type.

The two arrays are added on an element by element basis as shown in the following equation:

$$Y_n[n_{pri}] = X_{n_1}[n_{pri}] + X_{n_1}[n_{pri}] \ .$$

[7]This is to ensure that the two input arrays are of the same length.

For arrays that represent data in range, rather than azimuth, the array index would be n_{bin} and not n_{pri}. The function was designed with the `MOTIONRND` and `MOTIONSIN` functions in mind to allow one to add random noise to a sinusoidal motion offset pattern, that is to add two arrays of type 'M'. Also, a motion offset array of any shape can in fact be created by adding enough `MOTIONSIN` sinusoids together, as per the Fourier series theory. This feature is of course reserved for those very keen on typing. As the design is generic, this function also adds arrays of all other types.

The `_RUN` part of this function first creates the output array, Y_n, of the same type and length as the input arrays. A loop then iterates through the input arrays, within which the corresponding elements of the two input arrays are added together and stored at the correct position in the output array. Note that the output array takes on the SGI of the input arrays.

3.5.6 Function `GEOMETRY`

The command file format for this function is shown below. This is followed by Table 3.7 which formally describes each parameter.

```
$GEOMETRY Yn     ! Create array Yn and place simulation geometry data in it
        A        ! [ ]   Use which GENERAL parameters ?
       Xn        ! [ ]   Input array with aircraft offset motion data
    ff.fff       ! [m]   Aircraft altitude above ground level
    ff.fff       ! [m/s] Aircraft velocity (groundspeed)
```

Table 3.7: Parameter Details for the `GEOMETRY` Function

Symbol	Unit	Range	Description
Y_n	n/a	'G'	Output array, of type 'geometry'
SGI	n/a	'A' to 'Z'	Simulation Group Identifier
X_n	n/a	'M'	Input array, of type 'motion'
h	m	real	Aircraft altitude with respect to ground level
v	m/s	real	Aircraft velocity with respect to ground

Note that if one does not wish to use an aircraft motion offsets array, the keyword `NULL` must be used instead of an input array in the command file.

The purpose of this function is to generate the simulation geometry for each PRI position of the aircraft.

This is achieved by calculating the range, azimuth angle, and elevation angle for each target for each PRI, taking the current position of the aircraft as the origin. Strictly speaking, the aircraft is being kept still while the targets are advanced, but of course this results in the same set of values as if the aircraft was actually flown over the targets. Internally, two additional parameters are derived. The first is the total number of targets to be processed for that SGI, that is N_{tar}. As may be induced, the resulting output array has three columns and $N_{pri}N_{tar}$ rows. The second derived parameter is the starting position of the aircraft, x_{start}, and is calculated as follows, given that the origin must be the exact middle of the entire ideal flight path:

$$x_{start} = -d/2 \ ,$$

where d was calculated earlier in Equation 2.3.

The `_RUN` part of this function is the most complex one dealt with so far. First, the output array Y_n is created with length $3N_{tar}N_{pri}$. An outer loop is then defined to run from zero to $N_{tar} - 1$, using the counter n_{tar}. The current x-axis PRI position in metres, $x_{current}$ is then set to equal x_{start}. The inner loop is then defined to run from zero to $N_{pri} - 1$, using the counter n_{pri}. Within this inner loop, the relative position of the current target with respect to the current position of the aircraft is calculated in Cartesian co-ordinates as follows:

$$\begin{aligned} x_a[n_{pri}, n_{tar}] &= x_t[n_{tar}] - (x_{current} + x_m[n_{pri}]) \\ y_a[n_{pri}, n_{tar}] &= y_t[n_{tar}] + y_m[n_{pri}] \\ z_a[n_{pri}, n_{tar}] &= h - z_t[n_{tar}] + z_m[n_{pri}] \end{aligned} \ .$$

Recall that x_m, y_m and z_m are the aircraft motion offsets as generated by the `MOTIONRND` or `MOTIONSIN` simulation functions. Of course, if `NULL` was specified in the command file instead of an aircraft motion offset array, the motion offset additions in the above equations are not present. Once these three values are obtained, the Cartesian co-ordinates are converted to spherical co-ordinates, that is range r_a, azimuth angle α_{a_a} and elevation angle α_{e_a}

as follows:

$$\begin{aligned} r_a[n_{pri}, n_{tar}] &= \sqrt{x_a[n_{pri}, n_{tar}]^2 + y_a[n_{pri}, n_{tar}]^2 + z_a[n_{pri}, n_{tar}]^2} \\ \alpha_{a_a}[n_{pri}, n_{tar}] &= \tan^{-1}(x_a[n_{pri}, n_{tar}]/y_a[n_{pri}, n_{tar}]) - \alpha_m[n_{pri}] \\ \alpha_{e_a}[n_{pri}, n_{tar}] &= -\sin^{-1}(z_a[n_{pri}, n_{tar}]/r_a[n_{pri}, n_{tar}]) \end{aligned} ,$$

which is in accordance with Equations 2.4, 2.5 and 2.6 in the previous chapter. As before, the motion offset addition is not present when no aircraft motion offset data has been provided. These three values are then stored at the correct position in the output array. Finally, the value of $x_{current}$ is updated as follows:

$$x_{current} = x_{current} + d_{pri} ,$$

where d_{pri} was calculated earlier in Equation 2.2. After this, the process returns to the beginning of the inner loop. Once the geometry figures have been calculated for every simulation PRI for that target, the process moves onto the next target by returning to the beginning of the outer loop. The process completes after all targets have been processed.

Finally, the reason why the target geometry is converted from Cartesian to spherical co-ordinates is due to the fact that the radar equation[8] works with the range to a target to determine the strength of the return pulse from that target, while the directional antenna equation[9] requires the azimuth and elevation angles to determine the antenna gain for that same target.

3.5.7 Function PULSEGEN

The command file format for this function is shown on the next page. This is followed by Table 3.8 which formally describes each parameter.

[8]Refer to Equations 2.35 and 2.38 in Chapter 2.
[9]Refer to the section dealing with the RETURNL simulation function.

```
$PULSEGEN Yn1 Yn2 ! Create arrays Yn1, Yn2 and place waveform data in them
        A         ! [ ]   Use which GENERAL parameters ?
    ff.fff        ! [ns]  Pulse rise time
    ff.fff        ! [ns]  Pulse fall time
    ff.fff        ! [ns]  Pulse width
    ff.fff        ! [V]   Peak amplitude
       ttt        ! [ ]   'CHIRP' or 'MONO' for chirp or monochromatic pulse
    ff.fff        ! [GHz] Chirp pulse bandwidth; ignored for monochromatic
```

Table 3.8: Parameter Details for the PULSEGEN Function

Symbol	Unit	Range	Description
Y_{n_1}	n/a	'T'	Output array, of type 'time'
Y_{n_2}	n/a	'A'	Output array, of type 'amplitude'
SGI	n/a	'A' to 'Z'	Simulation Group Identifier
t_{rise}	ns	≥ 0.0	Pulse rise time
t_{fall}	ns	≥ 0.0	Pulse fall time
t_{length}	ns	> 0.0	Pulse length during peak amplitude
V_{peak}	V	$\neq 0.0$	Peak amplitude
n/a	n/a	'CHIRP'/'MONO'	Pulse type - chirp or monochromatic
f_{BW}	GHz	> 0.0	Bandwidth of chirp pulse

Note that there is an internal check that ensures that the total pulse length is less than or equal to the simulation space, that is:

$$t_{rise} + t_{width} + t_{fall} \leq t_{total}$$

where t_{total} was calculated earlier in Equation 2.9.

The purpose of this function is to generate a linear chirp or monochromatic pulse both in magnitude only and IQ pairs in the time domain. This is the pulse waveform whose transmission and return is simulated by the RETURNL function.

There are two types of pulses that may be generated. The first is the simple monochromatic pulse which has no quadrature component, and whose in-phase component is the same as the amplitude only version. The second is the chirp pulse where the frequency in I and Q varies linearly with time. The given bandwidth of this pulse determines the range of frequency variation. The length of the amplitude, 'A', output array is N_{range} while the

length of the time, 'T', output array is $2N_{range}$ as there is one column for I values and one for Q values.

The _RUN part of this function begins by creating the two output arrays, Y_{n_1}, with length $2N_{range}$, and Y_{n_2} with length N_{range}. Then, the amplitude only version of the pulse, $p_{chirp_{env}}$, is generated first according to Equation 2.17. These magnitude values are then stored in the correct place in the output array, Y_{n_2}. After this, the time domain pulse is created, each point consisting of an IQ pair. A monochromatic pulse, p_{mono} is generated as follows:

$$p_{mono_I}[n_{bin}] = p_{chirp_{env}}[n_{bin}]$$
$$p_{mono_Q}[n_{bin}] = 0.0$$

On the other hand, the chirp pulse, p_{chirp} is generated according to Equation 2.22 as discussed at length in the section on pulse generation in Chapter 2. The I and Q values are then stored in the correct place in the output array, Y_{n_1}.

3.5.8 Function MATCHFILT

The command file format for this function is shown below. This is followed by Table 3.9 which formally describes each parameter.

```
$MATCHFILT Yn ! Create array Yn and place matched filter array in it
       Xn     ! [ ] Input waveform to be matched
   fff.ff     ! [ ] Hanning windowing factor ( 1.0 = no windowing )
```

Table 3.9: Parameter Details for the MATCHFILT Function

Symbol	Unit	Range	Description
Y_n	n/a	'F'	Output array, of type 'Frequency'
X_n	n/a	'F'	Input array, of type 'Frequency'
W	n/a	≥ 0.0	Hanning windowing factor

Note that there is a further condition that if $W = 1.0$, no windowing is performed on the input filter waveform.

This function is used to generate a matched filter waveform for the input waveform.

Almost always, the input waveform is exactly the same as the waveform whose transmission and return is to be simulated. The matched filter is then used within the simulation loop and multiplied in the frequency domain with the so-called returned pulse in order to perform range compression. In addition, the input waveform may be multiplied by a Hanning window in order to attenuate the sidelobes of the matched filter, and thus of the range compressed pulse.

The output array, Y_n, is created to be the same length as the input array, X_n, and with the same SGI. An ideal matched filter is achieved by negating the Quadrature part of the input waveform as discussed in Chapter 2 in the section dealing with matched filtering. Equation 2.31 is implemented as follows:

$$Y_{n_I}[n_{bin}] = X_{n_I}[n_{bin}] \qquad \text{and} \qquad Y_{n_Q}[n_{bin}] = -X_{n_Q}[n_{bin}] \ .$$

The Hanning windowing function $w[n]$ for a discrete system such as this is defined as [21]:

$$w[n] = \begin{cases} \frac{1}{2}(1 - \cos(2\pi n/M)) & \text{if} \quad 0 \leq n \leq M \\ 0 & \text{otherwise} \end{cases} ,$$

where M is the number of bins that make up the pulse to be windowed. Using the trigonometric identity [21]

$$\sin^2 u = \frac{1 - \cos 2u}{2} ,$$

and the fact that the entire input array is considered to be the waveform to be matched, the windowing equation changes to

$$w[n_{bin}] = \sin^2(\pi n_{bin}/(N_{bin} - 1)) \qquad \text{where} \quad 0 \leq n_{bin} < N_{bin} ,$$

noting that n_{bin} begins from zero, as when indexing an array in 'C', one must be subtracted from N_{bin} in the above equation. A modification added by Horrell [7, 8] allows for the window to be scaled in amplitude using the windowing constant, W, as follows:

$$w[n_{bin}] = W + (1 - W)\sin^2(\pi n_{bin}/(N_{bin} - 1)) \qquad \text{where} \quad 0 \leq n_{bin} < N_{bin} \ .$$

Note here that if W is set to zero, one obtains the original windowing equation. If W is set to one, the equation reduces to a constant $w[n_{bin}] = 1$. Although the command file parser allows for $W > 1.0$, the use of a value larger than one is not very useful. It is important to realise that the Hanning windowing function must be applied to the input waveform in the time domain and not the frequency domain.

The _RUN part of this function begins by creating the output array, Y_n, with the same length as the input array, X_n. If Hanning windowing is required ($W \neq 1.0$), then the input array is transformed into the time domain, a loop is used to iterate through the and multiply each complex pair by the windowing function. Thereafter, the array is converted back into the frequency domain. A loop then iterates through the array, that is from 0 to $N_{bin} - 1$, and the output array IQ pair is set to equal the complex conjugate of the input IQ pair.

3.5.9 Function FFT

The command file format for this function is shown below. This is followed by Table 3.10 which formally describes each parameter.

```
$FFT    Yn      ! Create array Yn and place transformed array in it
        Xn      ! [ ] Input waveform to be transformed
```

Table 3.10: Parameter Details for the FFT Function

Symbol	Unit	Range	Description
Y_n	n/a	'T', 'F'	Output array, of type 'Time' or 'Frequency'
X_n	n/a	'F', 'T'	Input array, of type 'Frequency' or 'Time'

Note that there is a condition that if the input array is of type 'Time', then the output array must be of type 'Frequency', and vice versa.

This function performs a Discrete Complex Fast Fourier Transform (CFFT) on a waveform in the time domain, resulting in a frequency domain transform. The function also performs the inverse action, transforming a frequency domain waveform back into the time domain.

The transforms have been defined in Chapter 2, and are governed by Equations 2.27 and 2.28. The software implementation of the CFFT and its inverse was taken from the "Numerical

Recipes In C" library source code [20].

The _RUN part of this function, creates the output array, Y_n, of length N_{bin} pairs, and of type 'Frequency' if the input array, X_n, is of type 'Time', or of type 'Time' if X_n is of type 'Frequency'. Then, X_n is copied into Y_n and Y_n is then passed into the CFFT function with a flag indicating whether a forward or inverse transform is required. After completion, if the inverse transform had been performed, Y_n is divided by N_{bin} to scale the amplitude correctly.

3.5.10 Function AMPLIFY

The command file format for this function is shown below. This is followed by Table 3.11 which formally describes each parameter.

```
$AMPLIFY Yn    ! Create array Yn and place amplified array in it
       Xn      ! [ ] Input waveform to be amplified
   fff.ff      ! [dB] Amplification factor
```

Table 3.11: Parameter Details for the AMPLIFY Function

Symbol	Unit	Range	Description
Y_n	n/a	'T' or 'F'	Output array, of type 'Time' or 'Frequency'
X_n	n/a	'T' or 'F'	Input array, of type 'Time' or 'Frequency'
G_a	dB	real	Amplification gain factor

There is a further condition that the output array type must be the same as the input array type.

This function is used to simulate a power amplifier.

The input array, X_n, is simply copied into the output array, Y_n, with each element being multiplied by the gain, $10^{G_a/20}$ as shown in Equation 2.25 in Chapter 2.

The _RUN part of this function begins by creating the output array, Y_n, with the same length as the input array, X_n. A loop is then defined to run from 0 to $N_{bin} - 1$, within which each IQ pair is copied from X_n into Y_n and multiplied by $10^{G_a/20}$ at the same time.

3.5.11 Function MULTIPLY

The command file format for this function is shown below. This is followed by Table 3.12 which formally describes each parameter.

```
$MULTIPLY Yn   ! Create array Yn and place product in it
       Xn1     ! [ ] First input waveform to be multiplied
       Xn2     ! [ ] Second input waveform to be multiplied
```

Table 3.12: Parameter Details for the **MULTIPLY** Function

Symbol	Unit	Range	Description
Y_n	n/a	'T' or 'F'	Output array, of type 'Time' or 'Frequency'
X_{n_1}	n/a	'T' or 'F'	Input array 1, of type 'Time' or 'Frequency'
X_{n_2}	n/a	'T' or 'F'	Input array 2, of type 'Time' or 'Frequency'

Note that there is a further condition that all three arrays must be of the same type, that is all three must be either of type 'Time' or type 'Frequency'. Also, the two input arrays must have the same SGI.

This function is used to multiply two time domain or two frequency domain waveforms. It is used most often to perform range compression, by multiplying the transform of a matched filter with the transform of the so-called return waveform, and thus performing convolution of the received waveform with its matched filter.

Refer to Equation 2.26 and the corresponding discussion in Chapter 2 with regard to the simple mathematics involved.

The **_RUN** part of this function creates the output array, Y_n, with the same length as the input arrays, X_{n_1} and X_{n_2}. A loop is the defined to run from zero to $N_{bin} - 1$, within which the complex elements of the two input arrays are multiplied and stored in Y_n.

3.5.12 Function A2D

The command file format for this function is shown on the next page. This is followed by Table 3.13 which formally describes each parameter.

```
$A2D    Yn    ! Create array Yn and place digitised values in it
        Xn    ! [ ]            Input waveform to be digitised
    fff.ff    ! [V]            LSB value
       iii    ! [bits]         Number of bits of A2D precision
       iii    ! [bins]         First bin to digitise ( inclusive )
       iii    ! [bins]         Number of bins to digitise after first bin
       iii    ! [bins/sample]  Bin sampling rate
```

Table 3.13: Parameter Details for the A2D Function

Symbol	Unit	Range	Description
Y_n	n/a	'D'	Output array, of type 'Digitised'
X_n	n/a	'T' or 'F'	Input array, of type 'Time' or 'Frequency'
V_{LSB}	V	> 0.0	Volt value of lowest significant bit
N_{bits}	bits	4 or 8	Number of bits of A2D precision
B_{start}	bins	0 to $N_{bin} - 1$	First bin to begin digitisation at
B_{num}	bins	1 to $N_{bin} - 1$	Number of bins to digitise after first bin
S_{rate}	bins/sample	1 to $N_{bin} - 1$	Bin sampling rate

Note that there is a further condition that if B_{num} is set to zero, the conversion continues until the end of the input array, X_n is reached. Further, the following must be true:

$$B_{num} + B_{start} < N_{bin}$$

This function is used to simulate an Analogue to Digital converter, as one would be used before data storage in a real system.

The conversion begins at the B_{start}'th IQ pair in the input array, X_n, and continues for B_{num} IQ pairs, or until the last IQ pair if B_{num} is set to zero. Note that only every S_{rate}'th IQ pair in that range is digitised. The digitisation is performed in accordance with Equation 2.46 and the related discussion in Chapter 2, with the one difference that here a "DC" offset is added each time so that a floating point value of 0.0 is converted to an integer value equal to $2^{N_{bits}-1} - 1$ and not 0. This adjustment allows for negative floating point values to be converted correctly.

The _RUN part of this function creates the output array, Y_n to be of length $(B_{num} - B_{start})/S_{rate}$. A main loop is then defined to run from B_{start} to $B_{start} + B_{num}$, incre-

menting the counter by S_{rate} at a time. This counter is used to index X_n, a separate counter, starting from zero and incrementing by one, is used to index Y_n. Within this loop, each element in X_n is digitised and stored at the appropriate position in Y_n. Within the loop, count is also kept of the number of overflows, underflows, the largest digitised value, the smallest digitised value and the number of digitised values that were zero. These counts are then printed on screen for the user to make meaningful adjustments to the V_{LSB} value, or to amplification stages within the simulation. Note that by virtue of the sampling rate, S_{rate}, being defined as the number of bins to skip for every sample, it follows that a sample must always coincide with a range bin and thus no interpolation is possible. For example it is not possible to take two samples for every three bins.

3.5.13 Function RETURNL

The command file format for this function is shown below. This is followed by Table 3.14 which formally describes each parameter.

```
$RETURNL Yn    ! Create array Yn and place waveform radar return in it
       Xn1     ! [ ]   Simulation geometry array to be used
       Xn2     ! [ ]   Envelope of pulse to be transceived
    fff.ff     ! [GHz] Radar centre frequency
    fff.ff     ! [dB]  Antenna gains
    fff.ff     ! [dB]  Losses ( TxLoss + RxLoss + PropagationLoss )
    fff.ff     ! [m^2] Target cross-sectional area
    fff.ff     ! [m]   Range offset for simulation geometry
       ttt     ! [ ]   Pulse type - 'CHIRP' or 'MONO'
    fff.ff     ! [GHz] Chirp bandwidth (arb non-zero for MONO)
       ttt     ! [ ]   Antenna type - 'SIN' or 'NULL'
    fff.ff     ! [deg] Antenna elevation beamwidth
    fff.ff     ! [deg] Antenna azimuth beamwidth
    fff.ff     ! [deg] Antenna elevation angle (down is negative)
    fff.ff     ! [deg] Antenna squint angle (backward is positive)
```

Table 3.14: Parameter Details for the RETURNL Function

Symbol	Unit	Range	Description
Y_n	n/a	'T'	Output array, of type 'Time'
X_{n_1}	n/a	'G'	Input array 1, of type 'Geometry'
X_{n_2}	n/a	'A'	Input array 2, of type 'Amplitude'
f_{centre}	GHz	≥ 0.0	Radar centre frequency
G_{ant}	dB	real	Antenna gains
L_{txrx}	dB	real	Losses (TxLoss+RxLoss+PropagationLoss)
a_{rcs_t}	m^2	> 0.0	Target cross-sectional area
R_o	m	real	Range offset for simulation geometry
n/a	n/a	'CHIRP'/'MONO'	Pulse type - chirp or monochromatic
f_{BW}	GHz	> 0.0	Bandwidth of linear chirp pulse
n/a	n/a	'SIN'/'NULL'	Antenna gain type - sinusoidal or omni.
θ_e	deg	0.0 to 180.0	Antenna elevation beamwidth
θ_a	deg	0.0 to 180.0	Antenna azimuth beamwidth
α_e	deg	-90.0 to 90.0	Antenna elevation angle
α_a	deg	-90.0 to 90.0	Antenna azimuth angle

There is a further condition that the two input arrays X_{n_1} and X_{n_2} must belong to the same SGI.

The purpose of this function is to simulate the transmission and return of a pulse in the simulation world.

Most of the mathematics has already been dealt with in Chapter 2, and as explained there, Equation 2.45 is used to calculate the return waveform. However, a number of modifications had to be made to make the processing viable.

First, it was found that t_{delay} was almost always much greater than the total width of the pulse received, t_{pulse}. Therefore, most of the calculated return array was filled with zeroes from the beginning, and only after N_{delay} elements was some useful data found. Therefore, the higher f_{sim} is, the more memory and processing time was being wasted. It was decided to introduce the R_o constant [8, 12, 13], which indicates how many metres are to be subtracted from the range to the target r_a. The result is that **all** range values within the simulation are brought closer to the aircraft by R_o metres. Although this changes the phase of the received waveform, it does not affect the **relative** phase between successive return waveforms and the points within them, which is the important issue as it is this relative phase that is used for azimuth compression and so a focused image with the correct proportions is still

obtained [8]. Returning to the original problem, since N_{delay} is now smaller, the length of the array that holds the return pulse can now also be made smaller, or the range direction sampling rate, f_{sim}, can be made larger so as to produce simulation results with a higher time resolution.

Secondly, there was a problem with excessive floating point inaccuracies. In order to evaluate Equation 2.45, very large numbers interact with very small numbers in several places. This causes severe loss of precision. Two things were done to alleviate this problem in so far as possible. Firstly, the units of some of the parameters given in the command file were changed - for example from watts to kilowatts and from hertz to gigahertz. Secondly, Equation 2.45 was rearranged and split up into a number of sub-equations such that the modified units cancelled successfully and so that large numbers did not interact with small numbers. These changes were successful in restoring the precision back to the desired five decimal places for the return waveform.

Finally, a problem that was most difficult to overcome and required the most modifications was that of so-called phase jumping. The complex pair time domain pulse waveform as generated by the `PULSEGEN` function was originally an input to this function, where the phase of the waveform, ϕ_{chirp}, was already calculated by the generating function at specific sampling points. The offset phase due to range, ϕ_{delay}, was then added to this phase. Note that the phase was dependant on t_{delay}, which was rounded off to the nearest t_{bin}. All the rounding resulted in a situation where the aircraft had to move along for a good number of PRIs before the range to the target changed so much so as to affect the phase. As it is the phase information that carries the most information in a SAR image, and not the magnitude, this was not acceptable.

It was decided to forego all rounding of any possible quantities that affected the phase information. In order to achieve this the original pulse had to be generated locally. Therefore, the input to the function was now an array of type 'A' which only contains magnitude information of the pulse. The ϕ_{delay} was calculated using exact floating point figures, especially a t_{delay} which had not been rounded to the nearest t_{bin}. Secondly, the exact phase of the linear chirp or monochromatic pulse ϕ_{chirp} was calculated for that exact t_{delay}. ϕ_{chirp} and ϕ_{delay} were added together and only then was the split into I and Q parts performed. Thus,

even though it still took a number of PRIs for the magnitude to change, the phase was now changing smoothly with each PRI, resulting in a more accurate simulation of the real entities in question [8].

Also, there is built in support for a sinusoidal gain response antenna in addition to the standard omnidirectional one. The voltage gain of the sinusoidal antenna with respect to elevation angle is governed by the following [8, 10] :

$$G_{ant_{sin_e}}(\alpha_e) = \left| \frac{\sin(\frac{0.88}{\theta_e}\pi \sin(\alpha_e))}{\frac{0.88}{\theta_e}\pi \sin(\alpha_e)} \right| ,$$

where an α_e that is zero with respect to the target results in the greatest gain. For the gain pattern in the azimuth angle plane, $G_{ant_{sin_a}}(\alpha_a)$ the equation is the same except that the azimuth beamwidth θ_a is used instead of θ_e. Note that all the angles must be in radians. Figure 3.3 below shows the shape of this antenna gain pattern in the dimension of elevation. In the figure, θ_e has been set to a realistic 12 degrees. The dotted line crosses the profile at its two -3dB points. As expected, the gain above -3dB approximately spans the required 12 degrees.

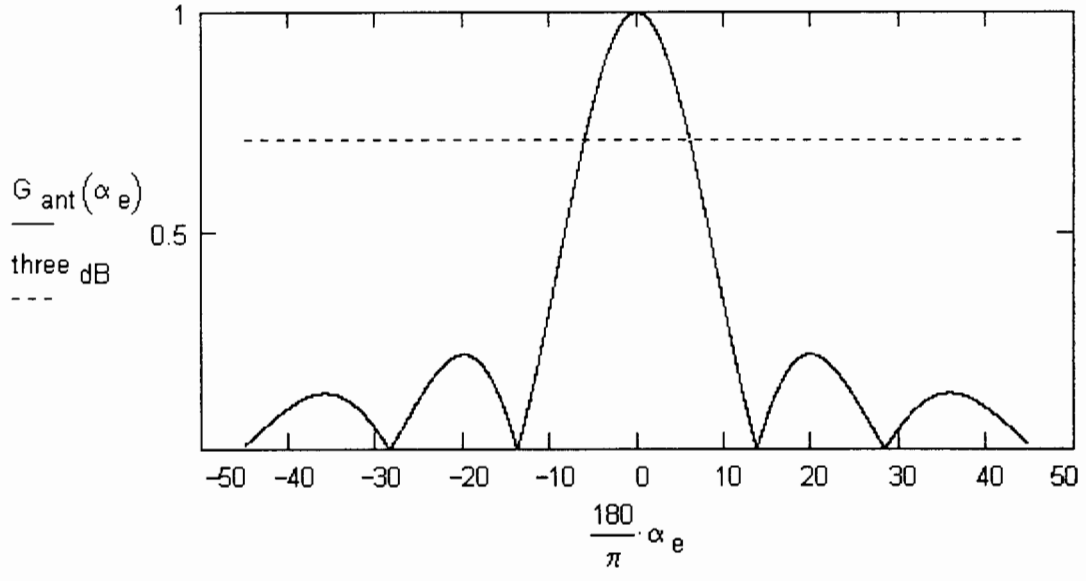

Figure 3.3: Sinusoidal Antenna Gain Pattern

The _RUN part of this function is the most complex of all the simulation functions. It begins by creating the output array, Y_n, with N_{bin} number of complex pair elements. Then, as in the PULSEGEN function, t_{mid} (Equation 2.13) and t_{width} (Equation 2.12) are calculated. In addition, a new variable t_o is calculated according to

$$t_o = 2R_o/c \ .$$

Then, a partial calculation of G is performed. An outer loop is then set up to run from 0 to $N_{tar} - 1$. The return pulse will now be calculated completely for each target and all of the results summed. A modified t_{delay} is then calculated as per Equation 2.32, but with t_o subtracted from it as explained earlier. The rest of the gain G is then calculated. If a sinusoidal response antenna was requested, the effective elevation and azimuth angles to the current target are calculated, and $G_{ant_{sin_e}}$ and $G_{ant_{sin_a}}$ are worked out. The total gain G is them multiplied by these two gains squared[10]. N_{delay} and ϕ_{delay} are then calculated. An inner loop is then defined within which the original pulse is calculated in the same way as in the PULSEGEN function, but taking into account the new phase, and the gain factor G.

Once the computation of the return waveform has been completed, the internal PRIs left counter is decremented[11].

3.5.14 Function ENDLOOP

The command file format for this function is shown below. This is not followed by a table which formally describes each parameter, since there are no parameters.

```
$ENDLOOP        ! Loop ends, re-trace back to the last RETURNL call
```

The purpose of this function is to return the processing sequence to the first unfinished RETURNL function in the command file.

The _RUN part of this function loops forwards through the simulation function call array, an resets the simulation function call array counter to point to the first uncompleted RETURNL

[10] once for transmission and once for reception, and this in elevation and azimuth.
[11] Refer to the ENDLOOP function where the use of this counter is explained.

function call found in the array. Recall that the RETURNL function keeps count of the number of PRIs it must process and decreases this count each time it is called, thus allowing the ENDLOOP function to determine whether a loop has been completed (RETURNL PRI counter $= -1$) or not (RETURNL PRI counter ≥ 0).

3.5.15 Function WRITE

The command file format for this function is shown below. This is followed by Table 3.15 which formally describes each parameter.

```
$WRITE        ! No output array
      Xn      ! [ ] Array to be written to file
    ttt1      ! [ ] File type, ASCII or binary ( 'ASC' or 'BIN' )
    ttt2      ! [ ] Write mode, append or overwrite ( 'APP' or 'OVR' )
    ttt3      ! [ ] Valid file name for output file
```

Table 3.15: Parameter Details for the WRITE Function

Symbol	Unit	Range	Description
X_n	n/a	'A', 'D', 'F', 'G', 'M' or 'T'	Input array, of any type
n/a	n/a	'ASC' or 'BIN'	File type, ASCII or Binary
n/a	n/a	'APP' or 'OVR'	File mode, Append or Overwrite
n/a	n/a	text	File name, valid for current OS

This function is used to write a data array of any type to file.

The data may be written out as ASCII, neatly justified if there is more than one column, or in raw Binary whereby the memory occupied by the array is simply dumped to file. An ASCII file would usually be required for human perusal, and for loading into third party software such as MathCAD. A Binary file would be used as input to post-processing software such as Horrell's SAR Azimuth Compression Processor[9]. There is also the choice of file writing mode. In Append mode, if a file with that file name already exists, the specified array is appended to the existing file. This is useful for writing arrays within the RETURNL loop, resulting in one large file containing data for each PRI simulated[12]. The Overwrite

[12]It is interesting to note here that the old VAX based simulator produced a separate file for each PRI iteration [3]. These files then had to be painstakingly joined together one by one [8].

mode is normally used outside of a loop to ensure that new data replaces the old in the file from one simulation run to another. Finally, the file name may be any text string of up to 256 characters in length as long as it does not contain white space and constitutes a valid file name for the OS currently in use.

The _RUN part of this function creates an outer loop that runs from zero to the length of the input array less one. An inner loop is then defined to run from zero to the number of columns defined for X_n less one. In the ASCII file case, each value is then printed as a floating point number with six decimal places, except for data from a 'digital' type array where each element is written as the 'C' language 'char' type. A tab character is then written, and the element in the next column is written. After all columns have been processed, the next row is dealt with until the end of the array. For a Binary file, the memory occupied by X_n is dumped to file, except for a 'D' type array which is first converted to an array of 'char' before dumping.

3.6 Formats of Output Data Files

This section discusses the formats of the output data files as generated by the WRITE simulation function for each of the six array types.

3.6.1 Arrays of Type 'A' - Amplitude

This array type may be thought of as consisting of one column of waveform amplitude values. There is a value for each range bin, and thus we are dealing with a 1 column by N_{bin} row array of floating point values. These values are written to file as output in two different formats, these being ASCII and binary.

In the ASCII case, the file is arranged in one column, and N_{bin} rows. The floating point values are printed to five decimal places.

In the binary case, the output file is a raw binary dump of the memory occupied by the array. Thus, the size of the output file in bytes is N_{bin} multiplied by the size of the float type as defined on the computing platform in use. The order in which the binary data is

written to file is the same as for the ASCII case.

Table 3.16 clarifies the units in which the values are written to file.

Table 3.16: Units For 'A'-Type Output Files

Symbol	Unit	Description
V	volts	Amplitude of waveform in time domain

Finally, as an example of this output format in ASCII, a few rows are shown that were cut from a real output file:

```
0.00000
0.21747
0.49754
0.79248
1.57714
2.44775
2.45751
2.13769
```

3.6.2 Arrays of Type 'D' - Digitised

This array type may be thought of as consisting of two columns, one each for in-phase digitised values and quadrature digitised values. There is an IQ pair for each sample, and thus we are dealing with a 2 column by B_{num}/S_{rate} row array of single-byte integer values. These values are written to file as output in two different formats, these being ASCII and binary.

In the ASCII case, the file is arranged in two columns separated by tabs, and B_{num}/S_{rate} rows. The order of the columns is I first and then Q.

In the binary case, the output file is a raw binary dump of the memory occupied by the array. Thus, the size of the output file in bytes is $2B_{num}/S_{rate}$. Note that whether one bit or eight bits of A2D precision was used, each value is still converted into an 8-bit, or 1 byte, `char` integer type.

Table 3.17 clarifies the units in which the values are written to file.

Table 3.17: Units For 'D'-Type Output Files

Symbol	Unit	Description
I	Byte	In-phase digitised ampl. for waveform in time or freq. domain
Q	Byte	Quadrature digitised ampl. for waveform in time or freq. domain

Finally, as an example of this output format in ASCII, a few rows are shown that were cut from a real output file:

```
 11          255
 15          255
 69          243
105          200
135          185
127          143
127          165
186          134
```

3.6.3 Arrays of Type 'F' - Frequency

This array type may be thought of as consisting of two columns, one each for in-phase values and quadrature values. There is an IQ pair for each range bin, and thus we are dealing with a 2 column by N_{bin} row array of floating point values. These values are written to file as output in two different formats, these being ASCII and binary.

In the ASCII case, the file is arranged in two columns separated by tabs, and N_{bin} rows. The order of the columns is I first and then Q. The floating point values are printed to five decimal places.

In the binary case, the output file is a raw binary dump of the memory occupied by the array. Thus, the size of the output file in bytes is $2N_{bin}$ multiplied by the size of the `float` type as defined on the computing platform in use. The order in which the binary data is written to file is the same as for the ASCII case.

Table 3.18 clarifies the units in which the values are written to file.

Table 3.18: Units For 'F'-Type Output Files

Symbol	Unit	Description
I	volts	In-phase amplitude for waveform in frequency domain
Q	volts	Quadrature amplitude for waveform in frequency domain

Finally, as an example of this output format, a few rows are shown that were cut from a real output file:

```
 0.52263      -1.21375
-0.40906      -1.14257
 0.49603      -1.11723
-0.38704      -1.04450
 0.46942      -0.72070
-0.36501      -0.64642
 0.44281      -0.42417
-0.34298      -0.34834
```

3.6.4 Arrays of Type 'G' - Geometry

This array type may be thought of as consisting of three columns, one each for range to target r_a, azimuth to target α_{a_a} and elevation to target α_{e_a}. There is a new set of these values for each PRI for each target, and thus we are dealing with a 3 column by $N_{pri}N_{tar}$ row array of floating point values. These values are written to file as output in two different formats, these being ASCII and binary.

In the ASCII case, the file is arranged in three columns separated by tabs, and $N_{pri}N_{tar}$ rows. The order of the columns is first r_a, then α_{a_a} and finally α_{e_a}. The floating point values are printed to five decimal places. Note that the rows are grouped by PRI and not by target. Thus, if there are three targets, there will be three rows for PRI number 1, then three rows for PRI number 2 and so on.

In the binary case, the output file is a raw binary dump of the memory occupied by the array. Thus, the size of the output file in bytes is $3N_{pri}N_{tar}$ multiplied by the size of the `float` type as defined on the computing platform in use. The order in which the binary

data is written to file is the same as for the ASCII case.

Table 3.19 clarifies the units in which the values are written to file.

Table 3.19: Units For 'G'-Type Output Files

Symbol	Unit	Description
r_a	metres	Range to target
α_{a_a}	radians	Azimuth angle to target
α_{e_a}	radians	Elevation angle to target

Finally, as an example of this output format, a few rows are shown that were cut from a real output file:

```
5083.45751      0.65552      -1.19628
5894.13769      0.51912      -1.03290
5083.01708      0.62896      -1.19979
5893.79248      0.49712      -1.03484
5082.57714      0.60239      -1.20328
5893.44775      0.47511      -1.03678
5082.13769      0.57581      -1.20678
5893.10351      0.45310      -1.03871
```

3.6.5 Arrays of Type 'M' - Motion

This array type may be thought of as consisting of four columns, one each for x-axis offsets, y-axis offsets, z-axis offsets and squint angle offsets. There is a new set of these offsets for each PRI, and thus we are dealing with a 4 column by N_{pri} row array of floating point values. These values are written to file as output in two different formats, these being ASCII and binary.

In the ASCII case, the file is arranged in four columns separated by tabs, and N_{pri} rows. The order of the columns is x_m, y_m, z_m and finally α_m. The floating point values are printed to five decimal places.

In the binary case, the output file is a raw binary dump of the memory occupied by the array. Thus, the size of the output file in bytes is $4N_{pri}$ multiplied by the size of the `float`

type as defined on the computing platform in use. The order in which the binary data is written to file is the same as for the ASCII case.

Table 3.20 clarifies the units in which the values are written to file.

Table 3.20: Units For 'M'-Type Output Files

Symbol	Unit	Description
x_m	metres	Aircraft motion offset along x-axis
y_m	metres	Aircraft motion offset along y-axis
z_m	metres	Aircraft motion offset along z-axis
α_m	radians	Aircraft motion offset in squint angle

Finally, as an example of this output format, a few rows are shown that were cut from a real output file:

```
 1.26171     1.72263    -1.75263    -0.21375
 1.41748     1.20906    -0.64257    -0.03456
-0.82470     1.49603     0.31723     0.29547
 2.07519     1.48704    -1.54450    -0.17940
-0.38867     1.46942     0.72070     0.11856
 1.73339     1.36501     0.94642    -0.24587
 1.95312     1.84281     0.12417     0.00138
-1.39257     1.14298    -1.04834     0.09437
```

3.6.6 Arrays of Type 'T' - Time

This array type may be thought of as consisting of two columns, one each for in-phase values and quadrature values. There is an IQ pair for each range bin, and thus we are dealing with a 2 column by N_{bin} row array of floating point values. These values are written to file as output in two different formats, these being ASCII and binary.

In the ASCII case, the file is arranged in two columns separated by tabs, and N_{bin} rows. The order of the columns is I and Q. The floating point values are printed to five decimal places.

In the binary case, the output file is a raw binary dump of the memory occupied by the

array. Thus, the size of the output file in bytes is $2N_{bin}$ multiplied by the size of the `float` type as defined on the computing platform in use. The order in which the binary data is written to file is the same as for the ASCII case.

Table 3.21 clarifies the units in which the values are written to file.

Table 3.21: Units For 'T'-Type Output Files

Symbol	**Unit**	**Description**
I	volts	In-phase amplitude for waveform in time domain
Q	volts	Quadrature amplitude for waveform in time domain

Finally, as an example of this output format, a few rows are shown that were cut from a real output file:

```
-0.43675      -1.43636
 0.45723       1.67934
-0.44577      -1.56838
 0.48454      -1.56498
 0.46457       0.54686
 0.46545      -0.85396
-0.44457      -0.97633
-0.44467       0.86579
```

Chapter 4

Simulation Software Testing

In this chapter, results of tests performed on RadSim and the analyses of these results are presented. The tests are divided into eight sections preceded by an introduction. In this introduction is explained the motivation behind the specific sectioning of these tests, and the information layout within each section. The chapter ends with a report on the results of RadSim portability tests.

4.1 Introduction to the Simulation Function Tests

Each simulation function was designed and written to be independent of the others, and each may be tested on its own with the help of a symbolic debugger. Obviously each function was tested this way, with different sets of parameters to verify behaviour under all possible conditions. However, this is not the place to detail the results of these hundreds of tests.

Instead, it is much more interesting and meaningful to test the functions in related groups using selected sections of a typical command file and running this command file through RadSim. This will also show the interaction of different simulation functions with each other and show how data is passed between them.

Therefore, the testing of the simulation functions has been divided into eight self contained sections. For each section, the command file is presented and explained. An attempt

has been made to make all simulation function parameters as realistic as possible. Each command file is followed by a MathCAD sheet showing the mathematical relations under test. Thereafter, the results of the simulation are shown as plots and as raw data after being imported into MathCAD. These shown results are analysed, and the discussion ends off with a short summary of the tests presented in that section.

Note that some command files repeat parts of the command files from earlier sections. The descriptions that follow each command file make it clear which part of that command file is specifically under test in that section. Further, the system of units used in the MathCAD sheets is based on the metre, kilogram and second (MKS) standard. Finally, each test section will begin on a new page so as to show as much of the command file for that test as possible on one page. This eases referencing and presents an unbroken chain of thought with regard to study of the command file.

4.2 Test of the GENERAL, MOTIONSIN and WRITE Functions

The command file used for this test is shown below:

```
! TEST1.CMD - Test of the GENERAL, MOTIONSIN and WRITE functions.
! ================================================================

$GENERAL A
        32    ! [ ]   Number of PRIs to simulate
     400.0    ! [Hz]  Pulse Repetition Frequency
       256    ! [ ]   Number of range samples to simulate
       0.2    ! [GHz] Simulation frequency

$MOTIONSIN M1 ! Create array M1 and place sinusoidal motion data in it
         A    ! [ ]          Use which GENERAL parameters ?
      0.10    ! [m]          Sine wave amplitude along x-axis
    0.1250    ! [cycles/PRI] Sine wave frequency along x-axis
      0.05    ! [m]          Sine wave "DC" offset along x-axis

    0.10 0.0625 0.20 ! Same for y-axis
    0.05 0.1250 0.00 ! Same for z-axis
    0.50 0.0625 0.25 ! Same for squint angle

$WRITE        ! No output array
        M1    ! [ ] Array to be written to file
       ASC    ! [ ] File type, ASCII or binary ( 'ASC' or 'BIN' )
       OVR    ! [ ] Write mode, append or overwrite ( 'APP' or 'OVR' )
    M1.ASC    ! [ ] Valid file name for output file
```

In this simulation, the **GENERAL** function is called first with an SGI 'A', and parameters such that there will be 32 pulse repetition intervals at a frequency of 400 Hz. In the range direction there are 256 bins, at a sampled frequency of 200 megahertz. Next, the **MOTIONSIN** function is called. The function takes general parameters with SGI 'A'. The other parameters define the four sine waves along the x-axis, y-axis, z-axis and squint angle respectively. The output of the function is the motion offsets array **M1**. Finally, the **WRITE** function dumps the array **M1** to an ASCII file called **M1.ASC** at the same time overwriting any existing file with that name. These simulation functions have been described in Sections 3.5.1, 3.5.4, and 3.5.15 respectively.

The partial MathCAD sheet below shows all the mathematical relationships involved in the

calculation of the sinusoidal motion offsets. These are self explanatory and have been dealt with in the sections mentioned above as well as in the appropriate sections of Chapter 2. Strictly speaking, the counter n_{pri} should start at $-N_{pri}/2$ and end at $N_{pri}/2-1$. However, the starting position turns out to be rather arbitrary, and so all arrays are treated as C arrays starting at 0 and ending at $N_{pri}-1$ which simplifies matters somewhat. Note that the sine wave frequencies are given in cycles per PRI.

$N_{pri} := 32$ $\quad n_{pri} := 0, 1 .. N_{pri} - 1$

$x_{m.a} := 0.10 \cdot m$ $\quad y_{m.a} := 0.10 \cdot m$ $\quad z_{m.a} := 0.05 \cdot m$ $\quad \alpha_{m.a} := 0.50 \cdot deg$

$x_{m.f} := 0.1250$ $\quad y_{m.f} := 0.0625$ $\quad z_{m.f} := 0.1250$ $\quad \alpha_{m.f} := 0.0625$

$x_{m.o} := 0.05 \cdot m$ $\quad y_{m.o} := 0.20 \cdot m$ $\quad z_{m.o} := 0.00 \cdot m$ $\quad \alpha_{m.o} := 0.25 \cdot deg$

$$x_{m.sin_{n_{pri}}} := x_{m.a} \cdot \sin\left(2 \cdot \pi \cdot x_{m.f} \cdot n_{pri}\right) + x_{m.o}$$

$$y_{m.sin_{n_{pri}}} := y_{m.a} \cdot \sin\left(2 \cdot \pi \cdot y_{m.f} \cdot n_{pri}\right) + y_{m.o}$$

$$\alpha_{m.sin_{n_{pri}}} := \alpha_{m.a} \cdot \sin\left(2 \cdot \pi \cdot \alpha_{m.f} \cdot n_{pri}\right) + \alpha_{m.o}$$

$$z_{m.sin_{n_{pri}}} := z_{m.a} \cdot \sin\left(2 \cdot \pi \cdot z_{m.f} \cdot n_{pri}\right) + z_{m.o}$$

After running the simulation, the file `M1.ASC` was now imported into MathCAD and compared with the expected figures. The following MathCAD sheet cutting shows how the file was imported.

$x_{col} := 0$ $\quad y_{col} := 1$ $\quad z_{col} := 2$ $\quad \alpha_{col} := 3$ $\quad file_{m.sin} := READPRN(M1)$

$file_{x.m.sin_{n_{pri}}} := file_{m.sin_{n_{pri}}, x_{col}} \cdot m$ $\quad file_{y.m.sin_{n_{pri}}} := file_{m.sin_{n_{pri}}, y_{col}} \cdot m$

$file_{z.m.sin_{n_{pri}}} := file_{m.sin_{n_{pri}}, z_{col}} \cdot m$ $\quad file_{\alpha.m.sin_{n_{pri}}} := file_{m.sin_{n_{pri}}, \alpha_{col}} \cdot rad$

The first three columns of the imported file are now plotted and shown in Figure 4.1 on the following page. The amplitude axis is calibrated in metres. Consider the x-axis motion offsets plot - the amplitude of the waveform is 0.1 metres, and it is shifted up by 0.05 metres just as specified. Further, the frequency of the waveform was entered as 0.125 cycles per PRI, meaning one cycle per 8 PRIs, as evident from the plot.

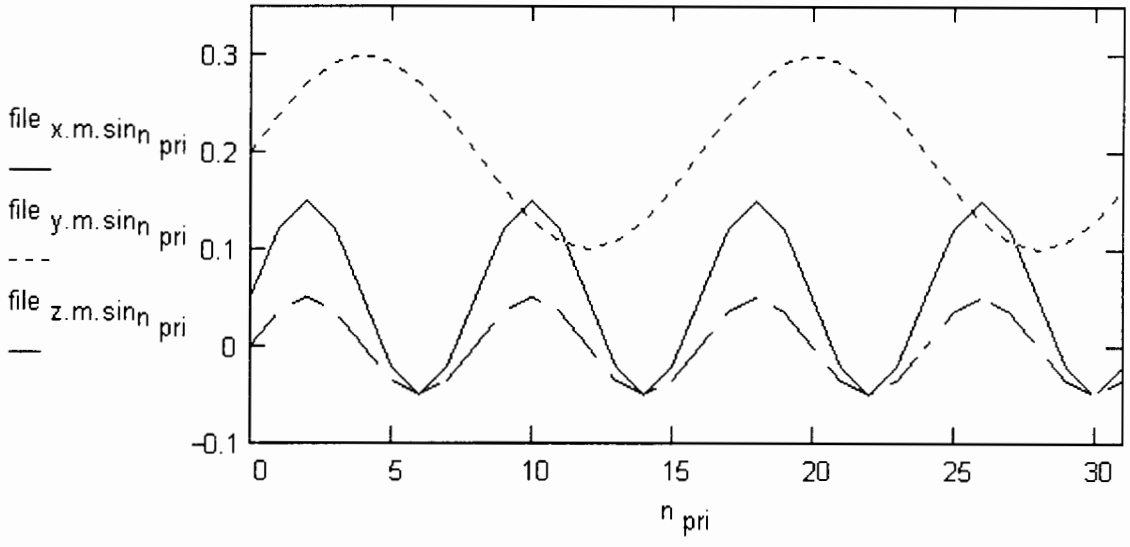

Figure 4.1: Simulated Sinusoidal Motion Offsets for the x-, y- and z- Axes

A more exact comparison is shown on the following page where the partial MathCAD sheet shows the actual values of the $\alpha_{m_{sin}}$ vector with the data from the fourth column of file `M1.ASC` alongside it. As the arrays are all symmetrical about their centres, only 16 of the 32 rows are given. As may be seen, the data agrees to four decimal places. Results are similar for the other three columns, however these are not shown here to conserve space.

In summary, this test shows that parameters entered under the `GENERAL` function can be successfully used by other functions that require them. In this case the value N_{pri} was passed to the `MOTIONSIN` function. Secondly, the `MOTIONSIN` function has been shown to generate the correct sinusoidal motion offset data based on the parameters passed to it. Finally, the `WRITE` function has been shown to successfully dump a motion offsets array to an ASCII file.

$\alpha_{m.sin} =$

	0
0	0.0044
1	0.0077
2	0.0105
3	0.0124
4	0.0131
5	0.0124
6	0.0105
7	0.0077
8	0.0044
9	0.0010
10	-0.0018
11	-0.0037
12	-0.0044
13	-0.0037
14	-0.0018
15	0.0010

·rad

file $\alpha.m.sin =$

	0
0	0.0044
1	0.0077
2	0.0105
3	0.0124
4	0.0131
5	0.0124
6	0.0105
7	0.0077
8	0.0044
9	0.0010
10	-0.0018
11	-0.0037
12	-0.0044
13	-0.0037
14	-0.0018
15	0.0010

·rad

4.3 Test of the MOTIONRND and ADD Functions

The command file used for this test is shown below:

```
! TEST2.CMD - Test of the MOTIONRND and ADD functions.
! ====================================================

$GENERAL A
        32    ! [ ]   Number of PRIs to simulate
     400.0    ! [Hz]  Pulse Repetition Frequency
       256    ! [ ]   Number of range samples to simulate
       0.2    ! [GHz] Simulation frequency

$MOTIONSIN M1 ! Create array M1 and place sinusoidal motion data in it
         A    ! [ ]          Use which GENERAL parameters ?
      0.10    ! [m]          Sine wave amplitude along x-axis
    0.1250    ! [cycles/PRI] Sine wave frequency along x-axis
      0.05    ! [m]          Sine wave "DC" offset along x-axis

    0.10 0.0625 0.20 ! Same for y-axis
    0.05 0.1250 0.00 ! Same for z-axis
    0.50 0.0625 0.25 ! Same for squint angle

$MOTIONRND M2 ! Create array M2 and place random motion data in it
         A    ! [ ]   Use which GENERAL parameters ?
       0.02   ! [m]   Maximum allowed +/- variation along x-axis
       0.01   ! [m]   Maximum allowed +/- variation along y-axis
       0.01   ! [m]   Maximum allowed +/- variation along z-axis
       0.10   ! [deg] Maximum allowed +/- variation in squint angle

$ADD M3       ! Create array M3 and place sum of the two input arrays in it
        M1    ! [ ] First array for addition
        M2    ! [ ] Second array for addition

$WRITE M1 ASC OVR M1.ASC

$WRITE M2 ASC OVR M2.ASC

$WRITE M3 ASC OVR M3.ASC
```

This simulation is an extension of the one in the previous section. After calling the **GENERAL** and **MOTIONSIN** functions, the **MOTIONRND** function is called, also with SGI 'A', to generate random motion offset data and store it in array **M2**. Then, the **ADD** function is called to add

the random motion offsets, M2, to the sinusoidal motion offsets, M1, and place the resulting motion offsets in array M3. Finally, the WRITE function is called three times - once for each of the M arrays generated - so as to dump them all to ASCII files. The MOTIONRND and ADD functions are described in Sections 3.5.3 and 3.5.5 respectively.

It is not possible to easily show a MathCAD sheet predicting the output of the MOTIONRND function due to the obvious and purposeful randomness of the results. Therefore, the following partial MathCAD sheet shows only how the three files, M1.ASC, M2.ASC and M3.ASC, were imported and how a few selected columns were extracted into separate arrays.

$N_{pri} := 32 \qquad n_{pri} := 0,1 .. N_{pri} - 1 \qquad x_{col} := 0 \qquad y_{col} := 1 \qquad z_{col} := 2 \qquad \alpha_{col} := 3$

$file_{m.sin} := READPRN(M1) \qquad file_{m.rnd} := READPRN(M2) \qquad file_{m.sum} := READPRN(M3)$

$file_{\alpha.m.sin_{n_{pri}}} := file_{m.sin_{n_{pri},\alpha_{col}}} \cdot rad \qquad file_{\alpha.m.rnd_{n_{pri}}} := file_{m.rnd_{n_{pri},\alpha_{col}}} \cdot rad$

$file_{\alpha.m.sum_{n_{pri}}} := file_{m.sum_{n_{pri},\alpha_{col}}} \cdot rad \qquad file_{x.m.rnd_{n_{pri}}} := file_{m.rnd_{n_{pri},x_{col}}} \cdot m$

A plot of the x-axis motion offsets from file M2.ASC is shown in Figure 4.2 on the following page. The units of magnitude are metres. As may be seen, the waveform consists of random points, each not above 0.02 metres or below -0.02 metres amplitude as specified in the command file.

It is a more simple matter to test the operation of the ADD function. The MathCAD sheet section on the following page shows 16 of the 32 elements of the squint angle column from files M1.ASC, M2.ASC and M3.ASC arranged in three columns alongside each other. As expected, the values in the third column are the sum of the values in the first two columns.

However, it may be seen that in some cases the last digit of the summed value is out by one. This is due to rounding as MathCAD has been configured to show floating point values to only four decimal places. Finally, note that all these values are in radians.

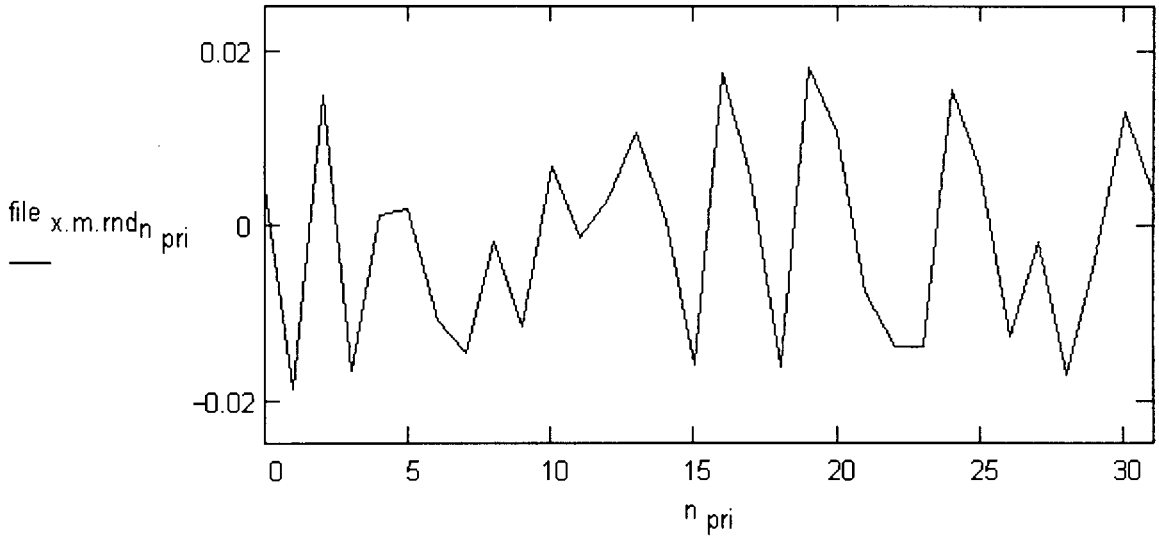

Figure 4.2: Simulated x-Axis Random Motion Offsets

file $_{x.m.sin}$ =

	0
0	0.0044
1	0.0077
2	0.0105
3	0.0124
4	0.0131
5	0.0124
6	0.0105
7	0.0077
8	0.0044
9	0.0010
10	-0.0018
11	-0.0037
12	-0.0044
13	-0.0037

file $_{x.m.rnd}$ =

	0
0	0.0002
1	-0.0014
2	-0.0004
3	0.0012
4	0.0014
5	0.0013
6	0.0017
7	0.0011
8	0.0012
9	-0.0007
10	-0.0010
11	0.0001
12	-0.0006
13	-0.0003

file $_{x.m.sum}$ =

	0
0	0.0045
1	0.0063
2	0.0101
3	0.0136
4	0.0145
5	0.0137
6	0.0122
7	0.0088
8	0.0055
9	0.0004
10	-0.0028
11	-0.0037
12	-0.0050
13	-0.0040

Finally, Figure 4.3 below shows a 32 point plot of the summed sinusoidal and random squint angle motion offsets as imported from file `M3.ASC`. The amplitude axis units are radians.

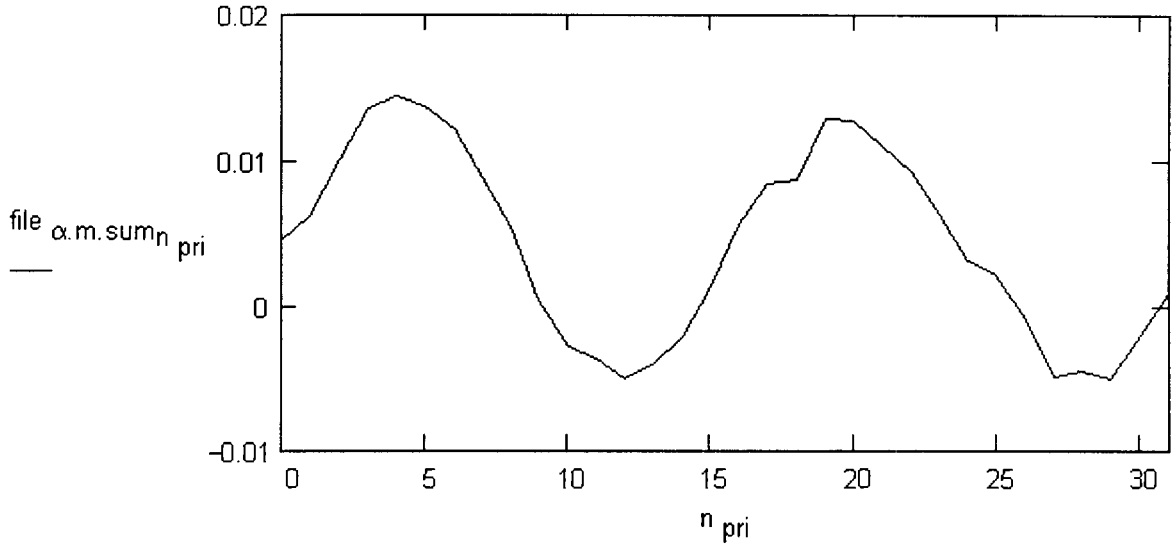

Figure 4.3: Result of Simulated Summation of Squint Angle Motion Offsets

In summary, this test shows that the `MOTIONRND` function produces random motion offset data limited to the specified amplitudes, and places it in an array of the correct format and length. The `ADD` function is also shown to add the two motion offset arrays correctly and produce a valid output array.

4.4 Test of the TARGET and GEOMETRY Functions

The command file used for this test is shown below:

```
! TEST3.CMD - Test of the TARGET and GEOMETRY functions.
! ======================================================

$GENERAL A
        32    ! [ ]   Number of PRIs to simulate
     400.0    ! [Hz]  Pulse Repetition Frequency
       256    ! [ ]   Number of range samples to simulate
       0.2    ! [GHz] Simulation frequency

$TARGET  A
       0.0    ! [m] x direction offset from mid path
    5000.0    ! [m] y ground range at closest approach
       0.0    ! [m] z height above ground level

$TARGET  A
     -10.0    ! [m] x direction offset from mid path
    5050.0    ! [m] y ground range at closest approach
      20.0    ! [m] z height above ground level

$GEOMETRY G1  ! Create array G1 and place simulation geometry data in it
         A    ! [ ]   Use which GENERAL parameters ?
      NONE    ! [ ]   Input array with aircraft offset motion data
    1500.0    ! [m]   Aircraft altitude above ground level
      80.0    ! [m/s] Aircraft velocity (groundspeed)

$WRITE G1 ASC OVR G1.ASC
```

In this simulation, the **GENERAL** function is as before. After that, two targets are defined for SGI 'A' by calling the **TARGET** function twice. The targets are positioned at $(0.0, 5000.0, 0.0)$ and $(-10.0, 5050.0, 20.0)$ metres taking the centre of the ideal flight path as the origin. Then, the **GEOMETRY** function is called so as to calculate the simulation geometry for the two predefined targets. The aircraft is set to be at 1500.0 metres above the ground plane, flying at 80 metres per second. There are no motion offsets and thus an ideal flight path is assumed. Finally, the **WRITE** function dumps the simulation geometry array **G1** to a plain ASCII file **G1.ASC**. The **TARGET** and **GEOMETRY** functions are described in Sections 3.5.2 and 3.5.6 respectively, and a theoretical background to the simulation geometry is given in

Section 2.4.

The partial MathCAD sheet below shows all the mathematical relations involved in the calculation of the simulation geometry. These are not complex and have been dealt with in the sections mentioned above as well as in the appropriate sections of Chapter 2. Recall here that the Cartesian co-ordinates of the targets as seen from the aircraft are converted to spherical co-ordinates. Further, the variable $x_{current}$ has been shortened to x_{cur}.

$x_{t.1} := 0.0 \cdot m \qquad y_{t.1} := 5000.0 \cdot m \qquad z_{t.1} := 0.0 \cdot m \qquad N_{pri} := 32$

$x_{t.2} := -10.0 \cdot m \qquad y_{t.2} := 5050.0 \cdot m \qquad z_{t.2} := 20.0 \cdot m \qquad n_{pri} := 0, 1 .. N_{pri} - 1$

$h := 1500.0 \cdot m \qquad f_{prf} := 400.0 \cdot Hz \qquad v := 80.0 \cdot m \cdot sec^{-1}$

$t_{pri} := \frac{1}{f_{prf}} \qquad t_{pri} = 0.0025 \cdot sec \qquad d_{pri} := v \cdot t_{pri} \qquad d_{pri} = 0.2000 \cdot m$

$d := (N_{pri} - 1) \cdot d_{pri} \qquad d = 6.2000 \cdot m \qquad x_{cur_{n_{pri}}} := -\frac{d}{2} + n_{pri} \cdot d_{pri}$

$x_a(x_t, x_{cur}) := x_t - x_{cur} \qquad y_a(y_t) := y_t \qquad z_a(z_t) := h - z_t$

$r_a(x_t, y_t, z_t, x_{cur}) := \sqrt{x_a(x_t, x_{cur})^2 + y_a(y_t)^2 + z_a(z_t)^2}$

$\alpha_{a.a}(x_t, y_t, x_{cur}) := \operatorname{atan}\left(\frac{x_a(x_t, x_{cur})}{y_a(y_t)}\right)$

$\alpha_{e.a}(x_t, y_t, z_t, x_{cur}) := -\operatorname{asin}\left(\frac{z_a(z_t)}{r_a(x_t, y_t, z_t, x_{cur})}\right)$

After running the simulation, the geometry file `G1.ASC` was now imported into MathCAD, as shown in the following MathCAD cutting.

$r_{col} := 0 \qquad a_{col} := 1 \qquad e_{col} := 2 \qquad file_{geom} := READPRN(G1)$

$file_{r.1_{n_{pri}}} := file_{geom_{2 \cdot n_{pri}, r_{col}}} \cdot m \qquad file_{r.2_{n_{pri}}} := file_{geom_{2 \cdot n_{pri}+1, r_{col}}} \cdot m$

$file_{a.1_{n_{pri}}} := file_{geom_{2 \cdot n_{pri}, a_{col}}} \cdot rad \qquad file_{a.2_{n_{pri}}} := file_{geom_{2 \cdot n_{pri}+1, a_{col}}} \cdot rad$

$file_{e.1_{n_{pri}}} := file_{geom_{2 \cdot n_{pri}, e_{col}}} \cdot rad \qquad file_{e.2_{n_{pri}}} := file_{geom_{2 \cdot n_{pri}+1, e_{col}}} \cdot rad$

Because of the vastly differing magnitudes, it is not possible to plot range, azimuth and elevation on one graph. Instead, the graph shown in Figure 4.4 on the following page shows

the familiar shape of the range curvature as seen by the moving radar for target **1**. The x-axis shows distance flown by the aircraft in metres with the total flight path distance centred around its middle. The y-axis represents range to target in metres.

As a simple check, the range to target 1 at closest approach should be $\sqrt{1500^2 + 5000^2} = 5220.15325$ metres, and indeed this is the range to target 1 when $x_{cur} = 0$ as shown in Figure 4.4 by the dotted line.

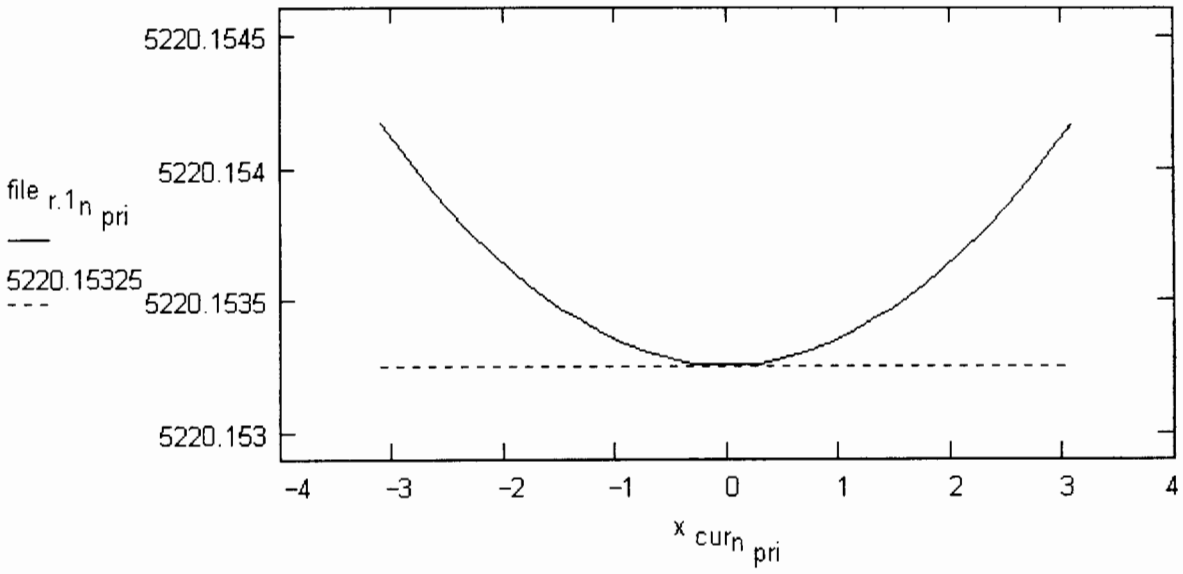

Figure 4.4: Simulated Range Curvature for Target 1

A more accurate comparison of expected and simulated values is shown in the partial MathCAD sheet on the following page. For a number of randomly chosen PRIs, the range, azimuth and elevation values imported from file `G1.ASC` are matched against those generated by MathCAD.

$$r_{a}\left(x_{t.1}, y_{t.1}, z_{t.1}, x_{cur_{12}}\right) = 5220.1533 \cdot m \qquad file_{r.1_{12}} = 5220.1533 \cdot m$$

$$r_{a}\left(x_{t.2}, y_{t.2}, z_{t.2}, x_{cur_{29}}\right) = 5262.4197 \cdot m \qquad file_{r.2_{29}} = 5262.4197 \cdot m$$

$$\alpha_{a.a}\left(x_{t.1}, y_{t.1}, x_{cur_{22}}\right) = -0.0003 \cdot rad \qquad file_{a.1_{22}} = -0.0003 \cdot rad$$

$$\alpha_{a.a}\left(x_{t.2}, y_{t.2}, x_{cur_{17}}\right) = -0.0020 \cdot rad \qquad file_{a.2_{17}} = -0.0020 \cdot rad$$

$$\alpha_{e.a}\left(x_{t.1}, y_{t.1}, z_{t.1}, x_{cur_{4}}\right) = -0.2915 \cdot rad \qquad file_{e.1_{4}} = -0.2915 \cdot rad$$

$$\alpha_{e.a}\left(x_{t.2}, y_{t.2}, z_{t.2}, x_{cur_{31}}\right) = -0.2851 \cdot rad \qquad file_{e.2_{31}} = -0.2851 \cdot rad$$

Although not shown here, tests were also performed with motion offset data being passed to the `GEOMETRY` function. The motion offsets were found to be correctly added to the simulated flight path, thus simulating weather conditions affecting the ideal path.

To summarise, this test shows how a number of targets may be specified and added to the chosen SGI by calling the `TARGET` function. Further, these targets are correctly processed by the `GEOMETRY` function which successfully simulates their positions with respect to the aircraft flight path, and converts their Cartesian representations to spherical co-ordinates. The resulting file `G1.ASC` is seen to be in the correct format with the rows grouped and ordered by the PRI number.

4.5 Test of the PULSEGEN Function

The command file used for this test is shown below:

```
! TEST4.CMD - Test of the PULSEGEN function.
! ==========================================

$GENERAL A
        32    ! [ ]   Number of PRIs to simulate
     400.0    ! [Hz]  Pulse Repetition Frequency
       256    ! [ ]   Number of range samples to simulate
       0.2    ! [GHz] Simulation frequency

$PULSEGEN T1 A1 ! Create arrays T1, A1 and place waveform data in them
         A    ! [ ]   Use which GENERAL parameters ?
       5.0    ! [ns]  Pulse rise time
      10.0    ! [ns]  Pulse fall time
     500.0    ! [ns]  Pulse width
       5.0    ! [V]   Peak amplitude
     CHIRP    ! [ ]   'CHIRP' or 'MONO' for chirp or monochromatic pulse
      0.05    ! [GHz] Chirp pulse bandwidth; ignored for monochromatic

$WRITE A1 ASC OVR A1.ASC

$WRITE T1 ASC OVR T1.ASC
```

The **GENERAL** function is as before, and is followed by a call to the function **PULSEGEN** which generates a pulse waveform both as an amplitude only array and as a time domain IQ pair array. The waveform is specified to have a rise time of 5 nanoseconds, a peak duration of 1 microsecond and a 10 nanosecond fall time thereafter. It should reach a peak amplitude of 5 volts. Further, the waveform is to be a chirp pulse, with the chirp bandwidth being 100 megahertz. As always, the **WRITE** function is called to write the two arrays, **A1** and **T1**, to ASCII files. The **PULSEGEN** function is described in Section 3.5.7, and a theoretical background to how pulses are generated by the simulator is given in Section 2.5.

The partial MathCAD sheet on the following page shows all the mathematical relations involved in the calculation of the chirp pulse envelope as well as its I and Q components. This is one of the more complex examples, and so it is especially important to refer to Section 2.5 where almost all of these equations are presented and explained. First, all the

constants are set and derived, and followed by the definition of a two and three input logical **AND** function due to a lack of one in MathCAD. Then, the pulse envelope, the pulse phase angle and the I and Q chirp pulse components are defined according to Equations 2.17, 2.21 and 2.24 respectively. Note that the symbol for the chirp pulse envelope, $p_{chirp_{env}}$, has been shortened to p_c.

$N_{bin} := 256$ $\quad n_{bin} := 0, 1 .. N_{bin} - 1$ $\quad f_{sim} := 200 \cdot MHz$ $\quad V_{peak} := 5.0 \cdot V$

$t_{rise} := 25.0 \cdot nsec$ $\quad t_{fall} := 50.0 \cdot nsec$ $\quad t_{length} := 500.0 \cdot nsec$ $\quad f_{BW} := 50 \cdot MHz$

$t_{bin} := \frac{1}{f_{sim}}$ $\quad t_{total} := \frac{N_{bin}}{f_{sim}}$ $\quad t_{pulse} := t_{rise} + t_{length} + t_{fall}$ $\quad t_{mid} := \frac{t_{pulse}}{2}$

$t_{width} := \frac{t_{rise}}{2} + t_{length} + \frac{t_{fall}}{2}$ $\quad N_r := \frac{t_{rise}}{t_{bin}}$ $\quad N_{rl} := \frac{t_{rise} + t_{length}}{t_{bin}}$

$N_{rlf} := \frac{t_{rise} + t_{length} + t_{fall}}{t_{bin}}$

$and2(a, b) := if(a \neq 0, if(b \neq 0, 1, 0), 0)$

$and3(a, b, c) := and2(a, and2(b, c))$

$t_{bin} = 5.0000 \cdot nsec$ $\quad t_{total} = 1280.0000 \cdot nsec$ $\quad t_{pulse} = 575.0000 \cdot nsec$ $\quad N_r = 5.0000$

$t_{mid} = 287.5000 \cdot nsec$ $\quad t_{width} = 537.5000 \cdot nsec$ $\quad N_{rl} = 105.0000$ $\quad N_{rlf} = 115.0000$

$$p_{c_{n_{bin}}} := if\left[and3\left(t_{rise} > 0.0 \cdot nsec, n_{bin} \geq 0, n_{bin} < N_r\right), V_{peak} \cdot \left(\frac{t_{bin}}{t_{rise}}\right) \cdot n_{bin}, 0 \cdot V\right] \ldots$$
$$+ if\left(and2\left(n_{bin} \geq N_r, n_{bin} < N_{rl}\right), V_{peak}, 0 \cdot V\right) \ldots$$
$$+ if\left[and3\left(t_{fall} > 0.0 \cdot nsec, n_{bin} \geq N_{rl}, n_{bin} < N_{rlf}\right), V_{peak} \cdot \left(\frac{t_{bin}}{t_{fall}}\right) \cdot \left(N_{rlf} - n_{bin}\right), 0 \cdot V\right] \ldots$$
$$+ if\left(and2\left(n_{bin} \geq N_{rlf}, n_{bin} < N_{bin}\right), 0 \cdot V, 0 \cdot V\right)$$

$$\phi_{chirpn_{bin}} := \pi \cdot f_{BW} \cdot \frac{\left(n_{bin} \cdot t_{bin} - t_{mid}\right)^2}{t_{width}}$$

$p_{chirp.I_{n_{bin}}} := p_{c_{n_{bin}}} \cdot \cos\left(\phi_{chirpn_{bin}}\right)$ $\quad p_{chirp.Q_{n_{bin}}} := p_{c_{n_{bin}}} \cdot \sin\left(\phi_{chirpn_{bin}}\right)$

The simulation was now run and the two files, amplitude only `A1.ASC` and time domain IQ pairs `T1.ASC`, were imported into MathCAD. The solitary column of file `A1.ASC` agreed with the expected pulse envelope waveform. Also, the I and Q components of file `T1.ASC` agreed with their theoretical counterparts. The extract of the MathCAD sheet below shows how the files `A1.ASC` and `T1.ASC` were imported, along with a number of randomly chosen

points from the expected and simulated arrays for the sake of comparison.

$\text{file}_{\text{ampl}} := \text{READPRN}(A1)\cdot V \qquad \text{file}_{\text{time}} := \text{READPRN}(T1)\cdot V \qquad I_{\text{col}} := 0 \qquad Q_{\text{col}} := 1$

$\text{file}_{a_{n_{\text{bin}}}} := \text{file}_{\text{ampl}_{n_{\text{bin}}}} \qquad \text{file}_{t.I_{n_{\text{bin}}}} := \text{file}_{\text{time}_{n_{\text{bin}},I_{\text{col}}}} \qquad \text{file}_{t.Q_{n_{\text{bin}}}} := \text{file}_{\text{time}_{n_{\text{bin}},Q_{\text{col}}}}$

$\text{file}_{t.I_{42}} = -0.9172\cdot V \qquad p_{\text{chirp}.I_{42}} = -0.9172\cdot V$

$\text{file}_{t.I_{108}} = 3.4177\cdot V \qquad p_{\text{chirp}.I_{108}} = 3.4176\cdot V$

$\text{file}_{t.Q_{42}} = 4.9152\cdot V \qquad p_{\text{chirp}.Q_{42}} = 4.9152\cdot V$

$\text{file}_{t.Q_{108}} = -0.7548\cdot V \qquad p_{\text{chirp}.Q_{108}} = -0.7548\cdot V$

$\sqrt{\left(\text{file}_{t.I_{42}}\right)^2 + \left(\text{file}_{t.Q_{42}}\right)^2} = 5.0000\cdot V \qquad \text{file}_{a_{42}} = 5.0000\cdot V \qquad p_{c_{42}} = 5.0000\cdot V$

$\sqrt{\left(\text{file}_{t.I_{108}}\right)^2 + \left(\text{file}_{t.Q_{108}}\right)^2} = 3.5000\cdot V \qquad \text{file}_{a_{108}} = 3.5000\cdot V \qquad p_{c_{108}} = 3.5000\cdot V$

Lastly, there are two figures on the following page. The first, Figure 4.5, shows the amplitude envelope of that same pulse as read from file `A1.ASC`. The second, Figure 4.6, shows a plot of the I and Q components of the time domain pulse as imported from file `T1.ASC`. In both cases only the first 128 of the 256 points are shown since the rest are all zero. The unit for the y-axes of both plots is the volt. As may be seen, the pulse envelope in Figure 4.6 correctly represents the specified rise, length and fall times. The shown pulse is 115 range bins long, which corresponds to the pulse length of 575 nanoseconds divided by the 5 nanosecond t_{bin}. Inspection of the data making up the I and Q plots in Figure 4.5 reveals that they are centred, as expected, around range bin 57.5, which corresponds to a time of $t_{pulse}/2$. The instantaneous frequency of these I and Q plots then increases linearly towards the sides.

In summary, this test shows how the radar pulse to be transmitted is specified by calling function `PULSEGEN` and that these specifications are correctly processed. Further, it is seen that the two arrays that are produced are stored and dumped in the correct format.

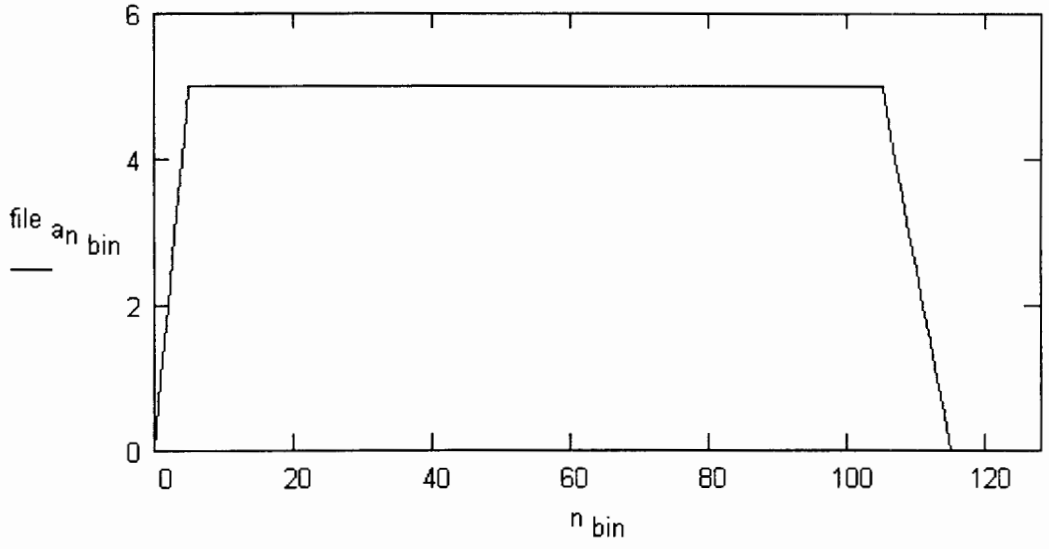

Figure 4.5: Envelope of Simulated Chirp Waveform

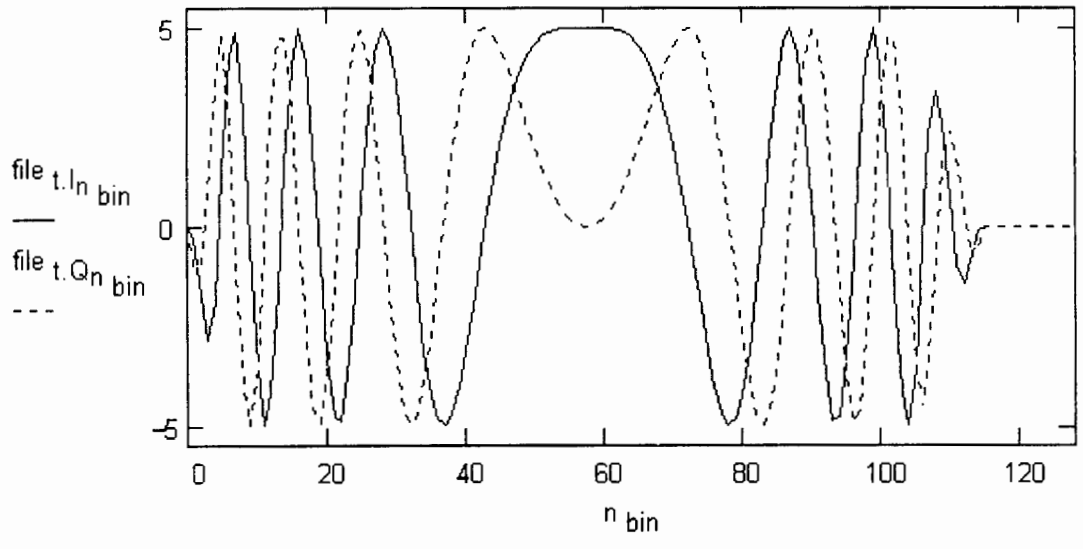

Figure 4.6: In-phase and Quadrature Components of Simulated Chirp Waveform

4.6 Test of the FFT and MATCHFILT Functions

The command file used for this test is shown below:

```
! TEST5.CMD - Test of the FFT and MATCHFILT functions.
! ====================================================

$GENERAL A
        32    ! [ ]   Number of PRIs to simulate
     400.0    ! [Hz]  Pulse Repetition Frequency
       256    ! [ ]   Number of range samples to simulate
       0.2    ! [GHz] Simulation frequency

$PULSEGEN T1 A1 ! Create arrays T1, A1 and place waveform data in them
         A    ! [ ]   Use which GENERAL parameters ?
      25.0    ! [ns]  Pulse rise time
      50.0    ! [ns]  Pulse fall time
     500.0    ! [ns]  Pulse width
       5.0    ! [V]   Peak amplitude
     CHIRP    ! [ ]   'CHIRP' or 'MONO' for chirp or monochromatic pulse
      0.05    ! [GHz] Chirp pulse bandwidth; ignored for monochromatic

$FFT F1       ! Create array F1 and place transformed array in it
        T1    ! [ ] Input waveform to be transformed

$MATCHFILT F2 ! Create array F2 and place matched filter array in it
        F1    ! [ ] Input waveform to be matched
      1.00    ! [ ] Hanning windowing factor ( 1.0 = no windowing )

$FFT T2       ! Create array T2 and place transformed array in it
        F2    ! [ ] Input waveform to be transformed

$WRITE T1 ASC OVR T1.ASC

$WRITE T2 ASC OVR T2.ASC

$WRITE F1 ASC OVR F1.ASC

$WRITE F2 ASC OVR F2.ASC
```

The **GENERAL** and **PULSEGEN** functions are as before. After the pulse is generated, the IQ waveform, **T1**, is transformed to the frequency domain by calling function **FFT** which generates array **F1**. This array is then passed to the **MATCHFILT** function which generates

the matched filter waveform, without performing Hanning windowing, and stores the result in array F2. This array is then transformed back into the time domain as array T2 by calling the FFT function again. The two time domain and two frequency domain arrays are then dumped to ASCII files by calls to the WRITE function. From a more theoretical viewpoint, the second time domain waveform, T2, should be a time reversed complex conjugate of the original waveform, T1, as per the ideal matched filter definition. At the same time the FFT algorithm is conveniently tested. The FFT and MATCHFILT functions are described in Sections 3.5.9 and 3.5.8 respectively, and a theoretical background to each one is given in Sections 2.7 and 2.8 respectively.

The partial MathCAD sheet on the following page shows how the four arrays were imported into MathCAD. Further, based on the imported array T1, MathCAD is used to calculate F1 by performing a complex FFT, then F2 by applying the complex conjugate function to F1, and finally T2 by inverse complex FFT of F2. Note that MathCAD automatically scales the complex FFT results, and so a cancelling scaling factor of N_{bin} is used in the forward transform. This allows for the use of the $1/N_{bin}$ factor on the inverse transform as discussed in Section 2.7. Note that the letters cx are an abbreviation for the word "complex".

$I_{col} := 0 \qquad Q_{col} := 1 \qquad N_{bin} := 256 \qquad n_{bin} := 0, 1 .. N_{bin} - 1$

$file_{t1} := READPRN(T1) \cdot V \qquad file_{t2} := READPRN(T2) \cdot V$

$file_{f1} := READPRN(F1) \cdot V \qquad file_{f2} := READPRN(F2) \cdot V$

$file_{t1.cx_{n_{bin}}} := file_{t1_{n_{bin}}, I_{col}} + i \cdot file_{t1_{n_{bin}}, Q_{col}}$

$file_{t2.cx_{n_{bin}}} := file_{t2_{n_{bin}}, I_{col}} + i \cdot file_{t2_{n_{bin}}, Q_{col}}$

$file_{f1.cx_{n_{bin}}} := file_{f1_{n_{bin}}, I_{col}} + i \cdot file_{f1_{n_{bin}}, Q_{col}}$

$file_{f2.cx_{n_{bin}}} := file_{f2_{n_{bin}}, I_{col}} + i \cdot file_{f2_{n_{bin}}, Q_{col}}$

$f1_{mcad} := N_{bin} \cdot CFFT(file_{t1.cx}) \qquad f2_{mcad} := \overline{f1_{mcad}} \qquad t2_{mcad} := \frac{1}{N_{bin}} \cdot ICFFT(f2_{mcad})$

On the following page, Figure 4.7 shows the magnitudes of the original radar chirp pulse as read from array T1 and the ideal matched filter as read from array T2. As expected, the matched filter is the time reversed copy of the original pulse. The magnitudes are in volts.

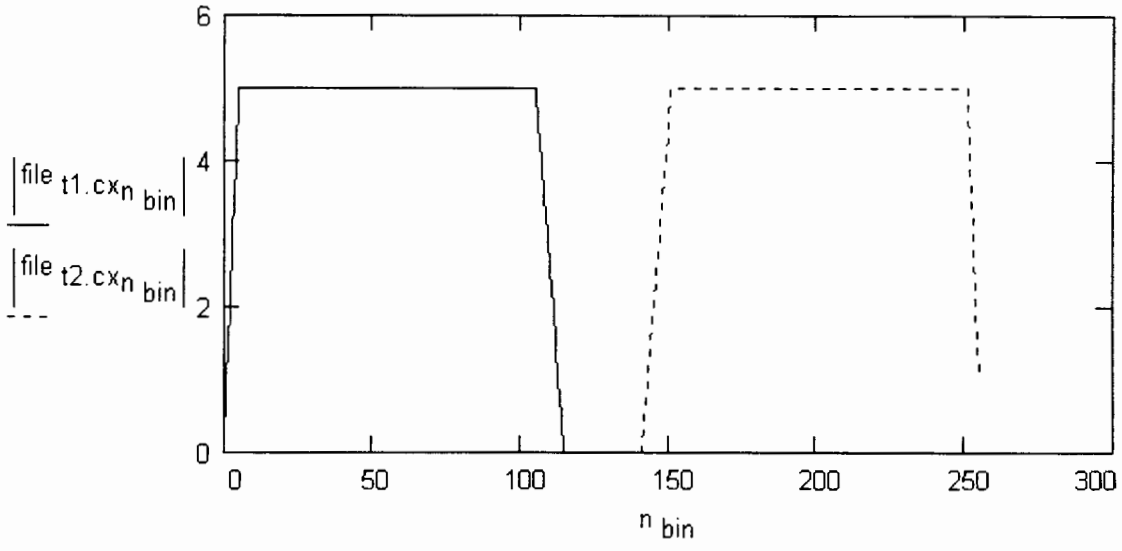

Figure 4.7: Overlaid Plots of Radar Chirp Pulse and Ideal Matched Filter

The MathCAD generated arrays `F1`, `F2` and `T2` were now compared with their imported RadSim array counterparts. The partial MathCAD below shows comparisons of the three arrays at two random points. It may be noticed that in two cases there is a difference of one in the fourth decimal place between the MathCAD predicted arrays and the ones generated by RadSim. This is a result of the MathCAD calculations being based on data from file `T1.ASC`, where the floating point values inside it have been rounded to five decimal places by the `WRITE` function. This tends to perpetuate rounding errors throughout the MathCAD calculation.

$\text{file}_{f1.cx_{15}} = 97.7185 + 12.7144i \cdot V$ $\quad$ $f1_{mcad_{15}} = 97.7185 + 12.7145i \cdot V$

$\text{file}_{f1.cx_{243}} = -108.7461 + 11.8414i \cdot V$ $\quad$ $f1_{mcad_{243}} = -108.7461 + 11.8414i \cdot V$

$\text{file}_{f2.cx_{15}} = 97.7185 - 12.7144i \cdot V$ $\quad$ $f2_{mcad_{15}} = 97.7185 - 12.7145i \cdot V$

$\text{file}_{f2.cx_{243}} = -108.7461 - 11.8414i \cdot V$ $\quad$ $f2_{mcad_{243}} = -108.7461 - 11.8414i \cdot V$

$\text{file}_{t2.cx_{15}} = 0.0000 \cdot V$ $\quad$ $t2_{mcad_{15}} = 0.0000 - 0.0000i \cdot V$

$\text{file}_{t2.cx_{243}} = -1.6230 - 4.7292i \cdot V$ $\quad$ $t2_{mcad_{243}} = -1.6230 - 4.7292i \cdot V$

To summarise, this test shows that the `FFT` function performs both forward and inverse transforms correctly. Further, the `MATCHFILT` function generates the required matched filter which, when transformed back to the time domain, is indeed the time reversed complex conjugate of the original radar pulse as expected. Although a test of Hanning windowing was not shown here, this feature was found to perform correctly under other test conditions.

4.7 Test of the AMPLIFY and MULTIPLY Functions

The command file used for this test is shown below:

```
! TEST6.CMD - Test of the AMPLIFY and MULTIPLY functions.
! ======================================================

$GENERAL A
        32    ! [ ]   Number of PRIs to simulate
     400.0    ! [Hz]  Pulse Repetition Frequency
       256    ! [ ]   Number of range samples to simulate
       0.2    ! [GHz] Simulation frequency

$PULSEGEN T1 A1 ! Create arrays T1, A1 and place waveform data in them
         A    ! [ ]   Use which GENERAL parameters ?
      25.0    ! [ns]  Pulse rise time
      50.0    ! [ns]  Pulse fall time
     500.0    ! [ns]  Pulse width
       5.0    ! [V]   Peak amplitude
     CHIRP    ! [ ]   'CHIRP' or 'MONO' for chirp or monochromatic pulse
      0.05    ! [GHz] Chirp pulse bandwidth; ignored for monochromatic

$FFT F1       ! Create array F1 and place transformed array in it
        T1    ! [ ] Input waveform to be transformed

$MATCHFILT F2 ! Create array F2 and place matched filter array in it
        F1    ! [ ] Input waveform to be matched
      1.00    ! [ ] Hanning windowing factor ( 1.0 = no windowing )

$AMPLIFY T3   ! Create array T3 and place amplified array in it
        T1    ! [ ] Input waveform to be amplified
     -40.0    ! [dB] Amplification factor

$FFT F3       ! Create array F3 and place transformed array in it
        T3    ! [ ] Input waveform to be transformed

$MULTIPLY F4  ! Create array F4 and place product in it
        F2    ! [ ] First input waveform to be multiplied
        F3    ! [ ] Second input waveform to be multiplied

$FFT T4       ! Create array T4 and place transformed array in it
        F4    ! [ ] Input waveform to be transformed

$WRITE T1 ASC OVR T1.ASC
```

```
$WRITE T3 ASC OVR T3.ASC

$WRITE T4 ASC OVR T4.ASC

$WRITE F2 ASC OVR F2.ASC

$WRITE F3 ASC OVR F3.ASC

$WRITE F4 ASC OVR F4.ASC
```

This test is an extension of the one presented in the previous section. The **GENERAL**, **PULSEGEN**, the first **FFT** and the **MATCHFILT** functions are all as before. At this point, array **T1** contains the original radar chirp pulse, **F1** contains the frequency domain representation of **T1**, and **F2** contains the frequency domain representation of the ideal matched filter to the radar chirp waveform. Then, the **AMPLIFY** function is called in order to scale the original radar chirp pulse by -40 decibels, that is a voltage gain factor of 0.01, and place the results in array **T3**. This scaled pulse is then transformed to the frequency domain by calling the **FFT** function which generates the array **F3**. The arrays **F2** and **F3** are now multiplied by calling the **MULTIPLY** function, which then stores the product in array **F4**. This waveform is then transformed into the time domain using the function **FFT**, resulting in array **T4**. Finally, the arrays **T1**, **T3**, **T4**, **F2**, **F3** and **F4** are written to plain ASCII files using the **WRITE** function. What is being done, is that the original radar chirp pulse is scaled down, transformed to the frequency domain and multiplied with the frequency domain representation of the matched filter for that pulse, which amounts to the convolution of the pulse with its matched filter. The result is then converted to the time domain. Since the original pulse is not exactly rectangular, but has rise and fall times which may be seen as triangles, the result of the convolution should be a narrow triangular looking response, with some $\sin(x)/x$ rounding and some visible sidelobes. This response should be centred about $n_{bin} = 0$ as the scaled pulse starts at that point. The **AMPLIFY** and and **MULTIPLY** functions are described in Sections 3.5.10 and 3.5.11 respectively, and a theoretical background to them both is given in Section 2.6.

The partial MathCAD sheet on the following page shows how all of the arrays were imported into MathCAD and their columns combined to form complex valued vectors. MathCAD is

then used to apply the gain factor of 0.01 to array T1, the radar chirp pulse, thus generating the equivalent of array T3 as read from file. This array is then transformed to the frequency domain and multiplied with the matched filter array F2. The method of multiplication was discussed in Section 2.6. The resulting vector, F4, is then transformed back to the time domain, the result thus being the equivalent of array T4 as read from file.

$I_{col} := 0 \qquad Q_{col} := 1 \qquad N_{bin} := 256 \qquad n_{bin} := 0, 1 .. N_{bin} - 1 \qquad G_a := 0.01$

$file_{t1} := READPRN(T1) \cdot V \qquad file_{t3} := READPRN(T3) \cdot V \qquad file_{t4} := READPRN(T4) \cdot V$

$file_{f2} := READPRN(F2) \cdot V \qquad file_{f3} := READPRN(F3) \cdot V \qquad file_{f4} := READPRN(F4) \cdot V$

$file_{t1.cx_{n_{bin}}} := file_{t1_{n_{bin}}, I_{col}} + i \cdot file_{t1_{n_{bin}}, Q_{col}}$

$file_{t3.cx_{n_{bin}}} := file_{t3_{n_{bin}}, I_{col}} + i \cdot file_{t3_{n_{bin}}, Q_{col}}$

$file_{t4.cx_{n_{bin}}} := file_{t4_{n_{bin}}, I_{col}} + i \cdot file_{t4_{n_{bin}}, Q_{col}}$

$file_{f2.cx_{n_{bin}}} := file_{f2_{n_{bin}}, I_{col}} + i \cdot file_{f2_{n_{bin}}, Q_{col}}$

$file_{f3.cx_{n_{bin}}} := file_{f3_{n_{bin}}, I_{col}} + i \cdot file_{f3_{n_{bin}}, Q_{col}}$

$file_{f4.cx_{n_{bin}}} := file_{f4_{n_{bin}}, I_{col}} + i \cdot file_{f4_{n_{bin}}, Q_{col}}$

$t3_{mcad_{n_{bin}}} := G_a \cdot file_{t1.cx_{n_{bin}}} \qquad f3_{mcad} := N_{bin} \cdot CFFT(t3_{mcad})$

$f4_{mcad_{n_{bin}}} := f3_{mcad_{n_{bin}}} \cdot file_{f2.cx_{n_{bin}}} \cdot \frac{1}{V} \qquad t4_{mcad} := \frac{1}{N_{bin}} \cdot ICFFT(f4_{mcad})$

The small section of a MathCAD sheet on the following page shows a number of comparisons. First, two random points of array T1 are compared with the same points of array T3. As may be seen, the values in array T3 have been correctly scaled by a factor of 0.01. Secondly, two random points of array F4 are compared with the same points of the same array that was generated with MathCAD. As may be seen, the simulation data and the MathCAD benchmark data agree. Note that the imaginary part of these values is zero, as it should be after an array has been multiplied by its complex conjugate. The final row shows that the values of F4 two rows above are indeed the result of the multiplication of elements in array F2 by corresponding elements in array F3.

$\text{file}_{t1.cx_{31}} = 2.0309 - 4.5690i \cdot V$ $\text{file}_{t3.cx_{31}} = 0.0203 - 0.0457i \cdot V$

$\text{file}_{t1.cx_{94}} = -4.7637 - 1.5190i \cdot V$ $\text{file}_{t3.cx_{94}} = -0.0476 - 0.0152i \cdot V$

$\text{file}_{f4.cx_{31}} = 27.8526 \cdot V$ $\text{file}_{f4.cx_{94}} = 0.0047 \cdot V$

$\text{file}_{f2.cx_{31}} \cdot \text{file}_{f3.cx_{31}} \cdot \frac{1}{V} = 27.8525 - 0.0000i \cdot V$ $\text{file}_{f2.cx_{94}} \cdot \text{file}_{f3.cx_{94}} \cdot \frac{1}{V} = 0.0047 + 0.0000i \cdot V$

$\text{f4}_{mcad_{31}} = 27.8526 + 0.0000i \cdot V$ $\text{f4}_{mcad_{94}} = 0.0047 + 0.0000i \cdot V$

Now, Figure 4.8 below shows a plot of array T3 which is the output of the AMPLIFY function with the original radar chirp pulse as the input. The amplitude axis units are volts. Recall that the original pulse peaked at 5 volts, while the one shown peaks at 0.05 volts. It may thus be seen that the input waveform was correctly scaled by a factor of 0.01 as specified in the command file. Only the first 128 range bins have been plotted as the pulse is at zero volts thereafter.

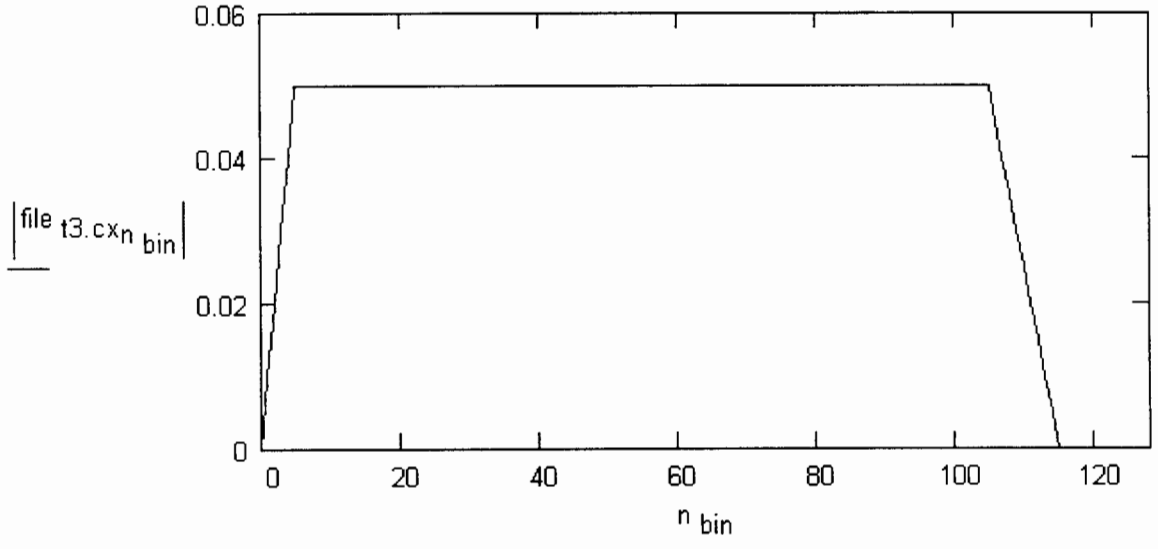

Figure 4.8: Radar Chirp Pulse After Scaling

Further, Figure 4.9 on the following page shows a plot of the array T4, also with volts as the amplitude units. This array is the result of the convolution of the scaled radar chirp pulse waveform, T3, and the matched filter waveform, T2. That is, T4 is the time domain

representation of the product of arrays F2 and F3. As expected, the shape of the mainlobe is that of a narrow triangle, with the $\sin(x)/x$ rounding and sidelobes caused by the triangular rise and fall shape of the radar pulse. The peak occurs at $n_{bin} = 0$ as predicted, with the waveform being symmetrical about this point. As we are not working with negative values of n_{bin} it may be seen that the left side of the waveform has been wrapped to the end of the array due to the cyclic nature of the discrete FFT.

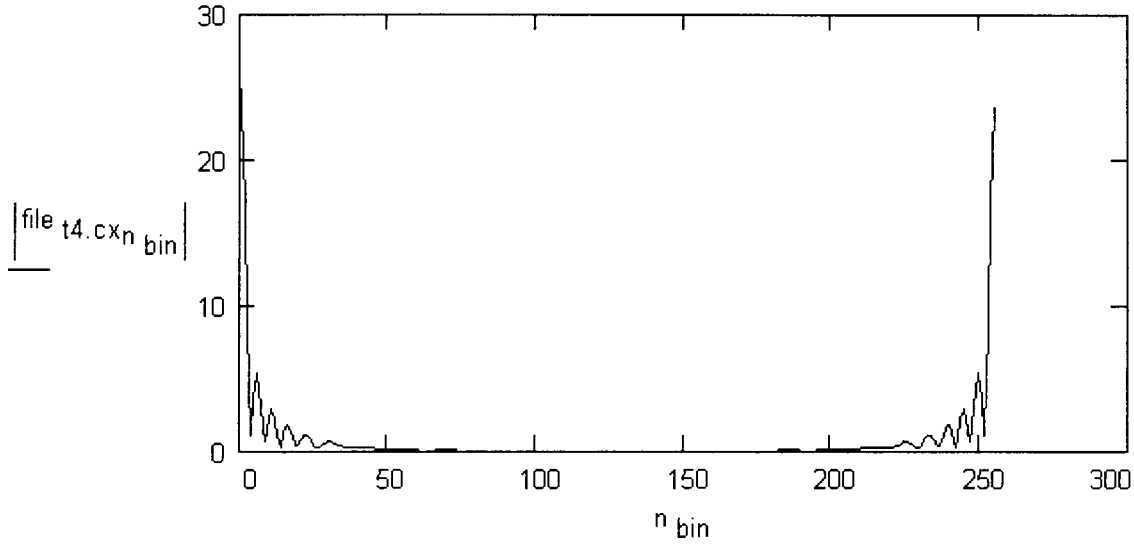

Figure 4.9: Result of Convolution of Radar Chirp Pulse and its Ideal Matched Filter

As a rough check of the peak amplitude value of the waveform shown in Figure 4.9, it is necessary to estimate the area underneath the product of the two original time domain waveforms at the time when they are most overlapped [21]. This maximum overlapping occurs when the 500 nanosecond main 'bodies' of the pulses are overlapping. At this time, there is also overlapping of the rise and fall curves, but it is limited to the length of the shorter of the two. In this case the shorter is the rise time of 25 nanoseconds. Now, 500 nanoseconds corresponds to 100 range bins, according to t_{bin} which was calculated to be 5 nanoseconds in Section 4.5. Similarly, 25 nanoseconds corresponds to 5 range bins. Finally, the amplitude of the pulses is 5 and 0.05 volts. Therefore, the area in question is calculated as follows:

$$5 \cdot 0.05 \cdot (5/2 + 100 + 5/2) = 26.2500\,.$$

Inspection of array T4 shows that the peak value is 26.2625 volts. The two results agree

very well taking into account that peak value in array `T4` is a result of intensive computation based on discrete data.

In summary, this test shows that the `AMPLIFY` function performs its constant gain or loss scaling correctly, and that the `MULTIPLY` function successfully performs element by element multiplication.

4.8 Test of the A2D Function

The command file used for this test is shown below:

```
! TEST7.CMD - Test of the A2D function.
! ====================================

$GENERAL A
        32    ! [ ]   Number of PRIs to simulate
     400.0    ! [Hz]  Pulse Repetition Frequency
       256    ! [ ]   Number of range samples to simulate
       0.2    ! [GHz] Simulation frequency

$PULSEGEN T1 A1 ! Create arrays T1, A1 and place waveform data in them
         A    ! [ ]   Use which GENERAL parameters ?
      25.0    ! [ns]  Pulse rise time
      50.0    ! [ns]  Pulse fall time
     500.0    ! [ns]  Pulse width
       5.0    ! [V]   Peak amplitude
     CHIRP    ! [ ]   'CHIRP' or 'MONO' for chirp or monochromatic pulse
      0.05    ! [GHz] Chirp pulse bandwidth; ignored for monochromatic

$A2D D1       ! Create array D1 and place digitised values in it
        T1    ! [ ]           Input waveform to be digitised
      0.05    ! [V]           LSB value
         8    ! [bits]        Number of bits of A2D precision
         0    ! [bins]        First bin to digitise ( inclusive )
       128    ! [bins]        Number of bins to digitise after first bin
         2    ! [bins/sample] Bin sampling rate

$WRITE D1 ASC OVR D1.ASC

$WRITE D1 BIN OVR D1.BIN

$WRITE T1 ASC OVR T1.ASC
```

This test uses the same calls to functions **GENERAL** and **PULSEGEN** as the tests in previous sections. This time however, the array **T1** is passed to the **A2D** function, the result of which is array **D1**. The analogue to digital converter is specified to be eight bit, with the least significant bit representing 0.05 volts. It is further required that only the first 128 bins are considered, and of these only alternate ones actually sampled, beginning with bin number 0. Finally, the **WRITE** function is called to dump array **T1** to an ASCII file and the digitised

waveform contained in array D1 to both ASCII and binary files. The A2D function is dealt with in Section 3.5.12 and a theoretical background is given in Section 2.10.

The partial MathCAD sheet on the following page shows how the files T1.ASC and D1.ASC were imported into MathCAD and their columns combined to form complex valued vectors. First however, it is important to note the definition of counter n_{dbin}, where 'dbin' is an abbreviation for *digitised range bin*. This counter runs from bin B_{start} and runs for the specified number of bins, B_{num}, divided by the given A2D sampling rate, S_{rate}. To convert from n_{dbin} to the usual n_{bin} multiply by S_{rate}. Secondly, note the definition of the DC offset applied to all digitised values. This offset is added so as to place a floating point 0.0 at the middle of the dynamic range of the A2D, which is $2^{N_{bits}-1} - 1 = 127$ as discussed in Section 3.5.12. Finally, MathCAD is used to digitise array T1 by applying Equation 2.46 from Section 2.10 to both the I and Q components. The DC offset is then added. Note that the values in array D1 are binary numbers converted to base 10, representing volts according to the value of V_{LSB}.

$$I_{col} := 0 \qquad Q_{col} := 1 \qquad N_{bin} := 256 \qquad n_{bin} := 0, 1 .. N_{bin} - 1 \qquad V_{LSB} := 0.05 \cdot V$$

$$B_{start} := 0 \qquad B_{num} := 128 \qquad S_{rate} := 2 \qquad N_{bits} := 8 \qquad DC_{offset} := 2^{N_{bits}-1} - 1$$

$$n_{dbin} := B_{start}, B_{start} + 1 .. \frac{B_{num} - B_{start}}{S_{rate}} - 1 \qquad DC_{offset.cx} := DC_{offset} + i \cdot DC_{offset}$$

$$file_{t1} := READPRN(T1) \cdot V \qquad file_{t1.cx_{n_{bin}}} := file_{t1_{n_{bin}, I_{col}}} + i \cdot file_{t1_{n_{bin}, Q_{col}}}$$

$$file_{d1} := READPRN(D1) \qquad file_{d1.cx_{n_{dbin}}} := file_{d1_{n_{dbin}, I_{col}}} + i \cdot file_{d1_{n_{dbin}, Q_{col}}}$$

$$a2d(float) := if\left(float \geq 0 \cdot V, floor\left(\frac{float}{V_{LSB}} + \frac{1}{2}\right), ceil\left(\frac{float}{V_{LSB}} - \frac{1}{2}\right)\right)$$

$$d1_{mcad_{n_{dbin}}} := a2d\left(file_{t1_{n_{dbin} \cdot S_{rate}, I_{col}}}\right) + i \cdot a2d\left(file_{t1_{n_{dbin} \cdot S_{rate}, Q_{col}}}\right) + DC_{offset.cx}$$

Figure 4.10 on the following page shows a plot of the magnitude of array D1. Note that the DC offset must be subtracted first so as to restore the original negative values so that the complex magnitude function returns meaningful results. As may be seen, the waveform peaks at value 100, which corresponds to the 5 volts of the original waveform divided by V_{LSB}. Secondly, the pulse ends at digitised bin 57, which is half of bin range 114. The

original waveform ended at range bin 115, but this bin is skipped since $S_{rate} = 2$. This shows that the A2D function did indeed subsample at the specified rate of S_{rate} bins per sample. Finally, the length of the waveform is 64 digitised bins, meaning that 128 range bins were processed before subsampling, as specified.

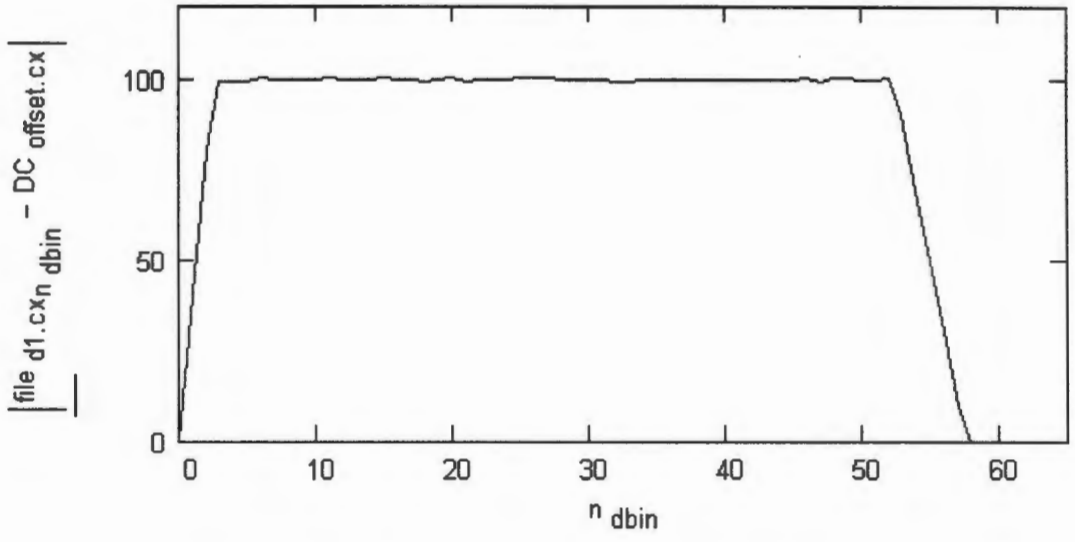

Figure 4.10: Digitised Version of Radar Chirp Pulse Waveform

The small section of a MathCAD sheet on the following page shows a number of comparisons. First, it is shown that the complex value at $n_{dbin} = 5$ of the imported array `D1` agrees with that predicted by MathCAD calculation. On the right hand side, the complex value of array `T1` at $n_{bin} = 5 * S_{rate}$, divided by V_{LSB} and with the DC offset added is shown. As may be seen, if the I and Q components are rounded off, the result will equal the one calculated by function `A2D`. This exercise is then repeated, but for $n_{dbin} = 40$.

Now, in the binary file `D1.BIN`, its twelfth and thirteenth characters correspond to the I and Q components of digitised bin 5 respectively. These components are represented in the file by the ASCII characters '8' and '9' which have decimal ASCII table values of 56 and 57 respectively. As seen in the MathCAD sheet cutting on this page, the value of digitised bin 5 should be $56 + 57i$. This confirms that the digitised values are correctly dumped to a binary file by the `WRITE` simulation function.

$$d1_{mcad_5} = 56.0000 + 57.0000i \qquad \frac{file_{t1.cx_{5 \cdot s_{rate}}}}{V_{LSB}} + DC_{offset.cx} = 55.6466 + 56.9380i$$

$$file_{d1.cx_5} = 56.0000 + 57.0000i$$

$$d1_{mcad_{40}} = 42.0000 + 74.0000i \qquad \frac{file_{t1.cx_{40 \cdot s_{rate}}}}{V_{LSB}} + DC_{offset.cx} = 42.1200 + 74.1286i$$

$$file_{d1.cx_{40}} = 42.0000 + 74.0000i$$

To summarise, this test shows that the **A2D** function performs digitisation of the input waveform correctly, and in accordance with all the specified parameters. Further, the **WRITE** function is seen to correctly dump binary digitised values to file.

4.9 Test of the RETURNL and ENDLOOP Functions

The command file used for this test is shown below:

```
! TEST8.CMD - Test of the RETURNL and ENDLOOP functions.
! ======================================================

$GENERAL A
        32    ! [ ]   Number of PRIs to simulate
     400.0    ! [Hz]  Pulse Repetition Frequency
       256    ! [ ]   Number of range samples to simulate
       0.2    ! [GHz] Simulation frequency

$TARGET  A
       0.0    ! [m] x direction offset from mid path
    5000.0    ! [m] y ground range at closest approach
       0.0    ! [m] z height above ground level

$GEOMETRY G2  ! Create array G2 and place simulation geometry data in it
         A    ! [ ]   Use which GENERAL parameters ?
      NONE    ! [ ]   Input array with aircraft offset motion data
    1500.0    ! [m]   Aircraft altitude above ground level
      80.0    ! [m/s] Aircraft velocity (groundspeed)

$PULSEGEN T1 A1 ! Create arrays T1, A1 and place waveform data in them
         A    ! [ ]   Use which GENERAL parameters ?
      25.0    ! [ns]  Pulse rise time
      50.0    ! [ns]  Pulse fall time
     500.0    ! [ns]  Pulse width
       5.0    ! [V]   Peak amplitude
     CHIRP    ! [ ]   'CHIRP' or 'MONO' for chirp or monochromatic pulse
      0.05    ! [GHz] Chirp pulse bandwidth; ignored for monochromatic

$RETURNL T5   ! Create array T5 and place waveform radar return in it
        G2    ! [ ]   Simulation geometry array to be used
        A1    ! [ ]   Envelope of pulse to be transceived
       1.0    ! [GHz] Radar centre frequency
      90.0    ! [dB]  Antenna gains
       5.0    ! [dB]  Losses ( TxLoss + RxLoss + PropagationLoss )
       1.0    ! [m^2] Target cross-sectional area
    5200.0    ! [m]   Range offset for simulation geometry
     CHIRP    ! [ ]   Pulse type - 'CHIRP' or 'MONO'
      0.05    ! [GHz] Chirp bandwidth (arb non-zero for MONO)
      NULL    ! [ ]   Antenna type - 'SIN' or 'NULL'
```

```
        0.0     ! [deg] Antenna elevation beamwidth
        0.0     ! [deg] Antenna azimuth beamwidth
        0.0     ! [deg] Antenna elevation angle (down is negative)
        0.0     ! [deg] Antenna squint angle (backward is positive)

$WRITE T5 ASC APP T5.ASC

$ENDLOOP        ! Loop ends, re-trace back to the last RETURNL call

$WRITE G2 ASC OVR G2.ASC
```

This is the final functionality test, and it constitutes a full SAR simulation, albeit the most simple one possible. The calls to functions **GENERAL**, **TARGET**, **GEOMETRY** and **PULSEGEN** are the same as those used throughout previous sections, with the exception that only one target has been specified and not two as before. The **RETURNL** function is then called with the simulation geometry and radar chirp pulse envelope arrays, **G2** and **A1** respectively as inputs. The output is the time domain array **T5** which contains the original radar chirp pulse which has been scaled down, delayed in time and modified in phase angle. Further, the radar centre frequency is specified to be 1.0 gigahertz, the antenna gains are 90 decibels and the losses are 5 decibels. Note that the antenna gain has been exaggerated because in a standard simulation the waveform to be transmitted would be amplified first with a call to the **AMPLIFY** function, and amplified again after its return. This is not done here for sake of simplicity. Each point target is to be seen as an object with a radar cross-section of 1 metre squared. The range offset, R_o, as discussed in Section 3.5.13, is set to 5200 metres. This figure is based on the range to the single target at closest approach, which is $\sqrt{1500^2 + 5000^2} = 5220.1533$ metres. The radar pulse that is regenerated within the function is to be a chirp pulse with a bandwidth of 50 megahertz - the same as the pulse generated by **PULSEGEN**. The antenna is specified to be omnidirectional. The waveform contained in array **T5** is then dumped to an ASCII file with a call to function **WRITE**. Note that each successive call will append the contents of array **T5** to the end of the existing file. The **ENDLOOP** function is now called so as to return simulation flow back to the **RETURNL** function. The loop should execute 32 times, once for each of the 32 PRIs specified in the **GENERAL** function. Finally, array **G2** is written to an ASCII file with another call to the **WRITE** function. The **RETURNL** and **ENDLOOP** functions are dealt with in Sections 3.5.13 and 3.5.14 respectively and a theoretical background to the calculation of the return pulse is

given in Section 2.9.

The usual introductory section of a MathCAD sheet has been divided into two parts due to the complexity of this test. The first part, shown below, illustrates the set up of the n_{bin} and n_{pri} counters. Since there are N_{pri} radar return waveforms appended together in array T5, and since each one is N_{bin} range bins long, a new counter, n_{total}, is defined to count through the entire array. The I and Q columns of the array are combined to form a complex vector. At the same time the array is split up into its constituent radar return waveforms, resulting in a two dimensional array with N_{pri} columns and N_{bin} rows. The magnitude of each complex value in this array is then calculated and stored in a new array with the 'mag' subscript. Similarly, the relative phase angles are calculated and stored in a new array with the 'arg' subscript. Note that since the antenna is omnidirectional, only the first column of array G2, range to target, is extracted from the imported simulation geometry file. The other two columns, elevation to target and azimuth to target do not affect matters in any way.

$$I_{col} := 0 \qquad Q_{col} := 1 \qquad R_{col} := 0 \qquad N_{bin} := 256 \qquad n_{bin} := 0,1 .. N_{bin} - 1$$

$$N_{pri} := 32 \qquad n_{pri} := 0,1 .. N_{pri} - 1 \qquad n_{total} := 0,1 .. N_{pri} \cdot N_{bin} - 1$$

$$file_{t5} := READPRN(T5) \cdot V \qquad file_{g2} := READPRN(G2) \qquad file_{r.t\,n_{pri}} := file_{g2\,n_{pri}, R_{col}} \cdot m$$

$$file_{t5.cx\,n_{bin}, n_{pri}} := file_{t5(n_{pri} \cdot N_{bin}) + n_{bin}, I_{col}} + i \cdot file_{t5(n_{pri} \cdot N_{bin}) + n_{bin}, Q_{col}}$$

$$file_{t5.mag\,n_{bin}, n_{pri}} := \left| file_{t5.cx\,n_{bin}, n_{pri}} \right|$$

$$file_{t5.arg\,n_{bin}, n_{pri}} := if\left(file_{t5.mag\,n_{bin}, n_{pri}} \neq 0 \cdot V, \arg\left(file_{t5.cx\,n_{bin}, n_{pri}} \right), 0 \right)$$

The second part of the MathCAD sheet is shown on the following page. First, the gains and losses are converted from decibels to unitless factors. Then, the system scaling factor, S_{sc}, is defined according to Equation 2.38. Moving onto the phase shift, t_{delay} and N_{delay} are defined according to Equations 2.32 and 2.33. This is followed by the actual phase shift definition as specified in Equation 2.43. The original radar chirp waveform phase angle is defined. The sheet ends with the definition of the resultant phase angle of the return waveform at any range at any range bin, here called ϕ_{total}.

$$G_{ant} := 10^{\frac{90.0}{10}} \qquad L_{txrx} := 10^{\frac{5.0}{10}} \qquad R_o := 5200.0 \cdot m \qquad a_{rcs.t} := 1.0 \cdot m^2$$

$$c := 3 \cdot 10^8 \cdot m \cdot sec^{-1} \qquad f_{centre} := 1.0 \cdot GHz \qquad f_{sim} := 200 \cdot MHz \qquad f_{BW} := 50 \cdot MHz$$

$$\lambda_{centre} := \frac{c}{f_{centre}} \qquad t_{rise} := 25.0 \cdot nsec$$

$$S_{sc}(r_t) := \sqrt{\frac{G_{ant}^2 \cdot \lambda_{centre}^2 \cdot a_{rcs.t}}{(4 \cdot \pi)^3 \cdot r_t^4 \cdot L_{txrx}}} \qquad t_{fall} := 50.0 \cdot nsec$$

$$t_{bin} := \frac{1}{f_{sim}} \qquad t_{length} := 500.0 \cdot nsec$$

$$t_{pulse} := t_{rise} + t_{length} + t_{fall} \qquad t_{width} := \frac{t_{rise}}{2} + t_{length} + \frac{t_{fall}}{2} \qquad t_{mid} := \frac{t_{pulse}}{2}$$

$$t_{delay}(r_t) := \frac{2 \cdot r_t}{c} \qquad N_{delay}(r_t) := \frac{t_{delay}(r_t)}{t_{bin}}$$

$$\phi_{delay}(r_t) := 2 \cdot \pi \cdot f_{centre} \cdot t_{delay}(r_t) \cdot rad \qquad \phi_{chirp}(n_{bin}) := \pi \cdot f_{BW} \left[\frac{(n_{bin} \cdot t_{bin} - t_{mid})^2}{t_{width}} \right] \cdot rad$$

$$\phi_{total}(n_{bin}, r_t) := \arg\left(\cos\left(\phi_{chirp}(n_{bin}) - \phi_{delay}(r_t)\right) + i \cdot \sin\left(\phi_{chirp}(n_{bin}) - \phi_{delay}(r_t)\right)\right)$$

Figure 4.11 on the following page shows a full 256 point plot of the magnitude of the return waveform for a PRI number 7. As may be seen, the waveform is the same as the original pulse, but scaled in amplitude and delayed in time. According to the MathCAD sheet, the range to the single target at PRI 7 is 5220.1535, and for this range the value of A_V is 0.1390. Multiplying the amplitude of the original pulse, 5 volts, by S_{sc} gives 0.6949 volts as being the amplitude of the returned pulse. Inspection of the data making up the plot of Figure 4.11 shows that the peak amplitude is indeed 0.6949 volts. Further, the value of N_{delay} for the above mentioned range less R_o is calculated by MathCAD to be 27 range bins. Inspection of the data making up the plot of Figure 4.11 shows that the first point of the original pulse has been shifted from range bin 0 to range bin 27 as expected.

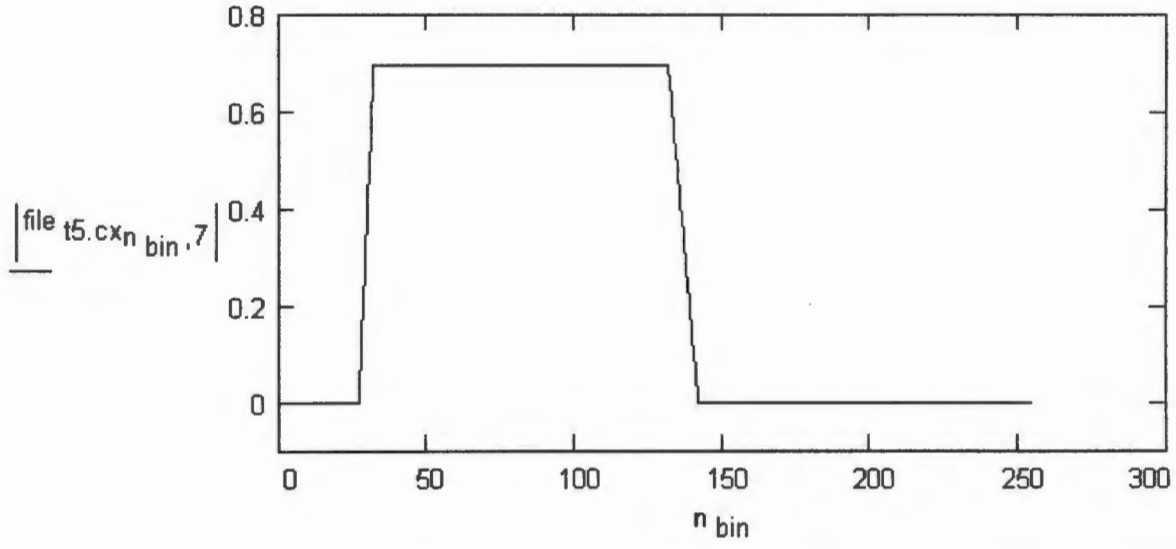

Figure 4.11: Magnitude of Return Pulse at PRI 7

This section ends off with another partial MathCAD sheet which may be found on the following page. The sheet shows a number of comparisons between data generated by RadSim and that generated by MathCAD. The first section deals with the return waveform at PRI number 7. Amongst others, the most important values shown are the range to the target, the value of S_{sc} for that range and N_{delay}. The second section compares the phase angle of the return waveform at PRI 7 and range bin 101 with the phase angle predicted by MathCAD. Similar comparisons are shown for two other PRIs. It may be seen that the MathCAD predicted phase angle is out by 0.0001 radians in one case. This is due to the range to target parameter passed to the total phase calculation function already being rounded off by the `WRITE` function when dumped to an ASCII file. The phase angle values calculated from IQ data generated by RadSim are more accurate since RadSim used the full precision of the range to target for its calculations.

$$\text{file}_{r.t_7} = 5220.1535 \cdot m \qquad S_{sc}\left(\text{file}_{r.t_7}\right) = 0.1390 \qquad S_{sc}\left(\text{file}_{r.t_7}\right) \cdot 5 \cdot V = 0.6949 \cdot V$$

$$\text{floor}\left(N_{delay}\left(\text{file}_{r.t_7} - R_0\right) + 0.5\right) = 27.0000 \qquad \left|\text{file}_{t5.cx_{101,7}}\right| = 0.6949 \cdot V$$

$$\text{file}_{t5.arg_{101,7}} = -0.2532 \cdot rad \qquad \phi_{total}\left(101 - 27, \text{file}_{r.t_7} - R_0\right) = -0.2532 \cdot rad$$

$$\text{file}_{t5.arg_{63,19}} = 1.1445 \cdot rad \qquad \phi_{total}\left(63 - 27, \text{file}_{r.t_{19}} - R_0\right) = 1.1446 \cdot rad$$

$$\text{file}_{t5.arg_{44,30}} = -2.8471 \cdot rad \qquad \phi_{total}\left(44 - 27, \text{file}_{r.t_{30}} - R_0\right) = -2.8471 \cdot rad$$

In summary, this test shows that the **RETURNL** function correctly calculates the magnitude, phase angle and delay of a return waveform based on the input pulse envelope and the simulation geometry together with all the other parameters. Also, the **ENDLOOP** function is seen to successfully return simulation flow to the **RETURNL** function until return waveforms have been calculated for all required PRIs.

4.10 Portability Tests

One of the requirements when implementing RadSim was that it must conform to the ANSI C standard so that it may be compiled and executed on various platforms that support this standard. This section aims to discuss these issues and present the results of the portability tests that were performed.

First, the ANSI C standard [2] was studied before implementation began and strictly adhered to during the coding stages. A number of other documents dealing with portability and good programming practice were studied, most notably the Frequently Asked Questions (FAQ) document of the `comp.lang.c` Internet news group [4] and "Programming in C++, Rules and Recommendations" by Nyquist *et al* [18]. Further, the tool **lint**[1] was obtained and used throughout every stage of development to ensure pure ANSI C code. The effort was a worthwhile one since after completion of the simulator, the code was successfully compiled and executed on every platform that was available for such testing, thereby satisfying the portability requirement.

The results are presented in the following subsections. Note that the compiler is a more important portability issue than the platform it is running on, as any compiler claiming ANSI C compatibility will hide the underlying differences between the various hardware platforms. Therefore the subsections have been divided according to compilers rather than hardware platforms.

At this point, thanks go to Mr A. Langman of the RRSG [15] for his help with GNU C installation and compilation on MS-DOS, Linux and FreeBSD. Also, thanks go to Mr G. Zsilavecz of Telkom [23] for his help with HP cc and HP CC compilation on HP UNIX machines under HPUX 9 and HPUX 10.

[1] lint is a software package that parses raw C files, checking them for possible errors, ambiguities, and calls and constructs not supported by the ANSI C standard. This tool is usually deemed to be superior to syntax parsers that form part of the generally available C and C++ compilers. The specific package used in this case was pclint.zip available as shareware from the more famous anonymous ftp sites. All information and opinion about lint was obtained from active members of the `comp.lang.c` Internet news group [5].

4.10.1 Borland C++ 3.1

Because of its excellent development interface, this compiler was used to develop the software on a 486 IBM compatible PC under the MSDOS 6.2 operating system. The software was compiled and executed on this platform successfully. However, the 64 kilobyte limit on array lengths implies that, for example, an array in the time domain may only consist of 8192 points[2]. This is large enough for testing the simulator, but certainly too limited for a serious simulation. This is a shortcoming of the operating system rather than the software or the compiler.

4.10.2 Borland C++ 4.02

This compiler appeared on the market towards the end of the development and testing stages. Tests were performed on a 486 IBM compatible PC under the MSDOS 6.2, Windows 3.1 and recently Windows 95 operating systems. In all cases compilation and execution was successful. The 64 kilobyte array limit was still a factor since there was no way to overcome it and still remain within the ANSI C standard. However, recent developments in true 32 bit programming with Borland C++ 5.0, and even more recently Borland C++ Builder, should finally alleviate the legacy memory limitation.

4.10.3 GNU C

GNU C is a shareware compiler not affiliated to any specific hardware platform as versions are freely available for almost all currently existing platforms. Tests were performed using various GNU C versions installed on a 486 IBM compatible PC under the MSDOS 6.2, Linux and FreeBSD operating systems, and on an HP server under the HPUX 9 and HPUX 10 operating systems. In all cases compilation and execution was successful.

[2] A floating point value on this platform is four bytes long, and each point consists of a complex pair.

4.10.4 GNU C++

As with GNU C, the tests were performed on the same range of hardware and in all cases compilation and execution was successful.

4.10.5 cc

This compiler is the standard C compiler for dedicated UNIX hardware. Tests were performed using cc under HPUX 9 and HPUX 10 on an HP server, and under SunOS System 5 on a SUN server. Once again, compilation and execution was successful in all cases.

4.10.6 CC

This is the C++ version of the cc compiler. Tests were performed under HPUX 9 and HPUX 10 on an HP server. Compilation and execution was successful in both cases.

Chapter 5

Sample Simulations

The aim of this chapter is to show RadSim involved in four realistic simulations which will attempt to illustrate just how RadSim would be used by members of the RRSG. For each simulation, the output generated by RadSim is processed and analysed by Horrell's azimuth compression software [9]. This shows the interaction of the simulator and the azimuth processor, as well as the intermediate and final images that are obtained.

Before going on to the main body of this chapter, the four simulations are introduced in the following section. Also, the presentation layout of each simulation is addressed.

5.1 Introduction to the Simulations

Of the four simulations presented in this chapter, the first is the simplest with only one point target, no aircraft motion errors and an omnidirectional antenna. The second simulation builds on the first one with the addition of both random and sinusoidal aircraft motion errors. The third simulation introduces two additional point targets as well as an amplification stage. Finally, the fourth simulation is the same as the third, but this time a directional antenna with a sinusoidal gain pattern is specified. Unless specifically noted otherwise in the text, in all cases it has been attempted to use realistic values for simulation function parameters.

Each simulation is presented in a separate section. Every such section begins with a listing of the RadSim command file that was used, together with a discussion thereof. This is followed by one or more intermediate images showing the magnitude component of the raw binary radar return waveform file as generated by RadSim. These images are then analysed to varying degrees. After this, one or more final images are shown - that is, the images obtained after full azimuth compression. The shown images are then analysed and some conclusions are drawn.

As in Chapter 4, each section begins on a new page so as to present most of the RadSim command file on one page for ease of readability.

5.2 Simple Simulation of One Point Target

The command file used for this simulation is shown below:

```
! SIM1.CMD - Simple Simulation of One Point Target.
! ================================================

$GENERAL A
      4000    ! [ ]   Number of PRIs to simulate
     400.0    ! [Hz]  Pulse Repetition Frequency
       256    ! [ ]   Number of range samples to simulate
       0.2    ! [GHz] Simulation frequency

$TARGET  A
       0.0    ! [m] x direction offset from mid path
    5000.0    ! [m] y ground range at closest approach
       0.0    ! [m] z height above ground level

$GEOMETRY G1  ! Create array G1 and place simulation geometry data in it
         A    ! [ ]   Use which GENERAL parameters ?
      NONE    ! [ ]   Input array with aircraft offset motion data
    1500.0    ! [m]   Aircraft altitude above ground level
      80.0    ! [m/s] Aircraft velocity (groundspeed)

$PULSEGEN T1 A1 ! Create arrays T1, A1 and place waveform data in them
         A    ! [ ]   Use which GENERAL parameters ?
       5.0    ! [ns]  Pulse rise time
      10.0    ! [ns]  Pulse fall time
    1000.0    ! [ns]  Pulse width
       5.0    ! [V]   Peak amplitude
     CHIRP    ! [ ]   'CHIRP' or 'MONO' for chirp or monochromatic pulse
      0.10    ! [GHz] Chirp pulse bandwidth; ignored for monochromatic

$FFT F1       ! Create array F1 and place transformed array in it
        T1    ! [ ] Input waveform to be transformed

$MATCHFILT F2 ! Create array F2 and place matched filter array in it
        F1    ! [ ] Input waveform to be matched
       1.0    ! [ ] Hanning windowing factor ( 1.0 = no windowing )

$RETURNL T2   ! Create array T2 and place waveform radar return in it
        G1    ! [ ]   Simulation geometry array to be used
        A1    ! [ ]   Envelope of pulse to be transceived
       1.0    ! [GHz] Radar centre frequency
      80.0    ! [dB]  Antenna gains
       5.0    ! [dB]  Losses ( TxLoss + RxLoss + PropagationLoss )
       1.0    ! [m^2] Target cross-sectional area
    5200.0    ! [m]   Range offset for simulation geometry
     CHIRP    ! [ ]   Pulse type - 'CHIRP' or 'MONO'
      0.10    ! [GHz] Chirp bandwidth (arb non-zero for MONO)
      NULL    ! [ ]   Antenna type - 'SIN' or 'NULL'
       0.0    ! [deg] Antenna elevation beamwidth
```

```
        0.0     ! [deg] Antenna azimuth beamwidth
        0.0     ! [deg] Antenna elevation angle (down is negative)
        0.0     ! [deg] Antenna squint angle (backward is positive)

$FFT F3         ! Create array F3 and place transformed array in it
         T2     ! [ ] Input waveform to be transformed

$MULTIPLY F4    ! Create array F4 and place product in it
         F2     ! [ ] First input waveform to be multiplied
         F3     ! [ ] Second input waveform to be multiplied

$FFT T3         ! Create array T3 and place transformed array in it
         F4     ! [ ] Input waveform to be transformed

$A2D     D1     ! Create array D1 and place digitised values in it
         T3     ! [ ]            Input waveform to be digitised
       0.20     ! [V]            LSB value
          8     ! [bits]         Number of bits of A2D precision
          0     ! [bins]         First bin to digitise ( inclusive )
        256     ! [bins]         Number of bins to digitise after first bin
          1     ! [bins/sample]  Bin sampling rate

$WRITE D1 BIN APP D1.BIN

$ENDLOOP        ! Loop ends, re-trace back to the last RETURNL call
```

As shown in the command file, the simulation space is specified to be 4000 PRIs at 400 hertz by 256 range bins at 200 megahertz. One point target is then placed at point (0,5000,0) in the Cartesian equivalent of this simulation space. The aircraft is placed 1500 metres above ground level, flying at 80 metres per second along an ideal flight path. Therefore as per Equation 2.4, the range to the target at closest approach is

$$\sqrt{1500^2 + 5000^2} = 5220.1533 \quad \text{metres.}$$

Similarly, the range to the target at either end of the flight path is

$$\sqrt{1500^2 + 5000^2 + (\frac{1}{2}\frac{4000-1}{400} \cdot 80)^2} = 5235.4484 \quad \text{metres,}$$

where the distance covered by the aircraft is calculated according to Equations 2.1, 2.2 and 2.3.

The radar pulse is specified to be a chirp pulse with a bandwidth of 100 megahertz and a 5 volt amplitude. Further, the rise time is to be 5 nanoseconds, the fall time 10 nanoseconds

and the main width 1 microsecond. The total length of 1015 nanoseconds represented in range bins is given by this total length multiplied by the simulation sampling frequency of 200 megahertz as per Equations 2.8 and 2.16,

$$1015{\cdot}10^{-9} \cdot 200{\cdot}10^{6} = 203 \quad \text{range bins.}$$

The pulse IQ waveform, `T1`, is then transformed to the frequency domain and the result is passed to the `MATCHFILT` function which generates an ideal matched filter with no windowing and stores it in array `F2`.

Next comes the call to the `RETURNL` function. The radar centre frequency is set to 1 gigahertz, and the total gains less the total losses amount to 75 decibels - this figure has been made unrealistically large so as to eliminate the need for an amplification stage later. The target cross-sectional area is 1 metre squared. Based on the range to the single target at closest approach as calculated on the previous page, the range offset, R_o, has been set to 5200 metres. Finally, the antenna is to be omnidirectional. It is useful to determine at this point whether the total of 256 range bins will be sufficient to store not only the radar pulse, but also the time shift due to range. Taking into account the value of R_o, the number of range bins required to represent the round trip time delay at maximum range to target will be twice that range divided by the speed of light in a vacuum[1] multiplied by the simulation sampling frequency as per rearrangement of Equation 2.10,

$$2 \cdot \frac{5235.4484 - 5200}{3{\cdot}10^{8}} \cdot 200{\cdot}10^{6} \approx 47 \quad \text{range bins.}$$

This means that at maximum range $203 + 47 = 250$ range bins are required to store the entire delayed waveform. Thus, the 256 range bins that constitute the simulation space in the range direction are enough to satisfy this requirement. The same calculation is now performed for the range to target at closest approach:

$$2 \cdot \frac{5220.1533 - 5200}{3{\cdot}10^{8}} \cdot 200{\cdot}10^{6} \approx 27 \quad \text{range bins.}$$

Therefore, the range curvature plot should bottom out at range bin 27 in the middle of the

[1] Assuming that atmospheric effects on this speed are negligible.

flight path, and reach range bin 47 on each end of the flight path.

After this, the return waveform which has been stored in array `T2` is transformed to the frequency domain and multiplied with its ideal matched filter. The return waveform has now been range compressed. This range compressed waveform is transformed back to the time domain and passed into the `A2D` function. Here, all of the 256 range bins are digitised and stored in array `D1`. The converter has eight bits of precision with the lowest significant bit weighing in at 200 millivolts.

The digitised waveform is then appended to file in raw binary format with a call to the `WRITE` function.

Finally, the call to the `ENDLOOP` function returns simulation flow back to the beginning of the `RETURNL` call until the loop has repeated 4000 times - once for each PRI. After this, the simulation ends.

The first image is shown in Figure 5.1 on the following page. The original image was obtained by calculating the magnitude of each IQ pair in file `D1.BIN` and converting to a `TIFF` file such that a magnitude of 255 is white and 0 is black. This file, 256 pixels in range (horizontal) by 4000 pixels in azimuth (vertical), was then imported into an image processing package. First, the last 128 range bins were discarded as they all had a magnitude of zero. Second, the image was mirrored about a horizontal axis so that the imaginary direction of flight was then from bottom to top rather than the other way round. Third, a gamma correction factor of 4.00 was applied to the image so as to make the sidelobes more visible to the eye for the sake of qualitative analysis. Fourth, the image was scaled down in azimuth by a factor of eight, resulting in an image with 500 pixels in azimuth and still 128 pixels in range. Fifth, in order to make the range direction more clear, the image was scaled up in range by a factor of two. Finally, the image was rotated anticlockwise by 90 degrees, resulting in the final image that is shown which is 500 pixels by 256 pixels in size. To put the simulation geometry in context, the aircraft may be thought of as flying from the bottom right corner of the image, PRI number 0, to the bottom left corner, PRI number 3999, looking out onto the target on top, with range increasing from bottom to top. At the exact halfway point of the flight, that is between PRI numbers 1999 and 2000, the target is closest and it was determined that the response peaked at pixel 27 (range bin 27) as expected. At each end

of the flight path, PRI numbers 0 and 3999, the target is at maximum range and it was determined that the response peaked at pixel 47 (range bin 47), also as calculated earlier[2].

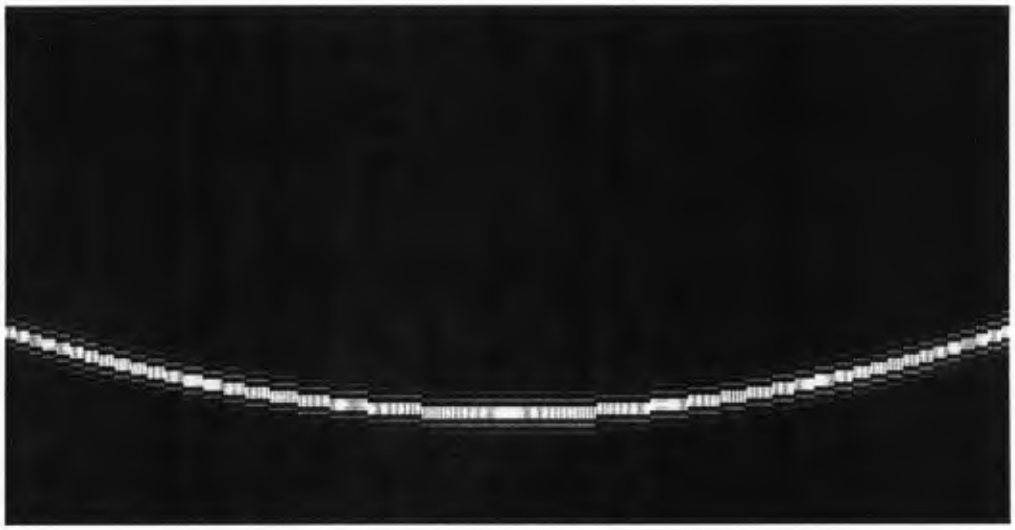

Figure 5.1: Sim 1 : Manipulated Range Compressed Image - PRIs 0-3999, Bins 0-127

For interest, Figure 5.2 on the following page shows a small section of the original image, that is before discarding range bins, gamma correction and scaling. The section shows a rectangle bounded by PRI numbers 1750 and 2249 and range bins 0 and 255. As before, the direction of flight in azimuth is from the bottom left corner to the bottom right corner of the image. Range increases from bottom to top.

[2]Recall that all pixel and range bin numbers are calculated as starting from zero according to the convention laid down earlier in this dissertation. Most graphics viewers also start numbering pixels from zero, thus allowing for easy comparison of pixel or range bin positions between a viewer and results of calculations performed here.

Figure 5.2: Sim 1 : Original Range Compressed Image - PRIs 1750-2249, Bins 0-255

The original binary file, `D1.BIN`, was now azimuth compressed and the final radar image was stored in file `D1.IMG`. This image was converted into a `TIFF` file as before. This file, still 4000 pixels in azimuth by 256 pixels in range, was then imported into an image processing package. First, the last 128 range bins were discarded as they all had a magnitude of zero. Second, as there is only one point target, the first and last 1900 PRIs were discarded as they were all zero, resulting in an image with 200 pixels in azimuth by 128 pixels in range. Third, a gamma correction factor of 4.00 was applied to the image to emphasise the sidelobes as before. Fourth, due to some orientation changes performed by the azimuth compression software, the image was rotated by 180 degrees. Finally, the image was scaled up both horizontally and vertically by a factor of two. This resulting image, with 400 pixels in azimuth and 256 pixels in range is shown in Figure 5.3 on the following page. Once again, the aircraft may be thought of as flying along the bottom edge of the image from right to left. Range increases from bottom to top. As expected, the peak of the target is at range bin 27 and spread over azimuth PRIs 1999 and 2000.

Figure 5.3: Sim 1 : Manipulated Azimuth Compressed Image - PRIs 1900-2099, Bins 0-127

Figure 5.4 on the following page shows a small section of the azimuth compressed image, that is before discarding range bins, gamma correction and scaling. The section shows a rectangle bounded by PRI numbers 1750 and 2249 and range bins 0 and 255 - that is the same area as in Figure 5.2 which showed the image before azimuth compression. As always, the direction of flight in azimuth is from the bottom left corner to the bottom right corner of the image. Range increases from bottom to top.

Finally, Figure 5.5, also on the following page, shows a MathCAD plot of the magnitude values of an azimuth line from PRI 1900 to PRI 2099 at range bin 27. The data used for this plot was also passed through a peak analysis program which forms part of Horrell's azimuth compression package. The peak was confirmed to be at PRI 1999 with a greyscale magnitude of 220. The 3 decibel peak width was 4.4 samples which converts to $4.4 \cdot 80/400 = 0.88$ metres. This means that the azimuth resolution is 88 centimetres. This value will be used for comparison with the resolution of targets forming part of non-ideal simulations later. The peak sidelobe was found to be at PRI 1992 with an attenuation of 11.8 decibels. Overall, it can be seen that the target has been focused to a narrow and tall peak, with relatively small sidelobes. This is of course what one would expect from a simple simulation under ideal conditions.

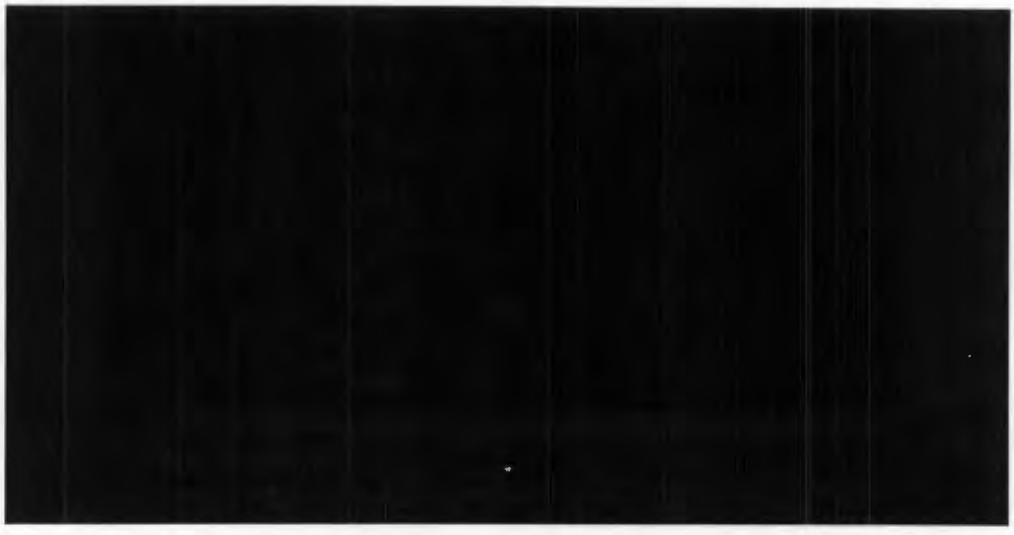

Figure 5.4: Sim 1 : Original Azimuth Compressed Image - PRIs 1750-2249, Bins 0-255

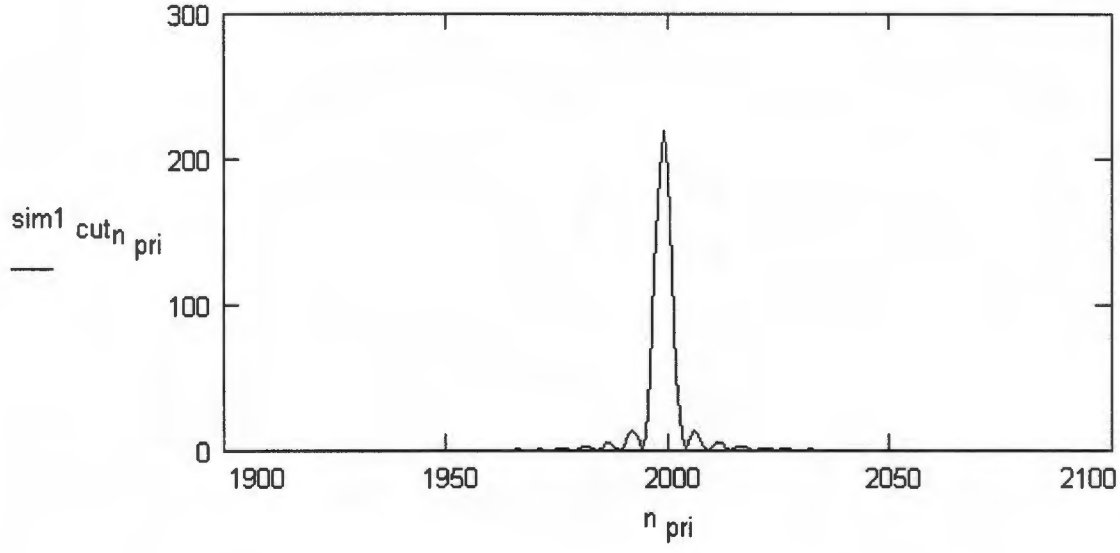

Figure 5.5: Sim 1 : Cut Through Bin 27 - PRIs 1900-2099

5.3 Simulation With Noisy Flight Path

The command file used for this simulation is shown below:

```
! SIM2.CMD - Simulation With Noisy Flight Path.
! ==========================================

.
.    <cut>
.

$MOTIONSIN M1 ! Create array M1 and place sinusoidal motion data in it
         A    ! [ ]          Use which GENERAL parameters ?
      0.00    ! [m]          Sine wave amplitude along x-axis
    0.0000    ! [cycles/PRI] Sine wave frequency along x-axis
       0.0    ! [m]          Sine wave "DC" offset along x-axis
   0.10 0.0001 0.0 ! Same for y-axis
   0.05 0.0020 0.0 ! Same for z-axis
   0.00 0.0000 0.0 ! Same for squint angle

$MOTIONRND M2 ! Create array M2 and place random motion data in it
         A    ! [ ]   Use which GENERAL parameters ?
     0.005    ! [m]   Maximum allowed +/- variation along x-axis
     0.010    ! [m]   Maximum allowed +/- variation along y-axis
     0.010    ! [m]   Maximum allowed +/- variation along z-axis
     0.000    ! [deg] Maximum allowed +/- variation in squint angle

$ADD M3       ! Create array M3 and place sum of the two input arrays in it
        M1    ! [ ] First array for addition
        M2    ! [ ] Second array for addition

$GEOMETRY G1  ! Create array G1 and place simulation geometry data in it
         A    ! [ ]   Use which GENERAL parameters ?
        M3    ! [ ]   Input array with aircraft offset motion data
    1500.0    ! [m]   Aircraft altitude above ground level
      80.0    ! [m/s] Aircraft velocity (groundspeed)

.
.    <cut>
.
```

This command file is the same as the one for the first simulation, with the exception that aircraft motion errors have been introduced. Therefore, only the changes and additions have been shown above. After the point target is specified, the `MOTIONSIN` function is called. For the y-axis, the sinusoid has been given such a large period that it advances only $2\pi \cdot 4000 \cdot 0.0001 = 2.51$ radians, or 144 degrees, over the entire 4000 PRIs. Through this is simulated a slow and steady drift of the aircraft in the positive y-axis direction, and thus

towards the target. This drift is reversed at 90 degrees as the aircraft begins turning back to its ideal y-axis position. In the z-axis however there is more activity since the period is smaller. This simulates a relatively rapid vertical movement of the aircraft over and under its ideal flight path. There are no sinusoidal motion errors introduced in the x-axis and squint angle.

The `MOTIONRND` function is called next and is used to generate very small pseudo-random motion errors in the x-, y- and z-axes. This is used to simulate vibration during flight as well as possibly minute aircraft guidance system inaccuracies and general system noise.

The two sets of motion errors are then added together, resulting in array `M3` which is consequently passed to the `GEOMETRY` function.

The first image of this simulation is shown in Figure 5.6 on the following page. As before, the magnitude of each IQ pair in the file `D1.BIN` was calculated and the results converted to a `TIFF` file. In the same way, the last 128 range bins were discarded; the image was mirrored about a horizontal axis; a gamma correction factor of 4.00 was applied; the image was scaled down by a factor of eight in azimuth and scaled up by a factor of two in range; and finally, the image was rotated anticlockwise by 90 degrees. The result is an image 500 pixels in azimuth by 256 pixels in range, with the aircraft flying from the bottom right corner to the bottom left corner with range increasing from bottom to top. Even though the printer resolution is limited, it is still possible to see that the main bright line traced out by the peak is now more unevenly coloured when compared with the image shown in Figure 5.1.

Also shown on the following page is Figure 5.7 where a small section of the original image is presented, that is before discarding range bins, gamma correction and scaling. The section shows a rectangle bounded by PRI numbers 1750 and 2249 and range bins 0 and 255. As before, the direction of flight in azimuth is from the bottom left corner to the bottom right corner of the image. Range increases from bottom to top. Comparison with the ideal image shown in Figure 5.2 earlier yields that even at this limited resolution the degradation of the image is obvious.

Figure 5.6: Sim 2 : Manipulated Range Compressed Image - PRIs 0-3999, Bins 0-127

Figure 5.7: Sim 2 : Original Range Compressed Image - PRIs 1750-2249, Bins 0-255

As in the first simulation, the file `D1.BIN` was azimuth compressed and the resulting image converted to a `TIFF` file. The last 128 range bins and the first and last 1900 PRIs were discarded; a gamma correction factor of 4.00 was applied; the image was rotated by 180 degrees; and finally the image was scaled up both horizontally and vertically by a factor of

two. The resulting image, with 400 pixels in azimuth and 256 pixels in range is shown in Figure 5.8 below. Once again, the aircraft may be thought of as flying along the bottom edge of the image from right to left. Range increases from bottom to top. Compare this image with the one shown in Figure 5.3 - the degradation in the focusing of the target is obvious, especially in azimuth where the spreading is more visible and two sidelobes are clearly seen. Analysis of the peak position shows that it is still in range bin 27, but has shifted to PRI 1997 in azimuth.

Figure 5.8: Sim 2 : Manipulated Azimuth Compressed Image - PRIs 1900-2099, Bins 0-127

Figure 5.9 on the following page shows a small section of the azimuth compressed image, that is before discarding range bins, gamma correction and scaling. The section shows a rectangle bounded by PRI numbers 1750 and 2249 and range bins 0 and 255 - that is the same area as in Figure 5.7 which showed the image before azimuth compression. As always, the direction of flight in azimuth is from the bottom left corner to the bottom right corner of the image. Range increases from bottom to top. Again, even at this resolution the degradation to the point target is visible.

Figure 5.9: Sim 2 : Original Azimuth Compressed Image - PRIs 1750-2249, Bins 0-255

Finally, Figure 5.10 on the following page shows a MathCAD plot of the magnitude values of an azimuth line from PRI 1900 to PRI 2099 at range bin 27. The peak is no longer as narrow as the one for the ideal case shown in Figure 5.5. The sidelobes are also greater and noise has distorted every portion of the plot. The peak analysis program calculated that the peak occurs at PRI 1997, meaning that the peak has shifted by about 3 PRIs or 60 centimetres towards the right (the starting position of the aircraft). The peak is now 7.2 samples wide which converts to 1.44 metres - compare this with the ideal peak of the previous simulation which was 0.88 metres wide. The peak sidelobe was found to be at PRI 2003 with an attenuation of only 3.1 decibels. This large sidelobe implies that a large number of target recognition or image processing systems may see this as two point targets instead of one. Other than that, the target is still reasonably well focused when compared with images obtained from the processing of real SAR flight data. This simulation has shown how even very minute flight path errors can adversely affect the final radar images. Hence the great emphasis placed upon motion compensation algorithms in modern SAR research.

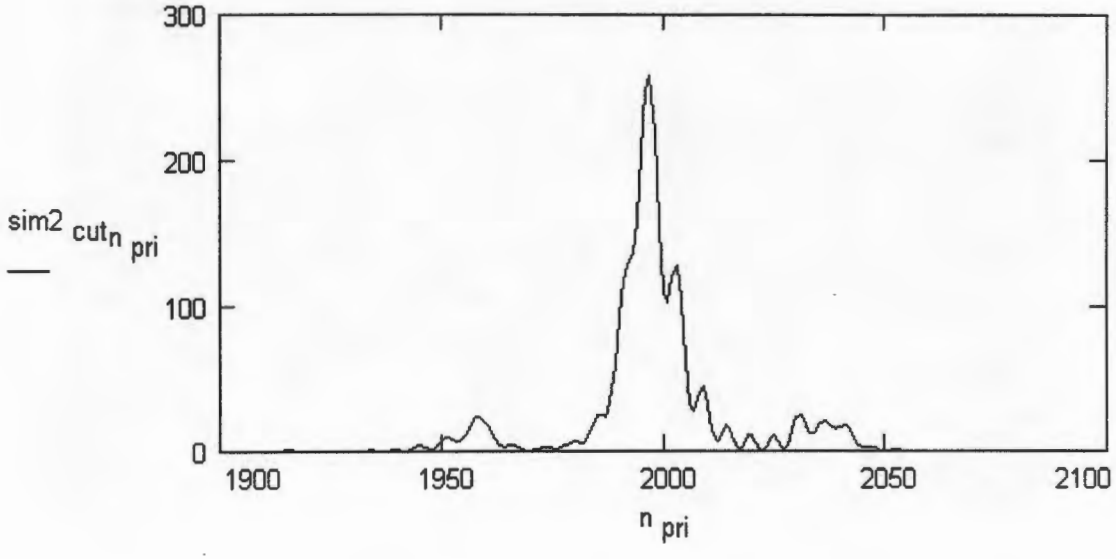

Figure 5.10: Sim 2 : Cut Through Bin 27 - PRIs 1900-2099

5.4 Simulation of Three Point Targets

The command file used for this simulation is shown below:

```
! SIM3.CMD - Simulation of Three Point Targets.
! ==============================================

$GENERAL A
      4000    ! [ ]   Number of PRIs to simulate
     400.0    ! [Hz]  Pulse Repetition Frequency
       256    ! [ ]   Number of range samples to simulate
       0.2    ! [GHz] Simulation frequency

$TARGET  A
       0.0    ! [m] x direction offset from mid path
    5000.0    ! [m] y ground range at closest approach
       0.0    ! [m] z height above ground level

$TARGET  A
       0.0    ! [m] x direction offset from mid path
    5050.0    ! [m] y ground range at closest approach
      20.0    ! [m] z height above ground level

$TARGET  A
     -40.0    ! [m] x direction offset from mid path
    5000.0    ! [m] y ground range at closest approach
       0.0    ! [m] z height above ground level

.
.    <cut>
.

$RETURNL T2   ! Create array T5 and place waveform radar return in it
        G1    ! [ ]   Simulation geometry array to be used
        A1    ! [ ]   Envelope of pulse to be transceived
       1.0    ! [GHz] Radar centre frequency
      38.0    ! [dB]  Antenna gains
       5.0    ! [dB]  Losses ( TxLoss + RxLoss + PropagationLoss )
       1.0    ! [m^2] Target cross-sectional area
    5200.0    ! [m]   Range offset for simulation geometry
     CHIRP    ! [ ]   Pulse type - 'CHIRP' or 'MONO'
      0.10    ! [GHz] Chirp bandwidth (arb non-zero for MONO)
      NULL    ! [ ]   Antenna type - 'SIN' or 'NULL'
       0.0    ! [deg] Antenna elevation beamwidth
       0.0    ! [deg] Antenna azimuth beamwidth
       0.0    ! [deg] Antenna elevation angle (down is negative)
       0.0    ! [deg] Antenna squint angle (backward is positive)

$AMPLIFY T2   ! Create array T2 and place amplified array in it
        T2    ! [ ] Input waveform to be amplified
     55.56    ! [dB] Amplification factor
```

```
.
.    <cut>
.

$A2D    D1     ! Create array D1 and place digitised values in it
        T3     ! [ ]            Input waveform to be digitised
      0.04     ! [V]            LSB value
         8     ! [bits]         Number of bits of A2D precision
         0     ! [bins]         First bin to digitise ( inclusive )
       256     ! [bins]         Number of bins to digitise after first bin
         1     ! [bins/sample]  Bin sampling rate

$WRITE D1 BIN APP D1.BIN

$ENDLOOP       ! Loop ends, re-trace back to the last RETURNL call
```

This simulation builds on the previous one by adding two more point targets, as well as an amplification stage after the RETURNL call.

The second target is placed 50 metres further along the ground (y-axis) than the first target, and also 20 metres above ground level. The third target is the same as the first, but 40 metres closer to the starting position of the aircraft. Therefore, the range curves for targets one and three should overlap for a large part of the 4000 PRI simulation. This is aimed at testing how well the azimuth compression software can discern two overlapping range curvatures and to see how the returns from the two targets interfere with each other. Also, note that the effective range to the second target at closest approach is

$$\sqrt{5050^2 + 1500^2 + 20^2} - 5200 = 68.1021 \quad \text{metres.}$$

The number of range bins required to represent that range is 91, according to a rearranged form of Equation 2.10. Together with the 203 range bins required to store the radar chirp pulse, one may see that during certain parts of the simulation at least $(203 + 91) - 256 = 38$ range bins will be discarded at the end of the return pulse for that target. Through this it is attempted to simulate a target that is so far away that the radar receiver is turned off before the entire return has been captured to make way for the transmission of the next chirp pulse. The interest lies in seeing how badly this will affect the focusing and brightness of that target. In theory, the target should be dimmer as less power is being received for that target. There should be no visible ill effects on the level of focusing as the return waveform will still be a square pulse shape in the range direction and therefore will match

well with the original pulse. In azimuth there should no change at all.

Further, after the generation of the return waveform, the resulting array, `T2`, is amplified by a gain of 55.56 decibels with a call to the `AMPLIFY` function, as would be done in a real SAR system. Consequently, the value of the antenna gains under the `RETURNL` function is now reduced to a more realistic value of 38 decibels.

Finally, the value of the lowest significant bit in the analogue to digital converter had to be lowered to 40 millivolts due to the changes in gains as described above.

Figure 5.11 on the following page shows a manipulated radar image before azimuth compression. As in the last two simulations, the magnitude of all IQ pairs in file `D1.BIN` was taken and the results converted to a `TIFF` file. Again, the last 128 range bins were discarded; the image was mirrored about a horizontal axis; a gamma correction factor of 4.00 was applied; the image was scaled down by a factor of eight in azimuth and scaled up by a factor of two in range; and finally, the image was rotated anticlockwise by 90 degrees. The result is an image 500 pixels in azimuth by 256 pixels in range, with the aircraft flying from the bottom right corner to the bottom left corner with range increasing from bottom to top. Looking at the two overlapping range curvatures a "beating" effect may be seen around the point where the two curvatures are almost exactly on top of each other. Around this area the difference in phase between the two targets causes the IQ components to add destructively (giving a low magnitude - dark) and then constructively again (giving a high magnitude - bright) and so on. Outside this small central area, the difference in range between the two targets increases such that their returns overlap and interfere less and less until at both ends of the flight the curvatures of the two targets are quite separate. Finally, the remaining target curvature at the top of the image is seen to be more dark than the first two. This is mostly due to the fact that it is further away and thus the amplitude decreases. In addition, the loss of a substantial amount of range bins from the end of the return waveform means that less power is being received, implying that the matched filter will return a peak of lower amplitude. Other than this there do not seem to be any easily visible effects due to these range bin losses.

Figure 5.11: Sim 3 : Manipulated Range Compressed Image - PRIs 0-3999, Bins 0-127

Figure 5.12 on the following page shows a small section of the original image. As always, this is before discarding range bins, gamma correction and scaling. The section shows a rectangle bounded by PRI numbers 1750 and 2249 and range bins 0 and 255. The direction of flight in azimuth is from the bottom left corner to the bottom right corner of the image. Range increases from bottom to top. From the image it is almost impossible to tell that there are three targets as the two overlapping targets seem as one. However, this image illustrates the "beating" phenomenon very well.

The file `D1.BIN` was now azimuth compressed and the results converted to a `TIFF` file. The last 128 range bins and the first 1700 and last 1900 PRIs were discarded; a gamma correction factor of 4.00 was applied; the image was rotated by 180 degrees; and finally the image was scaled up vertically (range) by a factor of two. The resulting image, with 400 pixels in azimuth and 256 pixels in range is shown in Figure 5.13 on the following page. As always, the aircraft is flying along the bottom edge of the image from right to left. Range increases from bottom to top. Although the calculations are not shown here, all three targets are correctly positioned in range and azimuth. As may be seen the azimuth compression package was able to separate the two overlapping target curvatures into two distinct point targets. The further target seems as bright as the two closer ones here, but this is deceiving as the

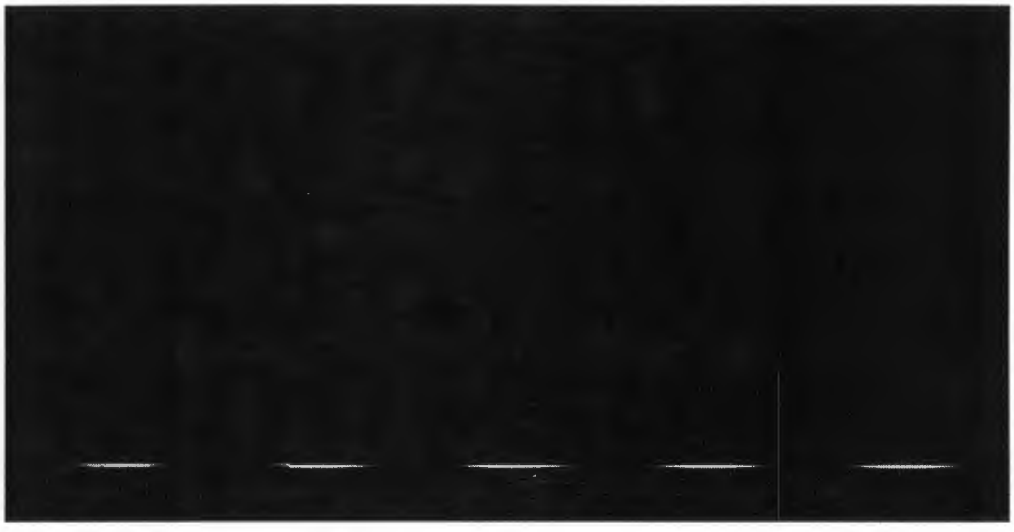

Figure 5.12: Sim 3 : Original Range Compressed Image - PRIs 1750-2249, Bins 0-255

gamma correction operation increased its brightness. Once again, it is difficult to see any degradation of this far target due to the cutting off of the return waveform.

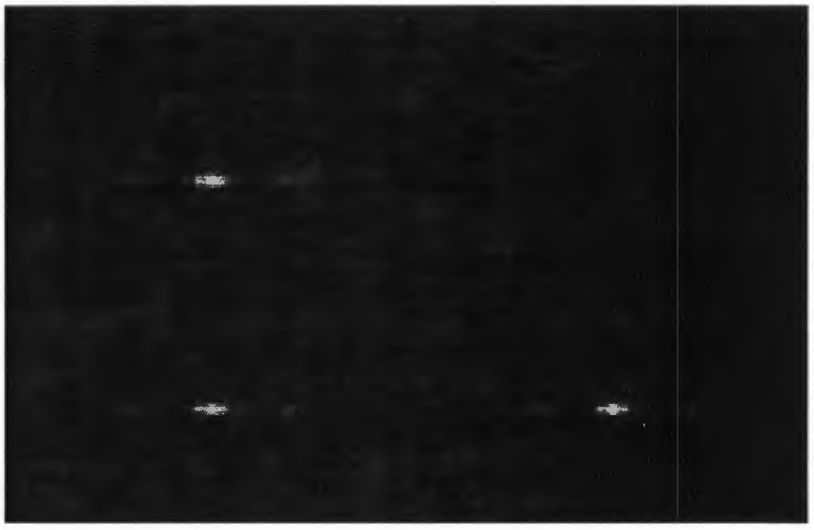

Figure 5.13: Sim 3 : Manipulated Azimuth Compressed Image - PRIs 1700-2099, Bins 0-127

Figure 5.14 on the following page shows a small section of the azimuth compressed image, that is before discarding range bins, gamma correction and scaling. The section shows a rectangle bounded by PRI numbers 1700 and 2199 and range bins 0 and 255. As always, the direction of flight in azimuth is from the bottom left corner to the bottom right corner of the image. Range increases from bottom to top. Here it may be seen that the far target is indeed dimmer than the two closer ones as one would expect. Once again, except for this dimness, there seem to be no other effects caused by the cutting off of the last range bins for this target.

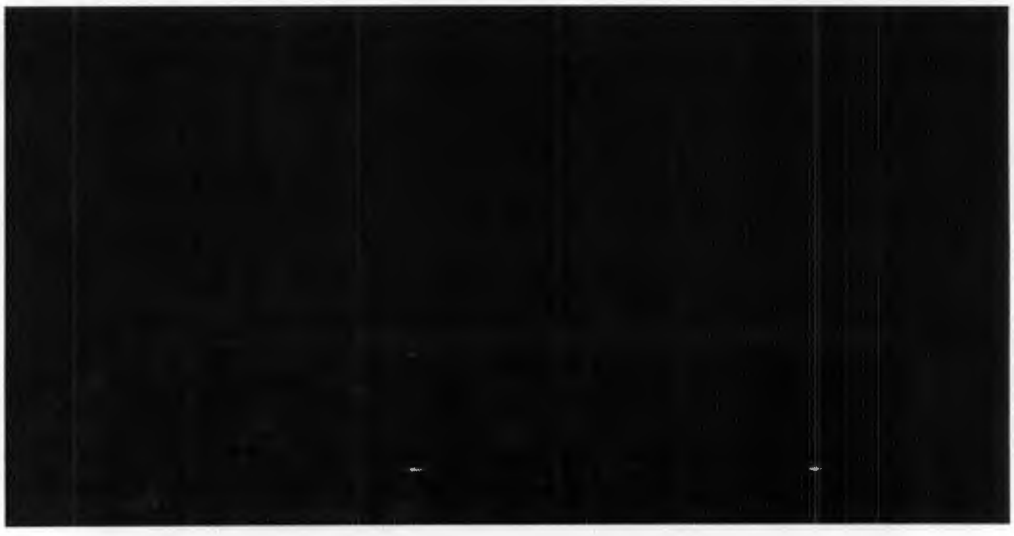

Figure 5.14: Sim 3 : Original Azimuth Compressed Image - PRIs 1700-2199, Bins 0-255

A MathCAD plot of a cut through the central target is not shown here as it is the same as the one shown for the second simulation in Figure 5.10 with the exception of a constant change in amplitude caused by gain changes.

5.5 Simulation With Directional Antenna

The command file used for this simulation is shown below:

```
! SIM4.CMD - Simulation With Directional Antenna.
! ================================================

.
.    <cut>
.

$RETURNL T2    ! Create array T5 and place waveform radar return in it
        G1     ! [ ]   Simulation geometry array to be used
        A1     ! [ ]   Envelope of pulse to be transceived
       1.0     ! [GHz] Radar centre frequency
      38.0     ! [dB]  Antenna gains
       5.0     ! [dB]  Losses ( TxLoss + RxLoss + PropagationLoss )
       1.0     ! [m^2] Target cross-sectional area
    5200.0     ! [m]   Range offset for simulation geometry
     CHIRP     ! [ ]   Pulse type - 'CHIRP' or 'MONO'
      0.10     ! [GHz] Chirp bandwidth (arb non-zero for MONO)
       SIN     ! [ ]   Antenna type - 'SIN' or 'NULL'
       8.0     ! [deg] Antenna elevation beamwidth
       5.0     ! [deg] Antenna azimuth beamwidth
     -16.7     ! [deg] Antenna elevation angle (down is negative)
       0.0     ! [deg] Antenna squint angle (backward is positive)

.
.    <cut>
.
```

This is the final sample simulation, and it is the most realistic one of the four presented in this chapter. The only difference between this simulation and the previous one are changes made to the **RETURNL** function. The antenna type has been changed to the **SIN** pattern with elevation and azimuth beamwidths equal to 8 and 5 degrees respectively. The azimuth beamwidth has been made unusually small so that the effects of the antenna gain pattern may be seen over only the 4000 PRIs that are being simulated here. Note that at the start of the flight path the azimuth angle to a target 5000 metres away and in the centre of the flight path is 4.57 degrees by simple geometry. This means that the target goes through a total of 9.14 degrees in azimuth over the entire 4000 PRIs. Hence, in order to see any clear effects of the sinusoidal antenna gain pattern in azimuth, the beamwidth must be made substantially less than 9.14 degrees - in this case 5 degrees as mentioned above. The antenna elevation angle has been set to 16.7 degrees down towards the targets. This angle is equal to the

elevation of the first target. Thus the antenna gain pattern in elevation has been centred on this first target. In azimuth, no squint angle has been introduced.

As the introduction of the sinusoidal response antenna does not visibly affect the azimuth compressed images, the only image presented in this section is shown in Figure 5.15 below. The image shows a manipulated version of the `TIFF` file obtained by taking the magnitude of all IQ pairs in file `D1.BIN`. As always, the last 128 range bins were discarded; the image was mirrored about a horizontal axis; a gamma correction factor of 4.00 was applied; the image was scaled down by a factor of eight in azimuth and scaled up by a factor of two in range; and finally, the image was rotated anticlockwise by 90 degrees. The result is an image 500 pixels in azimuth by 256 pixels in range, with the aircraft flying from the bottom right corner to the bottom left corner with range increasing from bottom to top. The effects of the sinusoidal antenna response are clearly illustrated in azimuth, where the antenna gain drops off sharply towards either end of the flight path as expected.

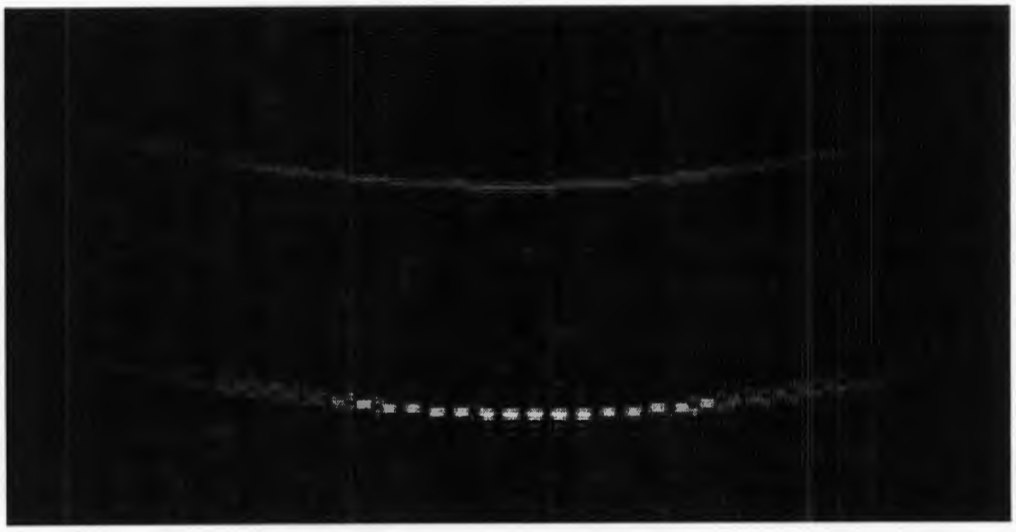

Figure 5.15: Sim 4 : Manipulated Range Compressed Image - PRIs 0-3999, Bins 0-127

Chapter 6

Conclusions and Recommendations

A need had arisen within the RRSG for a user friendly yet flexible, portable and fast simulator of SAR. In this dissertation, a simulator package called RadSim has been put forward, together with extensive test results showing that it meets the required criteria.

6.1 Conclusions

The first, and the most important conclusion is that all of the requirements laid out in the Terms of Reference at the beginning of this dissertation have been satisfied. That is,

- The RadSim source code complies with the ANSI C standard and had been shown to compile and execute on various hardware and operating system platforms successfully.
- Each simulation function call has been coded as a separate entity, with the code and data encapsulated in a similar manner to a C++ class. This allowed for many functions to be tested on their own, thus simplifying the various test procedures, as well as ensuring easier maintenance in the future.
- The input to RadSim is the plain ASCII command file, after which no user interaction is required except in cases of error. This means that for large simulations the program can be run as a batch process without human supervision.
- The simulation geometry may be specified in detail as defined by the requirements.

- The behaviour of the radar may also be specified in detail as defined by the requirements.
- Any array may be dumped to a raw binary or plain ASCII file at any point during the simulation.

The result of the research work is therefore a working, flexible, portable, easy to use and well coded SAR simulator. It also offers a welcome improvement over the VAX-VMS based simulator with regard to SAR simulation for two reasons. First, the simulation may be performed locally in the lab, or even at home, on a PC or UNIX workstation. Secondly, the return waveform arrays for each PRI are automatically appended to one file instead of being saved to hundreds of different files and requiring the user to append them manually.

The problem of phase jumping was successfully tackled, meaning that shorter arrays than before at a lower sampling rate are necessary to obtain meaningful and useful data. This results in a less memory intensive run and a simulation requiring less processing time.

6.2 Recommendations

The author hereby recommends that the following issues and suggestions be dealt with:

- Although the simulator may be used to simulate SAR under a myriad of conditions, some members of RRSG who are researching more specialised and advanced areas of SAR, have expressed that the following additional functionality or extensions to existing functionality should be useful:
  - Point targets that move linearly along a straight line path in three dimensions [6].
  - Planar targets with width and breadth in addition to point targets [14].
  - User defined antenna gain patterns which may be read from an external file [8].
  - User defined aircraft motion offsets which may be read from an external file [8].
  - Support for the simulation of stepped frequency pulsing within one PRI [8, 14].

- Two of the points above require the facility of reading data from external files. It should be useful to extend this idea and develop a simulation function called `READ` which, given the file name, the data type, and ASCII or Binary, will read the array contained in the file into the specified array. It may also be useful to be able to configure the number and position of the rows that are to be read, as one may not want the entire file loaded.

- The design of the simulator has been based on a model which lends itself to object orientation, where the data set-up and functionality of each simulation function is encapsulated and the interface to each simulation function is standard. This model was implemented in as far as ANSI C allows, since C does not cater for object orientation. Now that C++ and object orientation have become the industry standard and most platforms support ANSI C++, it should be a worthwhile exercise to convert the implementation to C++. This should result in a more clear, concise, maintainable and more modern implementation.

- The user interface to RadSim, that is the text command file, is rather crude when compared with trends shown by the success of Windows 3.1, Windows 95 and XWindow. It may thus be useful to develop a comprehensive GUI either using a package like Borland C++ Builder for Microsoft Windows or XMotif for UNIX platforms. In addition, there are now several third party multi-platform class libraries that allow for the design and implementation of portable GUIs. Use of such a library should be investigated as it would continue to make RadSim a truly portable program.

- Finally, it may prove interesting to develop an "expert" type system which, given a command file, will parse it to determine whether all parameters are not only within range, but that they together make sense and that the command file contains the calls that will result in a successful and meaningful simulation. As a further possibility, the system could generate a full command file for a simulation, given only a limited set of necessary parameters, with the rest of the parameters and simulation function calls being derived or intelligently assumed.

Appendix A

User's Manual and Function Reference

In this appendix, the function calls supported by RadSim are listed alphabetically. For each listing, the function call structure, the required parameters and the operation of the call is detailed. Numerous examples are given to illustrate the various uses of all these calls. Before that however, the introduction below deals with some prerequisite issues.

Note that it is assumed throughout this user's manual and function reference that the reader is familiar with the thesis dissertation proper and the ideas and terminology contained therein.

A.1 Introduction

There are six allowable array types as follows:

- A - 'amplitude only' array. This is a single column array used to represent the amplitude of a waveform.
- D - 'digitised' array. This is a two column array (I and Q values) used to represent digitised data. Thus, all the values in such an array have no fractional part.

- F - 'frequency' array. This is a two column array (I and Q values) used to represent analogue frequency domain waveforms.
- G - 'geometry' array. This is a three column array (range, elevation and azimuth) used to represent the simulation geometry of any number of point targets.
- M - 'motion' array. This is a four column array (x error, y error, z error and squint error) used to represent the displacements from the ideal flight path due to erratic aircraft motion.
- T - 'time' array. This is a two column array (I and Q values) used to represent analogue time domain waveforms.

The array id character must be followed by an 'array index'. This index is simply an integer used to differentiate between different arrays of the same type. The index must be within the range of 1 to 256. Thus, for example, to reference an F type array with an index of 10, one would write F10.

In the function reference listings below, such an array reference will be indicated by the text 'idix', meaning identifier-index.

A.2 Running RadSim

To run RadSim type

```
radsim command_file_name
```

RadSim will then use the command file specified to perform a simulation. All output files will be placed in the current directory. Also, all messages will be printed to standard output unless manually redirected to file.

There are three types of messages. Firstly, **Information** messages, which print status information and indicate the progress of the simulation. Secondly, **Warning** messages which may warn about unparsed sections of the command file, unused parameters, or suspicious parameter values. Finally, there are the **Error** messages which cause the simulation to abort. RadSim is able to pick up most errors before starting the simulation proper.

For examples of full simulations and the respective command files the reader is referred to Chapter 5 of this document where three example simulations of varying complexity are given.

A.3 Function Reference with Examples

A.3.1 A2D

Call Structure:

```
$A2D    Yn     ! Create array Yn and place digitised values in it
        Xn     ! [ ]            Input waveform to be digitised
    fff.ff     ! [V]            LSB value
       iii     ! [bits]         Number of bits of A2D precision
       iii     ! [bins]         First bin to digitise ( inclusive )
       iii     ! [bins]         Number of bins to digitise after first bin
       iii     ! [bins/sample]  Bin sampling rate
```

Parameter Meanings:

Symbol	Unit	Range	Description
Y_n	n/a	'D'	Output array, of type 'Digitised'
X_n	n/a	'T' or 'F'	Input array, of type 'Time' or 'Frequency'
V_{LSB}	V	> 0.0	Volt value of lowest significant bit
N_{bits}	bits	4 or 8	Number of bits of A2D precision
B_{start}	bins	0 to $N_{bin} - 1$	First bin to begin digitisation at
B_{num}	bins	1 to $N_{bin} - 1$	Number of bins to digitise after first bin
S_{rate}	bins/sample	1 to $N_{bin} - 1$	Bin sampling rate

Function:

This function simulates an analogue to digital converter. X_n is digitised to N_{bits} bits of precision. B_{start} is the starting bin and B_{num} bins are processed thereafter with $(S_{rate} - 1)$ bins being skipped for every sample taken. The value of the lowest significant bit is V_{LSB}. The results are returned in array Y_n.

Special Rules:

The value of $B_{start} + B_{num}$ must be less than the total number of bins being simulated. A B_{num} of zero indicates that all available bins after B_{start} are to be processed.

Examples:

1.) `$A2D D1 T1 0.00005 8 0 0 2`

The complex waveform in T1 is to be digitised to an 8 bit precision and stored in array D1. The starting bin is 0 and the number of bins to process is also 0 indicating that the entire range line is to the processed. Every second bin is sampled, and the value of the lowest significant bit represents 50 microvolts.

2.) `$A2D D1 T1 0.00005 8 10 20 1`

Same as above, but only bins 10 to 30 are to be processed, and a sample is to be taken for each one.

A.3.2 ADD

Call Structure:

```
$ADD Yn      ! Create array Yn and place sum of the two input arrays in it
     Xn1     ! [ ] First array for addition
     Xn2     ! [ ] Second array for addition
```

Parameter Meanings:

Symbol	Unit	Range	Description
Y_n	n/a	'A', 'D', 'F', 'G', 'M' or 'T'	Output array, of any type
X_{n_1}	n/a	'A', 'D', 'F', 'G', 'M' or 'T'	Input array 1, of any type
X_{n_2}	n/a	'A', 'D', 'F', 'G', 'M' or 'T'	Input array 2, or any type

Function:

This function simulates an adder. The complex arrays X_{n_1} and X_{n_2} are added and the result is stored in array Y_n.

Special Rules:

All three array types must be the same. The two input arrays must have the same SGI.

Examples:

1.) `$ADD A3 A1 A2`

Add arrays A1 and A2 and store the resulting array in A3.

2.) `$ADD D1 D1 D2`

Add array D1 to D2 and store the resulting array back in D1.

3.) `$ADD F1 F1 F1`

Add array F1 to itself and store the resulting array back in F1.

A.3.3 AMPLIFY

Call Structure:

```
$AMPLIFY Yn    ! Create array Yn and place amplified array in it
       Xn      ! [ ] Input waveform to be amplified
   fff.ff      ! [dB] Amplification factor
```

Parameter Meanings:

Symbol	Unit	Range	Description
Y_n	n/a	'T' or 'F'	Output array, of type 'Time' or 'Frequency'
X_n	n/a	'T' or 'F'	Input array, of type 'Time' or 'Frequency'
G_a	dB	real	Amplification gain factor

Function:

This function simulates an amplifier. The complex array X_n is multiplied by the scalar gain $10^{G_a/20}$ and the result is stored in array Y_n.

Special Rules:

The output array type must be the same as the input array type.

Examples:

1.) `$AMPLIFY T2 T1 10.0`

Amplify array T1 by a gain of 10.0 decibels and store the results in T2.

2.) `$AMPLIFY F1 F1 -3.00`

Amplify array F1 by a gain of -3.00 decibels and store the results back in F1.

A.3.4 ENDLOOP

Call Structure:

```
$ENDLOOP      ! Loop ends, re-trace back to the last RETURNL call
```

Parameter Meanings:

No parameters.

Function:

This function causes control to be passed back to the first $RETURNL call in the command file, unless that $RETURNL call has processed all of the required PRIs, in which event the next $RETURNL call is sought. If all PRIs have been processed for each $RETURNL call, this function passes control to the next call in the command file.

Special Rules:

This call is to be used in conjunction with the $RETURNL call in order to provide a processing loop facility. It is similar to a for...next loop where the range is from 1 to the number of PRIs to be simulated. These loops may not be nested.

A.3.5 FFT

Call Structure:

```
$FFT    Yn      ! Create array Yn and place transformed array in it
        Xn      ! [ ] Input waveform to be transformed
```

Parameter Meanings:

Symbol	Unit	Range	Description
Y_n	n/a	'T', 'F'	Output array, of type 'Time' or 'Frequency'
X_n	n/a	'F', 'T'	Input array, of type 'Frequency' or 'Time'

Function:

This function performs Fourier transformation from the time to the frequency domain or the other way round. If X_n is in the time domain, then Y_n is the Fourier transform in the frequency domain. If X_n is in the frequency domain, then Y_n is the inverse Fourier transform in the time domain.

Special Rules:

The output array type must be in a different domain to the input array type. Note that the inverse transform of a transform of a waveform results in an array exactly the same as the original array.

Examples:

1.) `$FFT F1 T1`

Transform array T1 into the frequency domain and store the results in F1.

2.) `$FFT T1 F1`

Inverse transform array F1 into the time domain and store the results in T1.

A.3.6 GENERAL

Call Structure:

```
$GENERAL A
      iii    ! [ ]   Number of PRIs to simulate
    ff.fff   ! [Hz]  Pulse Repetition Frequency
      iii    ! [ ]   Number of range samples to simulate
    ff.fff   ! [GHz] Simulation frequency
```

Parameter Meanings:

Symbol	Unit	Range	Description
SGI	n/a	'A' to 'Z'	Simulation Group Identifier
N_{pri}	n/a	> 0	Number of PRIs to simulate
f_{prf}	Hz	> 0.0	Pulse Repetition Frequency
N_{bin}	n/a	> 0	Number of range bins to simulate
f_{sim}	GHz	> 0.0	Simulation frequency

Function:

This call is usually the fist call in a simulation and is used to set up values and parameters that are used by most other function calls in order to execute the simulation.

Special Rules:

Two or more $GENERAL calls may not have the same SGI.

Examples:

1.) `$GENERAL A 256 50.00 512 2.00`

Set up the basic parameters for a simulation 'A' with 256 PRIs and 512 bins. The PRF being 50 Hz and the simulation frequency 2 GHz.

A.3.7 GEOMETRY

Call Structure:

```
$GEOMETRY Yn    ! Create array Yn and place simulation geometry data in it
        A       ! [ ]   Use which GENERAL parameters ?
       Xn       ! [ ]   Input array with aircraft offset motion data
    ff.fff      ! [m]   Aircraft altitude above ground level
    ff.fff      ! [m/s] Aircraft velocity (groundspeed)
```

Parameter Meanings:

Symbol	Unit	Range	Description
Y_n	n/a	'G'	Output array, of type 'geometry'
SGI	n/a	'A' to 'Z'	Simulation Group Identifier
X_n	n/a	'M'	Input array, of type 'motion'
h	m	real	Aircraft altitude with respect to ground level
v	m/s	real	Aircraft velocity with respect to ground

Function:

The parameters are used, together with the target specifications and the $GENERAL parameters, to calculate the simulation geometry array. This array consists of, for each target and PRI, the range to the target as well as the elevation and azimuth angles.

Special Rules:

$GENERAL must be called before this function is called and at least one target must be defined. If aircraft motion deviation data is to be used, this data clearly must have been created beforehand.

Examples:

1.) `$GEOMETRY G1 A NULL 3000.00 100.00`

Calculates the simulation geometry for simulation group 'A' for an aircraft at an altitude of 3000 metres and a velocity of 100 metres per second. No motion deviation data is used (i.e. the aircraft is flying along the ideal flight path). The resulting data is written to array G1.

2.) `$GEOMETRY G1 A M1 3000.00 100.00`

Same as the first example but this time motion deviation data will be used from array M1. (Refer to the $MOTIONRND and $MOTIONSIN function calls.)

A.3.8 MATCHFILT

Call Structure:

```
$MATCHFILT Yn ! Create array Yn and place matched filter array in it
       Xn     ! [ ] Input waveform to be matched
    fff.ff    ! [ ] Hanning windowing factor ( 1.0 = no windowing )
```

Parameter Meanings:

Symbol	Unit	Range	Description
Y_n	n/a	'F'	Output array, of type 'Frequency'
X_n	n/a	'F'	Input array, of type 'Frequency'
W	n/a	≥ 0.0	Hanning windowing factor

Function:

A matched filter is created from array idix2 and stored in array idix1. The matched filter waveform is convoluted with a Hanning window with the specified constant.

Special Rules:

The input and output arrays much be in the frequency domain. If the Hanning window constant is 1.0 then no windowing is performed.

Examples:

1.) `$MATCHFILT F2 F1 0.08`

Create a matched filter from array F1 with a Hanning window constant of 0.08 and store this matched filter in array F2.

2.) `$MATCHFILT F1 F1 1.0`

Create a matched filter from array F1 with no Hanning windowing and store this matched filter back into array F1.

A.3.9 MOTIONRND

Call Structure:

```
$MOTIONRND Yn    ! Create array Yn and place random motion data in it
         A       ! [ ]   Use which GENERAL parameters ?
     ff.fff      ! [m]   Maximum allowed +/- variation along x-axis
     ff.fff      ! [m]   Maximum allowed +/- variation along y-axis
     ff.fff      ! [m]   Maximum allowed +/- variation along z-axis
     ff.fff      ! [deg] Maximum allowed +/- variation in squint angle
```

Parameter Meanings:

Symbol	Unit	Range	Description
Y_n	n/a	'M'	Output array, of type 'motion'
SGI	n/a	'A' to 'Z'	Simulation Group Identifier
$x_{m_{max}}$	m	real	Maximum allowed +/− variation along x-axis
$y_{m_{max}}$	m	real	Maximum allowed +/− variation along y-axis
$z_{m_{max}}$	m	real	Maximum allowed +/− variation along z-axis
$\alpha_{m_{max}}$	degrees	0.0 to 90.0	Maximum allowed +/− variation in squint angle

Function:

This function generates random aircraft motion errors and stores them in array idix1. The total variations in the four planes are from minus the given parameter to plus the given parameter. In the case of the x plane this would be from $-x_{m_{max}}$ to $+x_{m_{max}}$.

Special Rules:

None.

Examples:

1.) `$MOTIONRND M1 A 5.0 5.0 5.0 1.0`

Create an aircraft motion error array, with the deviations in the x, y and z planes being a maximum of +/- 5.0 m and the squint plane being a maximum of +/- 1 degrees, and store all this data in array M1.

A.3.10 MOTIONSIN

Call Structure:

```
$MOTIONSIN Yn    ! Create array Yn and place sinusoidal motion data in it
         A       ! [ ]          Use which GENERAL parameters ?
     ff.fff      ! [m]          Sine wave amplitude along x-axis
     ff.fff      ! [cycles/PRI] Sine wave frequency along x-axis
     ff.fff      ! [m]          Sine wave "DC" offset along x-axis

     ff.fff ff.fff ff.fff ! Same for y-axis
     ff.fff ff.fff ff.fff ! Same for z-axis
     ff.fff ff.fff ff.fff ! Same for squint angle
```

Parameter Meanings:

Symbol	Unit	Range	Description
Y_n	n/a	'M'	Output array, of type 'motion'
SGI	n/a	'A'-'Z'	Simulation Group Identifier
x_{m_a}	m	real	Sine wave amplitude along x-axis
x_{m_f}	cycles/PRI	real	Sine wave frequency along x-axis
x_{m_o}	m	real	Sine wave "DC" offset along x-axis
y_{m_a}	m	real	Sine wave amplitude along y-axis
y_{m_f}	cycles/PRI	real	Sine wave frequency along y-axis
y_{m_o}	m	real	Sine wave "DC" offset along y-axis
z_{m_a}	m	real	Sine wave amplitude along z-axis
z_{m_f}	cycles/PRI	real	Sine wave frequency along z-axis
z_{m_o}	m	real	Sine wave "DC" offset along z-axis
α_{m_a}	deg	-90.0 to 90.0	Sine wave amplitude in squint angle
α_{m_f}	cycles/PRI	real	Sine wave frequency in squint angle
α_{m_o}	deg	-90.0 to 90.0	Sine wave "DC" offset in squint angle

Function:

This function generates sinusoidal aircraft motion errors and stores them in array idix1. For example, the variation in the z plane will be defined by $z_{m_a} * \sin(2\pi z_{m_f} * \text{currentPRI}) + z_{m_o}$.

Special Rules:

None.

Examples:

1.) `$MOTIONSIN M2 A  1.0 0.1 0.0  1.0 0.1 0.0  1.0 0.1 0.0  0.1 0.1 0.0`

Create an aircraft motion error array, with the deviations in the x, y, z and squint planes being sine waves, and store all this data in array M1. In the x-plane, this will be a sine wave with an amplitude of 1.0 metres and a frequency of 0.1 cycles per PRI.

A.3.11 MULTIPLY

Call Structure:

```
$MULTIPLY Yn   ! Create array Yn and place product in it
      Xn1      ! [ ] First input waveform to be multiplied
      Xn2      ! [ ] Second input waveform to be multiplied
```

Parameter Meanings:

Symbol	Unit	Range	Description
Y_n	n/a	'T' or 'F'	Output array, of type 'Time' or 'Frequency'
X_{n_1}	n/a	'T' or 'F'	Input array 1, of type 'Time' or 'Frequency'
X_{n_2}	n/a	'T' or 'F'	Input array 2, of type 'Time' or 'Frequency'

Function:

The array idix2 is multiplied by array idix3 and the result is stored in array idix1.

Special Rules:

All three array types must be the same. The two input arrays must belong to the same $GENERAL group.

Examples:

1.) `$MULTIPLY T3 T1 T2`

Multiply array T1 by array T2 and store the resulting array in T3.

2.) `$MULTIPLY F1 F1 F1`

Multiply array F1 by itself and store the resulting array back in F1.

A.3.12 PULSEGEN

Call Structure:

```
$PULSEGEN Yn1 Yn2 ! Create arrays Yn1, Yn2 and place waveform data in them
        A         ! [ ]   Use which GENERAL parameters ?
    ff.fff        ! [ns]  Pulse rise time
    ff.fff        ! [ns]  Pulse fall time
    ff.fff        ! [ns]  Pulse width
    ff.fff        ! [V]   Peak amplitude
      ttt         ! [ ]   'CHIRP' or 'MONO' for chirp or monochromatic pulse
    ff.fff        ! [GHz] Chirp pulse bandwidth; ignored for monochromatic
```

Parameter Meanings:

Symbol	Unit	Range	Description
Y_{n_1}	n/a	'T'	Output array, of type 'time'
Y_{n_2}	n/a	'A'	Output array, of type 'amplitude'
SGI	n/a	'A' to 'Z'	Simulation Group Identifier
t_{rise}	ns	≥ 0.0	Pulse rise time
t_{fall}	ns	≥ 0.0	Pulse fall time
t_{length}	ns	> 0.0	Pulse length during peak amplitude
V_{peak}	V	$\neq 0.0$	Peak amplitude
n/a	n/a	'CHIRP'/'MONO'	Pulse type - chirp or monochromatic
f_{BW}	GHz	> 0.0	Bandwidth of chirp pulse

Function:

The given parameters, together with $GENERAL parameters, are used to generate a chirp or a monochromatic pulse. The amplitude envelope of the pulse is stored in array idix2 and the time domain IQ complex representation of the pulse in array idix1.

Special Rules:

$GENERAL must be called before this function is called. Remember to ensure that the pulse fits into the simulation space.

Examples:

1.) `$PULSEGEN T1 A1 A 2.0 1.0 10.0 5.0 CHIRP 0.05`

Generate a chirp waveform with an FM bandwidth of 0.05 GHz. The pulse has a rise time of 2 ns, a width of 10 ns, a fall time of 1 ns and a peak amplitude of 5 V. The pulse amplitude envelope is stored in array A1 and the IQ waveform in array T1.

2.) `$PULSEGEN T1 A1 A 2.0 1.0 10.0 5.0 MONO 0.05`

Same as above but a monochromatic pulse is generated with a frequency of 0.05 GHz.

A.3.13 RETURNL

Call Structure:

```
$RETURNL Yn    ! Create array Yn and place waveform radar return in it
       Xn1     ! [ ]   Simulation geometry array to be used
       Xn2     ! [ ]   Envelope of pulse to be transceived
    fff.ff     ! [GHz] Radar centre frequency
    fff.ff     ! [dB]  Antenna gains
    fff.ff     ! [dB]  Losses ( TxLoss + RxLoss + PropagationLoss )
    fff.ff     ! [m^2] Target cross-sectional area
    fff.ff     ! [m]   Range offset for simulation geometry
       ttt     ! [ ]   Pulse type - 'CHIRP' or 'MONO'
    fff.ff     ! [GHz] Chirp bandwidth (arb non-zero for MONO)
       ttt     ! [ ]   Antenna type - 'SIN' or 'NULL'
    fff.ff     ! [deg] Antenna elevation beamwidth
    fff.ff     ! [deg] Antenna azimuth beamwidth
    fff.ff     ! [deg] Antenna elevation angle (down is negative)
    fff.ff     ! [deg] Antenna squint angle (backward is positive)
```

Parameter Meanings:

Symbol	Unit	Range	Description
Y_n	n/a	'T'	Output array, of type 'Time'
X_{n_1}	n/a	'G'	Input array 1, of type 'Geometry'
X_{n_2}	n/a	'A'	Input array 2, of type 'Amplitude'
f_{centre}	GHz	≥ 0.0	Radar centre frequency
G_{ant}	dB	real	Antenna gains
L_{txrx}	dB	real	Losses (TxLoss+RxLoss+PropagationLoss)
a_{rcs_t}	m^2	> 0.0	Target cross-sectional area
R_o	m	real	Range offset for simulation geometry
n/a	n/a	'CHIRP'/'MONO'	Pulse type - chirp or monochromatic
f_{BW}	GHz	> 0.0	Bandwidth of linear chirp pulse
n/a	n/a	'SIN'/'NULL'	Antenna gain type - sinusoidal or omni.
θ_e	deg	0.0 to 180.0	Antenna elevation beamwidth
θ_a	deg	0.0 to 180.0	Antenna azimuth beamwidth
α_e	deg	-90.0 to 90.0	Antenna elevation angle
α_a	deg	-90.0 to 90.0	Antenna azimuth angle

Function:

The pulse described by idix3, string and float7 is transmitted from the radar to the target at a range and angle described within the array idix2. The appropriate gains, phases and shifts are applied and the returning waveform is stored in array idix1. If an antenna type is specified, the pulse is passed through the relevant antenna gain pattern during transmission and reception.

Special Rules:

This function is meant to be used in conjunction with the $ENDLOOP function to create a processing loop. The simulation geometry and the pulse amplitude arrays must belong to the same $GENERAL group. Processing loops may not be nested.

Examples:

1.) `$RETURNL T2 G1 A1 6.0 7.0 5.0 1.0 500.0 CHIRP 0.1 SIN 8.0 6.0 -9.0 0.0`

The pulse A1, which is a chirp with an FM bandwidth of 0.1 GHz, is transmitted. The antenna gain of 7 dB is applied during transmission and reception with losses of 5 dB. The radar has a centre frequency of 6 GHz. For the purposes of simulation the targets are brought closer by 500 m. The pulse is passed through an antenna with a sin(x)/x gain

pattern with 8 and 6 degree beamwidths and pointing down at a depression angle of 9 degrees. The received waveform is stored in array T2 for further processing.

A.3.14 TARGET

Call Structure:

```
$TARGET  A
    ff.fff    ! [m] x direction offset from mid path
    ff.fff    ! [m] y ground range at closest approach
    ff.fff    ! [m] z height above ground level
```

Parameter Meanings:

Symbol	Unit	Range	Description
SGI	n/a	'A' to 'Z'	Simulation Group Identifier
x_t	m	real	x direction offset from mid path
y_t	m	real	y ground range at closest approach
z_t	m	real	z height above ground level

Function:

A point target is defined at the given simulation geometry co-ordinates. The target will belong to the simulation of id 'char'.

Special Rules:

$GENERAL with the relevant 'id' must be called before any targets with the id 'id' are specified. Each further call to $TARGET will simply add another target to the simulation geometry.

Examples:

1.) `$TARGET A 0.00 5000.00 50.00`

Set up a target which is placed in the middle of the x flight axis, is 5000 metres away along the Y axis and is 50 metres above ground level.

A.3.15 WRITE

Call Structure:

```
$WRITE        ! No output array
      Xn      ! [ ] Array to be written to file
    ttt1      ! [ ] File type, ASCII or binary ( 'ASC' or 'BIN' )
    ttt2      ! [ ] Write mode, append or overwrite ( 'APP' or 'OVR' )
    ttt3      ! [ ] Valid file name for output file
```

Parameter Meanings:

Symbol	Unit	Range	Description
X_n	n/a	'A', 'D', 'F', 'G', 'M' or 'T'	Input array, of any type
n/a	n/a	'ASC' or 'BIN'	File type, ASCII or Binary
n/a	n/a	'APP' or 'OVR'	File mode, Append or Overwrite
n/a	n/a	text	File name, valid for current OS

Function:

The array idix1 is written to file string3. The data is written in binary or ASCII format as specified by string1. The data is written to a completely new file or is appended to an existing file as specified in string2. This appending is performed only to files created during the current simulation. Thus, whether `APP` or `OVR` is selected, any file of the same name existing before the simulation begins is deleted, and a new file is created.

Special Rules:

The length of the array to be printed and the number of columns to be used is calculated automatically by the software.

Examples:

1.) `$WRITE T1 ASC OVR FOO.ASC`

Print the contents of the array T1 to the plain ASCII file FOO.ASC, overwriting any previous data in FOO.ASC

2.) `$WRITE F1 BIN APP BAR.BIN`

Print the contents of the array F1 to the raw binary file BAR.BIN, appending to the data already existing in BAR.BIN.

Appendix B

Contents of Accompanying Computer Disk

This appendix describes the contents of the $3\frac{1}{2}$ inch computer disk. There are eight `ZIP` files on the disk as follows:

`MATHCAD.ZIP`	All MathCAD sheets used in thesis.
`SIM1.ZIP`	Command file and TIFF files for sample simulation 1 in Chapter 5.
`SIM2.ZIP`	Command file and TIFF files for sample simulation 2 in Chapter 5.
`SIM3.ZIP`	Command file and TIFF files for sample simulation 3 in Chapter 5.
`SIM4.ZIP`	Command file and TIFF files for sample simulation 4 in Chapter 5.
`SOURCE.ZIP`	All source code for RadSim.
`TESTS.ZIP`	Command files and results of all tests performed in Chapter 4.
`THESIS.ZIP`	All LaTeX source, pictures and images.

The contents of each of these `ZIP` files are now discussed in more detail in the next eight sections. It is assumed that the reader is familiar with `ZIP` files and the methods involved in decompressing them.

B.1 Contents of MATHCAD.ZIP

This ZIP file contains all of the MathCAD sheets that were designed for the testing of the simulator, as well as for other purposes. The files are:

ANTENNA.MCD	– Plot of antenna gain pattern for Figure 3.3.
SIMCUTS.MCD	– Magnitude image before azimuth compression.
SIM1CUT.ASC	– ASCII data for range bin cut for sample simulation one.
SIM2CUT.ASC	– ASCII data for range bin cut for sample simulation two.
TEST1.MCD	– Sheet for test one, that is Section 4.2.
TEST2.MCD	– Sheet for test two, that is Section 4.3.
TEST3.MCD	– Sheet for test three, that is Section 4.4.
TEST4.MCD	– Sheet for test four, that is Section 4.5.
TEST5.MCD	– Sheet for test five, that is Section 4.6.
TEST6.MCD	– Sheet for test six, that is Section 4.7.
TEST7.MCD	– Sheet for test seven, that is Section 4.8.
TEST8.MCD	– Sheet for test eight, that is Section 4.9.
A1.ASC	– ASCII dump of array A1 from the tests section.
⋮	– and so on . . .
T5.ASC	– ASCII dump of array T5 from the tests section.

B.2 Contents of SIM1.ZIP

This ZIP file contains the command file and resulting images for the first of the four sample simulations as follows:

D1.TIF	– Magnitude image of azimuth compressed results.
D1POW.TIF	– Magnitude image before azimuth compression.
SIM1.CMD	– RadSim command file for the simulation.

B.3 Contents of SIM2.ZIP

This ZIP file contains the command file and resulting images for the second of the four sample simulations as follows:

D1.TIF	– Magnitude image of azimuth compressed results.
D1POW.TIF	– Magnitude image before azimuth compression.
SIM2.CMD	– RadSim command file for the simulation.

B.4 Contents of SIM3.ZIP

This ZIP file contains the command file and resulting images for the third of the four sample simulations as follows:

D1.TIF	– Magnitude image of azimuth compressed results.
D1POW.TIF	– Magnitude image before azimuth compression.
SIM3.CMD	– RadSim command file for the simulation.

B.5 Contents of SIM4.ZIP

This ZIP file contains the command file and resulting images for the fourth of the four sample simulations as follows:

D1.TIF	– Magnitude image of azimuth compressed results.
D1POW.TIF	– Magnitude image before azimuth compression.
SIM4.CMD	– RadSim command file for the simulation.

B.6 Contents of SOURCE.ZIP

This ZIP file contains all of the source code for RadSim. The files that make up the ZIP archive may be divided into two parts, namely the RadSim engine source code and the

RadSim simulation function source code as follows:

Engine:

COMMON.H and COMMON.C	– Simulation array tracking functions.
ENGINE.H and ENGINE.C	– main() function and central core of simulator.
GLOBAL.H and MACROS.H	– Global structures, constants and macros.
LINKLIST.H and LINKLIST.C	– Type-independent linked list and related functions.
MESSAGE.H and MESSAGE.C	– Information, Warning and Error message database.

Simulation Functions:

A2D.H and A2D.C	– A2D function.
ADD.H and ADD.C	– ADD function.
AMPLIFY.H and AMPLIFY.C	– AMPLIFY function.
ENDLOOP.H and ENDLOOP.C	– ENDLOOP function.
FFT.H and FFT.C	– FFT function.
GENERAL.H and GENERAL.C	– GENERAL function.
GEOMETRY.H and GEOMETRY.C	– GEOMETRY function.
MATCHFIL.H and MATCHFIL.C	– MATCHFILT function.
MOTIONRN.H and MOTIONRN.C	– MOTIONRND function.
MOTIONSI.H and MOTIONSI.C	– MOTIONSIN function.
MULTIPLY.H and MULTIPLY.C	– MULTIPLY function.
PULSEGEN.H and PULSEGEN.C	– PULSEGEN function.
RETURNL.H and RETURNL.C	– RETURNL function.
TARGET.H and TARGET.C	– TARGET function.
WRITE.H and WRITE.C	– WRITE function.

B.7 Contents of TESTS.ZIP

This ZIP file contains all of the MathCAD sheets that were designed for the testing of the simulator, as well as for other purposes. The files are:

TEST1.CMD	– RadSim command file for test one, that is Section 4.2.
TEST2.CMD	– RadSim command file for test two, that is Section 4.3.
TEST3.CMD	– RadSim command file for test three, that is Section 4.4.
TEST4.CMD	– RadSim command file for test four, that is Section 4.5.
TEST5.CMD	– RadSim command file for test five, that is Section 4.6.
TEST6.CMD	– RadSim command file for test six, that is Section 4.7.
TEST7.CMD	– RadSim command file for test seven, that is Section 4.8.
TEST8.CMD	– RadSim command file for test eight, that is Section 4.9.
A1.ASC	– ASCII dump of array A1 created by RadSim.
⋮	– and so on . . .
T5.ASC	– ASCII dump of array T5 created by RadSim.

B.8 Contents of THESIS.ZIP

This ZIP file contains all of the LaTeX source for this document including all pictures and images as follows:

LaTeX Source:

APPX1.TEX	– Appendix 1.
APPX2.TEX	– Appendix 2.
CHAP1.TEX	– Chapter 1.
CHAP2.TEX	– Chapter 2.
CHAP3A.TEX	– Chapter 3, part A.
CHAP3B.TEX	– Chapter 3, part B.
CHAP4.TEX	– Chapter 4.
CHAP5.TEX	– Chapter 5.

`CHAP6.TEX` – Chapter 6.

`MACROS.TEX` – LaTeX macros and command definitions.

`THESIS.TEX` – Main LaTeX document, includes all others.

Pictures:

`ALGORITH.PIC` – Figure 2.2, proposed simulation algorithm.

`AXES.PIC` – Figure 2.4, Cartesian axes definition.

`ENGINE.PIC` – Figure 3.2, flowchart of function `main()`.

`GEOM.PIC` – Figure 2.3, classic SAR geometry.

`OVERALLD.PIC` – Figure 3.1, two main parts of RadSim design.

`RADBLOCK.PIC` – Figure 2.1, SAR hardware block diagram.

`SLANT.PIC` – Figure 2.5, simplified slant range geometry.

Images:

`SIM1A,B,C,D,E.WMF` – Images and MathCAD plots for sample simulation one.

`SIM2A,B,C,D,E.WMF` – Images and MathCAD plots for sample simulation two.

`SIM3A,B,C,D.WMF` – Images and MathCAD plots for sample simulation three.

`SIM4A.WMF` – Images and MathCAD plots for sample simulation four.

`TESI1A,B,C,D.WMF` – MathCAD sheets and plots for Section 4.2.

`TESI2A,B,C,D.WMF` – MathCAD sheets and plots for Section 4.3.

`TESI3A,B,C,D.WMF` – MathCAD sheets and plots for Section 4.4.

`TESI4A,B,C,D.WMF` – MathCAD sheets and plots for Section 4.5.

`TESI5A,B,C.WMF` – MathCAD sheets and plots for Section 4.6.

`TESI6A,B,C,D,E.WMF` – MathCAD sheets and plots for Section 4.7.

`TESI7A,B,C.WMF` – MathCAD sheets and plots for Section 4.8.

`TESI8A,B,C,D.WMF` – MathCAD sheets and plots for Section 4.9.

Bibliography

[1] **Aitken P. and Jones B.,** "Teach Yourself C in 21 Days," SAMS Publishing, 1992.

[2] **American National Standard for Information Systems (ANSI),** "Programming Language C," electronic version from anonymous source, no date shown.

[3] **CAE Soft Corporation,** "CAE Soft - Radar Analysis / Simulation Tool (CS-RAST) - User's Manual, Revision F1 04-June-1993," CAE Soft Corporation, 1993.

[4] **comp.lang.c Internet news group,** "Frequently Asked Questions," electronic version, 16 Apr. 1994.

[5] **comp.lang.c Internet news group,** private electronic communication discussing programming practice, portability and efficiency spanning Nov. 1993 to Aug. 1996.

[6] **Hassenpflug G.,** private communication discussing RadSim bugs and recommendations for improvements spanning 1995 to 1997.

[7] **Horrell J.,** "Basic SAR Geometry and Theory," *Radar Signal Processing Lecture Notes*, University of Cape Town, Sep. 1994.

[8] **Horrell J.,** private communication discussing requirements, implementation methods, simulation algorithms, RadSim bugs and recommendations for improvements spanning 1993 to 1997.

[9] **Horrell J. and Inggs M.R.,** "The SASAR Ground Processor Specification, Document Version 1.2," *Technical Report RRSG1778:96,* UCT Radar Remote Sensing Group, 20 Sep. 1996.

[10] **Hovanessian S.A.,** "Radar System Design and Analysis," Artech House Inc., U.S.A., 1984.

[11] **Inggs M.R.,** "Basic Radar Definitions," *Radar Systems Lecture Notes*, University of Cape Town, 1994.

[12] **Keay L.,** "SAR Simulator," *Undergraduate Thesis*, University of Cape Town, Nov. 1993.

[13] **Keay L.,** private communication discussing design methodology behind a radar simulator and the problems experienced during her work, Nov. 1993.

[14] **Langefelder R.,** private communication discussing RadSim bugs and recommendations for improvements spanning 1996 to 1997.

[15] **Langman A.,** private communication discussing compilation of code using GNU C under MSDOS, Linux and SunOS spanning 1993 to 1994.

[16] **Lord R.,** private communication discussing RadSim bugs and recommendations for improvements spanning 1996 to 1997.

[17] **Morrison N.,** "Introduction to Fourier Analysis," *Lecture Notes in Applied Mathematics*, University of Cape Town, Jul. 1991.

[18] **Nyquist E. and Henricson M.,** "Programming in C++, Rules and Recommendations," Ellemtel Communication Systems Laboratories, Sweden, 1992.

[19] **Oppenheim A.V. and Schafer R.W.,** "Discrete-Time Signal Processing," Prentice Hall, 1989.

[20] **Press W.H., Vetterling W.T., Teukolsky S.A. and Flannery B.P.,** "Numerical Recipes in C - The Art of Scientific Computing," Cambridge University Press, 1992.

[21] **Stremler F.G.,** "Communication Systems," Addison Wesley, 1990.

[22] **Topham D.W.,** "Portable UNIX," Wiley, 1992.

[23] **Zsilavecz G.,** private communication discussing compilation of code using cc and CC under HPUX 9 and HPUX 10, 10 Jul. 1997.