

Design and Implementation of a Graphical User Interface for the Flexible, Extensible Radar and Sonar Simulator

Prepared by:

Poovendren Govender



A dissertation submitted to the Department of Electrical Engineering,
University of Cape Town, in fulfillment of the requirements for the degree
of Bachelor of Science in Engineering

21 October 2008

Supervisor: Yoann Paichard

Abstract

This project describes the design, implementation and testing of a Graphical User Interface for the Flexible Extensible Radar Simulator which was developed by the RRSg at UCT.

Prior to the actual design and implementation, research relevant to the GUI development needed to be examined. Research included a basic understanding of radar systems, the FERS simulator and an already existing graphical radar simulator, Sarsim2. The software tools used to complete this project included Qt, OpenGL and QWT.

The design phase involved determining the functionality of the GUI and separating the GUI into smaller manageable components. These were the main window, pulse compressions, 3D visualization and 2D graphs of results. This allowed for modular design of the GUI.

The implementation phase involved realizing the functionality set out in the design phase by using the software tools and relevant research.

Once implementation was complete the testing of the GUI was carried out. Any inconsistencies found at this stage were either corrected or included as future work.

The GUI created utilizes the FERS simulator to model both bistatic and monostatic radars and a maximum of 6 isotropic targets. The radar platforms and targets are displayed on a 3D plot created with OpenGL. The user is able to rotate, zoom and move the 3D coordinate system. When simulation is complete the user has the option of viewing basic 2D graphs of the results produced by the FERS simulator.

Acknowledgements

The author would like to thank the following people:

- Joan Paichard, my thesis supervisor for his guidance and support.
- Michael Inggs for providing the author with this project
- Marc Brooker who provided much insight into the functioning of the FERS simulator.
- Telkom which has paid for the author to attend the University of Cape Town for four years.
- The author's good friends Anndrea Naidu, Clement Clementson, Hilary Chisepo, Jared Baboo, Lorin Zaal, Neehal Mooruth, Nicolas Ip Cho, Mayuresh Kulkarni, Shannon Anne Boonzaier and Yuven Naidoo for the support throughout the years at UCT .
- The author's friends who were present in spirit, Kreben, Chervin, Ronell, Akash, Wesley, Yoganathan, Ruveshan, Annil and Kerusha.
- Last but certainly not least, the author's family for their love and encouragement throughout the years at UCT.
- The author would also like to acknowledge some of the Open Source software which was used in preparing this thesis: Kubuntu, OpenGL, QWT, Qt and LATEX.

Contents

Abstract	i
Acknowledgements	ii
Nomenclature	x
1 Introduction	1
1.1 Background	1
1.2 Projects requirements	1
1.3 Requirements Review	2
1.4 Project overview	3
1.4.1 Relevant research and software tools	3
1.4.2 Design	4
1.4.3 Implementation	5
1.4.4 Testing, Evaluation and Debugging	5
1.4.5 Command Summary	6
1.4.6 Conclusions	6
1.4.7 Future Work and Recommendations	7
2 Relevant research and software tools used	8
2.1 Radar Systems	8
2.1.1 Basic theory of radars	8
2.1.2 Classification of radars	8
2.1.3 Important radar concepts	9
2.2 FERS	10
2.2.1 Background	10
2.2.2 Functionality	11
2.2.3 FERS simulation definition	11
2.2.4 Results of the FERS simulator	13

2.3	User requirements	14
2.4	Existing GUI	14
2.5	SARSIM2	16
2.5.1	Features	16
2.5.2	SARSIM2 Input window	16
2.5.3	SARSIM2 3D coordinate system	18
2.6	Pulse compression	18
2.6.1	Monochrome	18
2.6.2	Chirp	19
2.6.3	Step frequency	19
2.7	Qt4.3 Designer	19
2.8	OpenGL	20
2.8.1	Objects	20
2.8.2	Transformations	21
2.8.3	Glut libraries	21
2.9	QWT	21
3	Design	23
3.1	System overview	24
3.2	Functional requirements	24
3.3	Additional features	26
3.4	System components	26
3.4.1	Input Window	26
3.4.2	Pulse compression	27
3.4.3	3D visualization	27
3.4.4	Simulation	27
3.4.5	Graphs	27
3.5	Component Diagram	28
4	Implementation	29
4.1	Input window	29
4.2	Pulse compression	29
4.2.1	Monochrome	29
4.2.2	Chirp	31
4.2.3	Step frequency	31
4.3	3D visualization	31

4.3.1	Coordinate system	31
4.3.2	Plotting objects	32
4.3.3	Manipulating the visualization	33
4.4	Running the Simulation	34
4.5	Drawing Graphs of results	35
4.5.1	Power vs Time	35
4.5.2	Doppler vs Time	35
4.5.3	Power vs Range	35
4.5.4	Doppler vs Range	36
5	Testing, Evaluation and Debugging	37
5.1	Main input window	37
5.2	Pulse compression	37
5.3	FERS definition script	37
5.4	3D Visualization	39
5.4.1	Plotting	39
5.5	Systems Testing	40
6	Command summary	41
6.1	Main Window	41
6.2	Simulation Inputs	42
6.2.1	Parameters	42
6.2.2	Pulse Type	43
6.2.3	Platforms	44
6.2.4	Working directory and results buttons	46
6.3	A simple example	46
6.3.1	Simulation setup	46
6.3.2	3D visualization	49
6.3.3	Simulation	50
6.3.4	Results	51
7	Conclusions	55
8	Future work and recommendations	57
A	Matlab source code for pulse compressions	60
A.1	Chirp	60
A.2	Step frequency	61

B	Software source code	62
B.1	Source header files	62
B.1.1	Fersgui	62
B.1.2	Glwidget	64
B.1.3	Fersout	66
B.2	Build notes	67

List of Figures

1.1	Overview of System	4
2.1	The Doppler Effect	10
2.2	FERS definition script	12
2.3	XML response	13
2.4	Interface for the simulator's parameters	15
2.5	SARSIM2 main window	17
2.6	SARSIM2 3D coordinate system	18
2.7	Qt4.3 Designer	20
2.8	Example of a QWT plot	22
3.1	System overview	24
3.2	Component diagram	28
4.1	Input window	30
4.2	3D coordinate system	32
4.3	Object representations	32
4.4	Objects' waypoints	33
5.1	Plotting test results	40
6.1	Main window	42
6.2	Bistatic platforms	46
6.3	Working directory	46
6.4	System parameters	47
6.5	Pulse settings	47
6.6	Monostatic platform settings	48
6.7	Target settings	49
6.8	3D visualization of system	50
6.9	Simulation complete window	51
6.10	Graph of Power vs Time	52

6.11 Doppler vs Time	52
6.12 Power vs Range	53
6.13 Doppler vs Range	54

List of Tables

5.1	Testing and results for main input window	38
5.2	Analysis of Plot results	40
6.1	Target setup	48

Nomenclature

API : Application Programming Interface

Azimuth : Angle in a horizontal plane, relative to a fixed reference, usually north or the longitudinal reference axis of the aircraft or satellite.

CSV : Comma Separated Values

Doppler : A shift in the radio frequency of the return from a target or object as a result of the object's radial motion relative to the radar.

GUI : Graphical User Interface

HDF5 : Hierarchical Data Format

IDE : Integrated Development Environment

Isotropic : Non-directional

OpenGL : API for writing 3D applications

PRF : Pulse repetition frequency

PRI : Pulse repetition interval

Qt : Toolkit for developing graphical interfaces

QWT : Qt Widgets for Technical Applications

Radar : Actual energy-transmitting device.

Range : The radial distance from a radar to a target

RCS : Radar cross section

RRSG : Radar Remote Sensing Group

UCT : University of Cape Town

XML : Extensible Markup Language

Chapter 1

Introduction

1.1 Background

RADAR and SONAR are both remote sensing systems with important military, scientific and commercial applications. RADAR sends out electromagnetic waves whilst SONAR transmits sound waves. In both systems, these waves return echoes from objects that allow important properties of the objects to be determined, namely shape, size, speed and distance. RADAR is mostly used for ground or atmospheric observations whilst SONAR is used for navigation and measurement under water. [3]

FERS [5] is the Flexible, Extensible Radar and Sonar Simulator. It aims to simulate the behavior of next generation radar and sonar systems. FERS takes an XML description of a scene and generates the waveform that the receivers will see.

FERS works with an arbitrary number of receivers and transmitters (monostatic and multistatic), works with continuous wave and pulsed radar systems, and can simulate a radar using any pulse shape. The simulator currently simulates the amplitude, phase, Doppler and noise effects of a radar system. [4]

1.2 Projects requirements

This project is aimed at designing and implementing a Qt [11] based GUI for the Flexible Extensible Radar Simulator(FERS) developed by the Radar Remote Sensing Group. Qt is a standard framework for developing high-performance cross-platform applications. [12] In 2007 a Qt based GUI was developed and aimed at educational use of the simulator for radar and sonar. Although successful, a number of new features and improvements must be included. [6]

1.3 Requirements Review

In order to fulfill the requirements of this project possessing a basic understanding of radar systems is essential. Most important is that this project is to design and implement a GUI for a radar simulator and not the simulator itself and therefore an extensive knowledge of radar systems is not a necessity. Once the relevant knowledge is gained an understanding of how the FERS simulator works with regards to the format and content of the input script and the generated output files are needed. FERS reads in an XML [8] script of the scene to be simulated and outputs the responses in XML, CSV [10] and HDF5 [22] formats. From this basic description one can infer that the user will enter the required data in the GUI, an XML script will be generated from this data and the FERS simulator will be run with this script.

The main characteristic of the GUI that must be kept in mind is that it will be used for educational purposes. The GUI should therefore be intuitive to any user who has a basic knowledge of radar systems. Limiting the possible inputs to the FERS simulator is needed to create a GUI that can be used for such a purpose.

User requirements are another vital component in the development of the GUI. Persons with extensive experience in the field will be interviewed and their suggestions incorporated into the design. From brief meetings prior to the beginning of this report it has been mentioned that a visualization of the scene simulated and plots of the responses would be an advantage to the GUI being developed. Researching a suitable way to create this visualizations and plots would be the next step.

Next an already existing radar simulator with a graphical front-end should be examined to determine preferable characteristics with regards to screen layout, visualizations and functionality. SARSIM2[7] was identified as a suitable candidate.

Once all the required research is completed the design and implementation of the GUI can begin. These phases will probably take the most amount of time and therefore to reduce the risk of running into delays the design should be reasonably well planned but at the same time not restrict the GUI in terms of functionality.

Testing is the final phase of the project. Upon completion, thorough testing of the GUI should be carried out.

The finer details of the above requirements will be clarified when the user requirements are established. The deliverables of the project are a working GUI for the FERS simulator that implements the requirements set out and a project report which documents the research, design, implementation and testing carried out.

1.4 Project overview

1.4.1 Relevant research and software tools

Chapter 2 presents the research and the software tools needed to fulfill the requirements of this project. The relevance of each research topic is discussed and a description of each software tool is also given.

The chapter begins by examining the aspects of radar systems to better understand the requirements of the project as well as the workings of the FERS simulator. Understanding the format of the FERS XML script and the output files that the FERS simulator creates are vital components in the creation of the GUI.

User requirements are essential to determine the functionality of the GUI. The most important user requirements is that the GUI must be able to simulate monostatic and bistatic radar configurations, contain simple pulse compression and display a visualization of the scene to be simulated. The user requirements also determined the further research that was required.

A user interface which currently exists for the FERS simulator[6] was examined for functionality and user-friendliness. The interface displays input windows for each section of the simulation parameters and uses this input to create the FERS definition script. It was determined that the new GUI could be more user-friendly by providing all user inputs on a single window.

Sarsim2 [7] was researched since it is a graphical radar simulator which has been around for many years and is well documented. It was found that Sarsim2 provided immense functionality but was much too complex to be used for educational purposes, even so it was still sufficient to be a benchmark to the GUI being developed.

The pulse compressions(monochrome, chirp and step frequency) which are built into the GUI were researched since their models needed to be understood for their implementations.

Qt4.3 designer was chosen to create the GUI front end with. As described on the Trolltech website, Qt is the standard framework for developing high-performance cross-platform applications. Qt provides single-source portability across Microsoft Windows, Mac OS X, Linux and all major commercial Unix variants. Qt is fully object-oriented, easily extensible, and allows true component programming. [12]Also Qt can be used to develop software that is licensed under the GNU GPL or a similarly recognized open source license which is in keeping up with the standards set out by the FERS simulator.[13]

The 3D visualization was coded using the OpenGL libraries which were readily available with Qt4.3. OpenGL is the perfect environment for developing 3D graphics applications. It is stable, easy to use and well documented. [14]

The QWT(Qt Widgets for Technical Applications) library was chosen to draw 2D plots of the results produced by the FERS simulator. Using this class is much simpler than any

other since it is built around the existing Qt4 library. [16]

1.4.2 Design

Chapter 3 details the design process that was followed during the project. Firstly a high level abstraction of the interactions between the user, the GUI and FERS is developed. This is shown in Figure 1.1

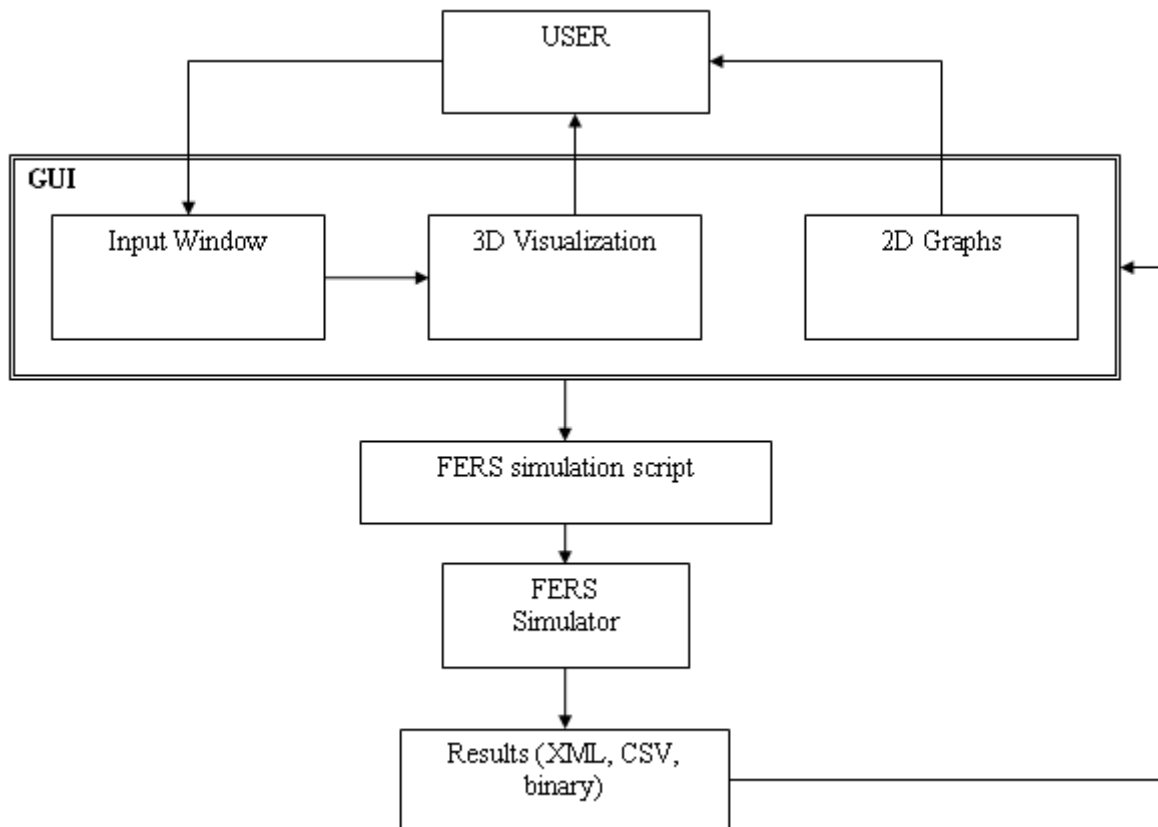


Figure 1.1: Overview of System

Secondly the functionality that needed to be implemented in the GUI was identified. Once this functionality was finalized the system was modularized so that the components could be individually developed according to their functionality.

The GUI was divided into 5 components:

- Input Window
- Pulse compression
- 3D visualization
- Simulation
- Graphs

It was determined that the GUI would be more user-friendly if all possible inputs were displayed on the main window rather than the user being bombarded with new windows requiring input. The vicinity to display radars and targets was constrained to 100km in the x,y and z directions on the 3D visualization.

1.4.3 Implementation

Chapter 4 describes the implementation of the GUI. After the main components were identified and their functionality finalized in the design phase, they were implemented using the software tools mentioned in chapter 2. The components were implemented individually but concurrently and integrated later on.

- Qt was used to create the GUI front-end. All inputs were displayed on the main window by utilizing tabbed windows.
- The pulse compressions were implemented by converting equivalent Matlab code to C++.
- The 3D visualization was implemented using OpenGL.
- The simulation was run by invoking the FERS simulator within the C++ code by using the QProcess class which is provided with Qt.
- The graphs of the results were implemented using the QWT libraries.

All implementation code can be found on the accompanying CD.

1.4.4 Testing, Evaluation and Debugging

Chapter 5 describes the procedures used to test and evaluate the GUI during and after the implementation phase. The main aspects of the GUI that required detailed testing were the pulse compression, FERS input script and 3D visualization. The pulses were tested by comparing output to equivalent code written in Matlab. It was found that the pulses outputted the correct values apart from a few rounding off errors. The FERS input scripts created by the GUI were tested continually throughout the project for consistency by running the script directly with the FERS simulator. Numerous errors were encountered, these included incorrect values being written and parameters being left out. The 3D visualization was tested for performance and functionality. During tests it was found that zooming in too much caused the GUI to crash. This was corrected by limiting the maximum and minimum zoom.

1.4.5 Command Summary

Chapter 6 gives a detailed description of the functionality that the GUI provides. The main input window, 3D visualization and 2D plot windows are also discussed in detail. Finally a simple step by step example which utilizes the GUI is followed through.

Monostatic and bistatic radars can be modeled with the GUI and a maximum of 5 waypoints and 5 fixed Rotation points can be specified for either. A maximum of 6 targets with a limit of 5 waypoints each are allowed. All targets and radars are considered to be isotropic.

The main window is divided into 2 parts, the input section on the left and the 3D visualization on the right.

Input includes the basic system parameters(start time, end time, system frequency etc.), the choice of radar platform(bistatic or monostatic) and the pulse to be transmitted(monochrome, chirp, step or user defined). Depending on the previous choices the user is only allowed to edit input that would be relevant to his/her choice i.e monostatic or bistatic radars. Finally one or more targets can be inputted. Targets and radars can be specified with a maximum distance of 100km in the x,y and z directions.

The choice of pulses to be transmitted includes but is not limited to monochrome, chirp and step. The user may choose a pulse file of his/her choice by choosing the browse option.

The 3D visualization contains a 3D plot which can be rotated using the left-mouse button, zoomed using the mouse scroll and moved around the screen using the right-mouse button. If the user clicks the centre-mouse button the plot is reset to the original zoom and position. A target is represented by a sphere whilst transmitter and receiver are represented by a red and green antenna symbol respectively. When a radar or target is specified it is plotted on the 3D visualization at the coordinates of its 1st waypoint. The other waypoints are also plotted and joined by lines to indicate the linear interpolated path of the object.

When simulation is complete 2D plots of 'power vs time', 'Doppler vs time', 'power vs range' and 'Doppler vs range' can be displayed. Units for Doppler is Hertz, time is seconds, range is metres and power is in Decibel watts

This chapter is concluded with a simple example that tests a pulsed radar system comprised of a monostatic radar, two moving targets and one stationary target..

1.4.6 Conclusions

Chapter 7 identifies the main objectives set out at the beginning of this project and compares them to the functionality of the completed GUI.

The objectives of this project were to design and implement a Qt based GUI for the FERS simulator to be used for educational purposes. The GUI has met the requirements of

the project in most aspects. The only function that was not implemented due to time constraints is the ability to import FERS definition files.

This GUI is simple and intuitive to students with a basic background of radar systems therefore making it ideal for students who are new to the subject of radars.

1.4.7 Future Work and Recommendations

Chapter 8 identifies the improvements which may be implemented in the future as well as features that were not implemented due to time constraints.

Adapting the GUI for advanced users is considered as the most important aspect for future work since this would take advantage of what the FERS simulator has to offer.

Chapter 2

Relevant research and software tools used

In order to understand the scope of this project a reasonable understanding of Radar systems was needed. Once this was acquired the functionality of the FERS simulator could be better understood. After establishing the user requirements further research and the software tools needed to fulfill these requirements were decided on. An already existing radar simulator, SARSIM2, was investigated for possible functionality that might be included in the FERS GUI.

2.1 Radar Systems

This section outlines the relevant knowledge of radar systems needed to utilize the FERS simulator.

2.1.1 Basic theory of radars

Radar is an acronym for radio detection and ranging [1]. Radar uses electromagnetic waves to detect objects and their characteristics from a remote location. A portion of the transmitted signal bounces off an object and is radiated in all directions. The energy that is radiated back to the receiver is processed to detect the presence of the target and extract its characteristics which include the distance, speed, size and direction.

2.1.2 Classification of radars

Radars can be classified according to the positions of the receiving and transmitting antennas and the nature of the transmitted signal. [19] These are outlined below:

- Monostatic : Transmit and receive station at same location

- Bistatic : Transmit and receive stations at different locations
- Multistatic : Many transmitting and receiving stations.
- Continuous wave : Transmits continuously
- Pulsed : Transmits pulses at equal intervals

2.1.3 Important radar concepts

PRF and PRI

The pulse repetition interval (PRI) of a pulsed radar system is the rate at which pulses are transmitted. The pulse repetition frequency (PRF) is the inverse of the PRI.

Non-ambiguous range

The non-ambiguous range of radar is the maximum range at which a received signal reflected of a target can still be matched with the correct transmit signal without ambiguity. The non-ambiguous range is related to the speed of propagation and the PRF of the system. Equation 2.1 shows this relationship. [19]

$$R_{na} = PRI * c/2 \quad (2.1)$$

The radar equation

The radar equation defines the power received at the receiver from a transmitted signal that is reflected of a target. [21]

For the bistatic case:

$$P_r = \frac{P_t G_t G_r L_t L_r \sigma_b \lambda^2}{(4\pi)^3 R_{kj}^2 R_{ik}^2} \quad (2.2)$$

where P_t is the power transmitted, G_t and G_r the gains of the transmitting and receiving antennas respectively, L_t and L_r the losses in the transmitting and receiving systems respectively, σ_b the bistatic radar cross section for transmitter and receiver angles, λ the wavelength of the system, R_{kj} the range between target k and transmitter j, and R_{ik} the range between target k and receiver i.

The monostatic case is just a simplified version of the bistatic case where $R_{kj} = R_{ik}$.

Range resolution

This is a limitation in the receiving system to discriminate between separate echoes only if the difference in their delays is greater than the transmitted pulse width. The range

resolution is $\frac{Tc}{2}$, where T is the pulse width and c is the propagation speed of the system. [19]

Doppler Effect

Radars use the Doppler Effect to extract information on a target's radial velocity. The principle behind the Doppler Effect is that depending on the speed that a target is moving, it will change the frequency of the wave reflected back. This is illustrated in the figure 2.1 [27]. The car is traveling towards a point very fast with its hooter on. The hooter will seem to decrease in pitch as the car passes by this point. This occurs because the moving car compresses the sound wave ahead of it so that the wave peaks come closer together, thus increasing the wave's frequency and the pitch that is heard. The concept is the same for radio waves that bounce off a moving target, its frequency is shifted according to the speed and direction of the target. [19]

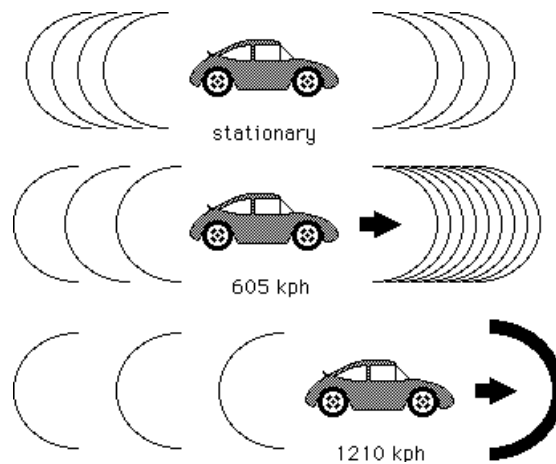


Figure 2.1: The Doppler Effect

2.2 FERS

FERS is the Flexible, Extensible Radar and Sonar Simulator. FERS aims to simulate the behavior of next generation radar and sonar systems. [4]

2.2.1 Background

FERS has been under active development as part of an ongoing project at the Radar Remote Sensing Group since March 2006, and is neither feature complete nor fully tested. [4]

“FERS was developed in the C++ and Python programming languages and was designed and written for maximum portability, and can be compiled and used without modification

on most modern operating systems (including Linux, Apple OSX and Microsoft Windows). The simulator uses open standards for data interchange, and does not require any commercial software or special purpose hardware. The complete source code for the simulator is available from the Radar Remote Sensing Group, or for download from the Sourceforge website.” [21]

2.2.2 Functionality

FERS works with an arbitrary number of receivers and transmitters (monostatic and multistatic), continuous wave and pulsed radar systems, and can simulate a radar using any pulse shape. The simulator currently simulates the amplitude, phase, Doppler and noise effects of a radar system. Simulation of other channel effects (such as multipath, fading, scattering, etc) is currently being considered.

The simulator allows loading of arbitrary pulse signals that can be generated using MATLAB or IDL. The simulation definition, or script, uses the extensible markup language (XML) to describe the radar system and environment to be simulated. Using the parameters (such as sample rate, carrier frequency, etc) defined in the XML script, FERS generates the waveform that the receivers will see. The results are written to XML, CSV and HDF5 for easy import into MATLAB, IDL, Excel etc.

[4]

2.2.3 FERS simulation definition

The simulation definition file is an XML file containing all the simulation parameters, the parameters of each of the objects in the simulated environment, and all other simulator settings. Where complete details of the parameters of an object (an antenna gain pattern, for example) cannot be stored in the definition, the definition references an external file. Understanding the format of this file is essential to the GUI design.

Figure 2.2 is an example of a typical definition file

```

<simulation name="sim1" >
  <parameters>
    <starttime>0</starttime>
    <endtime>5</endtime>
    <c>340.29</c>
    <rate>44100</rate>
    <export csv="true" binary="true" csvbinary="true" />
  </parameters>
  <pulse type="file" name="trpulse" filename="mono.csv" >
    <power>0.2</power>
    <carrier>40000</carrier>
  </pulse>
  <timing name="timing" >
    <frequency>10e6</frequency>
    <jitter>1e-9</jitter>
  </timing>
  <antenna pattern="isotropic" name="ant1" />
  <platform name="radarsite" >
    <motionpath interpolation="static" >
      <positionwaypoint>
        <x>0</x>
        <y>0</y>
        <altitude>0</altitude>
        <time>0</time>
      </positionwaypoint>
    </motionpath>
    <fixedrotation>
      <startazimuth>0</startazimuth>
      <startelevation>0</startelevation>
      <azimuthrate>0</azimuthrate>
      <elevationrate>0</elevationrate>
    </fixedrotation>
    <monostatic pulse="trpulse" type="pulsed" timing="timing" name="monol" antenna="ant1" >
      <window_skip>0</window_skip>
      <window_length>0.1</window_length>
      <prf>1</prf>
      <noise_temp>0</noise_temp>
    </monostatic>
  </platform>
  <platform name="aeroplane" >
    <motionpath interpolation="static" >
      <positionwaypoint>
        <x>5</x>
        <y>0</y>
        <altitude>0</altitude>
        <time>0</time>
      </positionwaypoint>
    </motionpath>
    <target name="tar1" >
      <rsc type="isotropic" >
        <value>23</value>
      </rsc>
    </target>
  </platform>
</simulation>

```

Figure 2.2: FERS definition script

The major sections of the simulation definition are[21]:

- parameters : a section describing simulation parameters such as start time, end time and propagation speed.
- pulse : one or more sections describing the waveforms to transmitted, or pointing to an external file containing waveform data.
- timing : one or more sections containing performance parameters for timing sources, such as phase noise and drift.
- antenna: one or more sections describing antenna gain patterns, or pointing to a file containing gain pattern data.

- platform: one or more sections describing platforms, and the receivers, transmitters or targets attached to them.

2.2.4 Results of the FERS simulator

The simulator produces two sets of output. The first set is saved in an XML file as well as a CSV file. It contains the results of the environment model — the basic parameters of radar performance such as the time the response was received, the return power, Doppler, and other similar results. Figure 2.3 is an example of a typical XML response file.

```

<receiver>
  <Response transmitter="trans1">
    <start>0.002938669958</start>
    <name>trpulse</name>
    <InterpolationPoint>
      <time>0.002938669958</time>
      <amplitude>0.0003027570949</amplitude>
      <phase>3.435635164</phase>
      <doppler>0</doppler>
      <power>9.1661858497e-08</power>
      <Iamplitude>-2.8976278578e-04</Iamplitude>
      <Qamplitude>-8.7746147928e-05</Qamplitude>
      <noise_temperature>1.000000000e+00</noise_temperature>
      <phasedeg>1.9684739487e+02</phasedeg>
    </InterpolationPoint>
    <InterpolationPoint>
      <time>0.002943669958</time>
      <amplitude>0.0003027570949</amplitude>
      <phase>3.435635164</phase>
      <doppler>0</doppler>
      <power>9.1661858497e-08</power>
      <Iamplitude>-2.8976278578e-04</Iamplitude>
      <Qamplitude>-8.7746147928e-05</Qamplitude>
      <noise_temperature>1.000000000e+00</noise_temperature>
      <phasedeg>1.9684739487e+02</phasedeg>
    </InterpolationPoint>
  </Response>
  <Response transmitter="trans1">
    <start>0.01244840571</start>
    <name>trpulse</name>
    <InterpolationPoint>
      <time>0.01244840571</time>
      <amplitude>9.158794559e-05</amplitude>
      <phase>-3128.625593</phase>
      <doppler>0</doppler>
      <power>8.3883517768e-09</power>
      <Iamplitude>8.4333457085e-05</Iamplitude>
      <Qamplitude>3.5724218578e-05</Qamplitude>
      <noise_temperature>1.000000000e+00</noise_temperature>
      <phasedeg>-1.7925704216e+05</phasedeg>
    </InterpolationPoint>
    <InterpolationPoint>
      <time>0.01245340571</time>
      <amplitude>9.158794559e-05</amplitude>
      <phase>-3128.625593</phase>
      <doppler>0</doppler>
      <power>8.3883517768e-09</power>
      <Iamplitude>8.4333457085e-05</Iamplitude>
      <Qamplitude>3.5724218578e-05</Qamplitude>
      <noise_temperature>1.000000000e+00</noise_temperature>
      <phasedeg>-1.7925704216e+05</phasedeg>
    </InterpolationPoint>
  </Response>
</receiver>

```

Figure 2.3: XML response

The signal renderer produces the second set of results stored in a binary file(HDF5 format) containing the samples of each receive window as captured by each receiver in the

simulated system. [21]

2.3 User requirements

In order to begin the design phase of the project, the required functionality of the GUI had to be determined.

After several information exchange sessions with persons experienced in the field of radar and the developer of the FERS simulator the following user requirements were established:

- Simulating Monostatic and Bistatic radar configurations
- Importing FERS definition files
- Ability to simulate several targets
- Visualization of the radar and targets
- Simple pulse compression – monochrome, chirp and step
- Importing waveforms from file
- Functionality similar to that of a previous FERS GUI
- Duplication of some of the functionality of Sarsim2
- Displaying results at a minimum
- Graphs of power(dB) vs range(m), Doppler(Hz) vs range(m), power(dB) vs time(s) and Doppler(Hz) vs time(s)

The sections that follow detail the research and software tools required to realize these requirements.

2.4 Existing GUI

In 2007 an interface to create an XML script for the FERS simulator was developed. The interface allowed the user to specify the system parameters, pulse properties, antenna properties, radar platform type and target platforms. Although the simulator was successful a number of improvements were needed. Investigating the functionality of this interface was a step needed in determining the functionality of the new FERS interface.

The interface to create the FERS XML script displays windows for each part of the simulation parameters needed. In the first window Figure 2.4 the initial system parameters are set. These included the simulation name, start and end time of the simulation, the sample

rate, propagation speed, interpolation rate, random seed, ADC bits, and the formats to export the results.

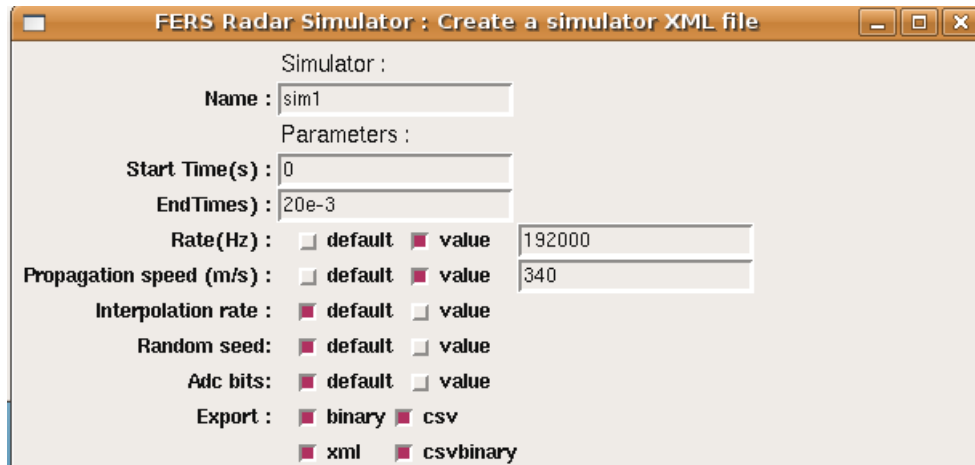


Figure 2.4: Interface for the simulator’s parameters

The next part of the input requires the user to specify the pulse being transmitted. The user has a choice of either specifying the pulse from a file or specifying the parameters for a rectangular pulse to be transmitted. Following this the jitter and frequency of timing sources used in the system are required. Next the antenna’s efficiency and pattern must be defined. Patterns to choose from are isotropic, sinc, and parabolic. Depending on the pattern chosen further parameters may be need to be inputted. The user is next required to choose the platform type, either monostatic or receiver/transmitter(bistatic) and the number of targets. For each platform the name and interpolation of which may be linear, static, cubic or Python interpolated are inputted, followed by the number of position waypoints and rotation waypoints which may also be interpolated. Next either zero or one fixed rotation can be specified.

If the monostatic platform is chosen it’s name, whether it’s pulsed or continuous, timing source, antenna type, pulse, window length, window skip and noise temp must be specified.

If the bistatic platform is chosen, the PRF, timing source, antenna type, window length, window skip, noise temp for the receiver must be specified and for the transmitter the name, whether it is pulsed or continuous, timing source, antenna type and pulse to be transmitted must be specified.

For the target platforms just the name and RCS isotropic value is needed.

The interface accesses a great amount of functionality that the FERS simulator provides. The new FERS interface is expected to be used for educational purposes and therefore some of the parameters that are not essential such as ADC bits and random seed can remain as the default values. The new FERS interface can also improve usability by having the input options displayed on a single window rather than displaying separate windows for the simulation parameters.

[6]

2.5 SARSIM2

SARSIM2[7] is a radar simulator with a graphical front-end that was developed by Rolf Legendfelder for his masters thesis in 1998. It has been around for many years and therefore has good documentation and therefore SARSIM2 was a good example of an existing radar simulator to be investigated for functionality as well identifying usability issues.

2.5.1 Features

Some of the features that SARSIM2 implements:

- Chirp, monochrome and user-defined pulse modulations
- Stepped-frequency implementation with constant or user-defined frequency increments
- Independent moving platforms with user-defined paths
- Generation of text scripts
- Configurable A/D conversion
- Angle dependent radar cross section (RCS) of point targets
- Rotating antennas
- Spot mode SAR Point target and platform motion errors
- All user-defined functions can be imported by a separate text file
- Powerful image viewer which can display SAR files of practically any size

2.5.2 SARSIM2 Input window

Figure 2.5 shows the main window of the SARSIM2 simulator.

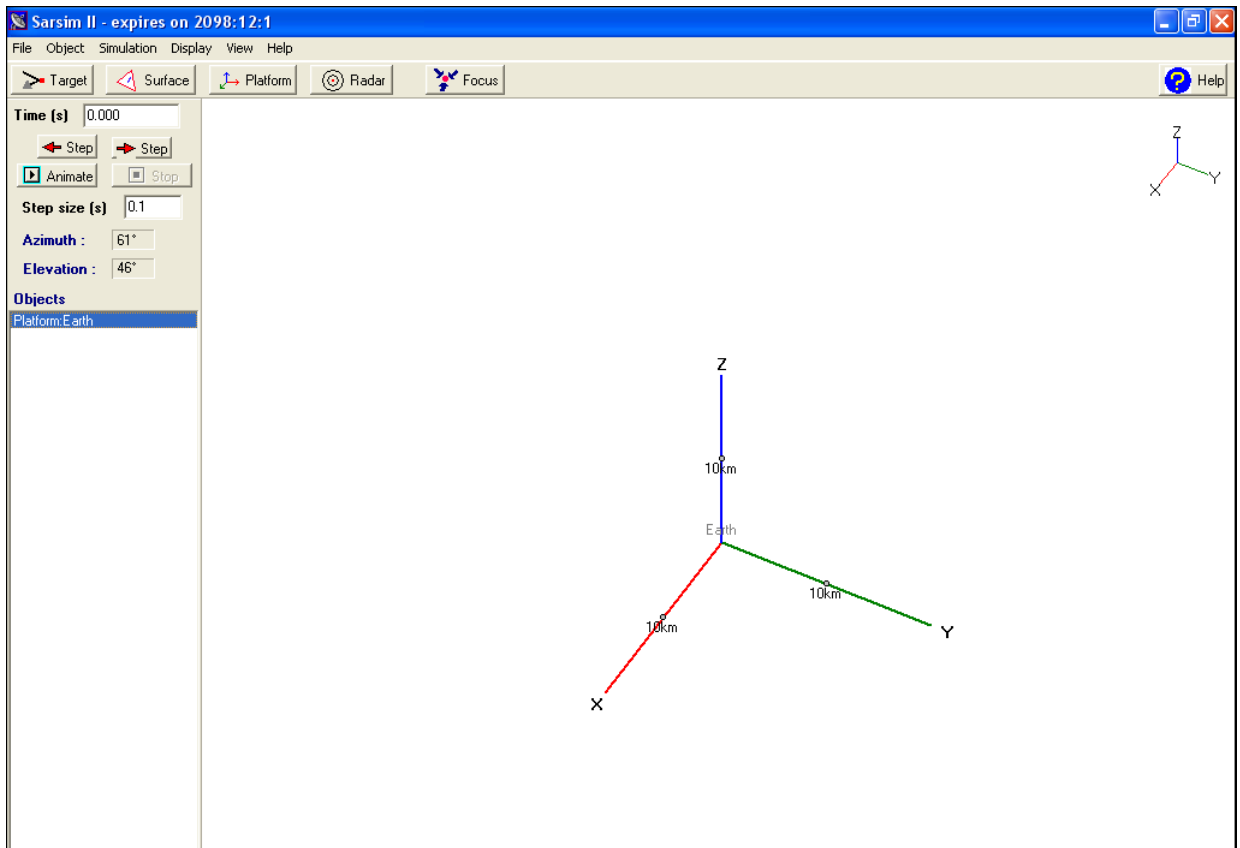


Figure 2.5: SARSIM2 main window

The user can add and edit platforms, surfaces, targets and radars. For each option points of interest were noted.

Platforms

The position and rotation of the platform must be specified. For every platform an additional 3D coordinate system is displayed.

Surfaces

The triangular position in 3D coordinates must be specified along with the Radar Cross Section parameters.

Targets

The platform that the target will be attached must first be specified, followed by the position, cross-section and reflectivity of the target.

Radars

Supports monostatic radars only. On the first page the platform that the radar is attached onto is specified along with the pulse attributes, frequency and transmission power. The

second page defines the antenna attributes such as gain and direction. Radars are displayed at the origin of their platform.

2.5.3 SARSIM2 3D coordinate system

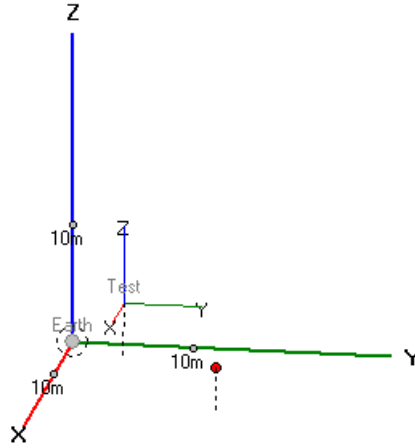


Figure 2.6: SARSIM2 3D coordinate system

The 3D coordinate system is extremely useful in helping the user to visualize the objects that are involved in the simulation. The plot can also be rotated and zoomed in and out using the mouse. An overview of the 3D coordinate system is also provided. Zooming does not increase the size of the axis, only the scale. Objects which are added are immediately reflected on the 3D coordinate system. The user is also given the option to choose the axis of rotation by changing the focus point.

2.6 Pulse compression

Pulse compression is an important function as it allows the user to see the effects the transmitted pulse has on the accuracy of the radar. In order to design the required compression as part of the GUI an understanding of how the pulse compressions are modeled is needed.

2.6.1 Monochrome

A monochrome pulse is modeled by

$$v_{TX}(t) = A \text{rect}\left(\frac{t}{T}\right) \cos(2\pi f_0 t) \quad (2.3)$$

[23]

where f_0 is the centre frequency, T the pulse length and $-\frac{T}{2} \leq t \leq \frac{T}{2}$.

2.6.2 Chirp

The chirp pulse is modeled by

$$v_{TX}(t) = A \text{rect}\left(\frac{t}{T}\right) \cos\left(2\pi\left[f_0 t + \frac{1}{2} K t^2\right]\right) \quad (2.4)$$

[23]

where f_0 is the centre frequency, T the pulse length, K is the chirp rate in Hz/s and $-\frac{T}{2} \leq t \leq \frac{T}{2}$.

2.6.3 Step frequency

A step frequency can be described as a series of pulses where the frequency of each pulse is incremented by a fixed frequency ∇f . The frequency of the n th pulse is then given by

$$f_n = f_0 + (n - 1)\nabla f \quad (2.5)$$

where f_0 is the nominal carrier frequency and ∇f is frequency step size. Each pulse has a fixed pulse width T and the pulses are transmitted at a fixed pulse repetition interval (PRI). [25]

2.7 Qt4.3 Designer

Qt by Trolltech [11] is the standard framework for developing high-performance cross-platform applications. “Qt provides single-source portability across Microsoft Windows, Mac OS X, Linux, all major commercial Unix variants, and embedded Linux. Qt provides application developers with all the functionality needed to build complex, high-performance GUI and console applications. Qt is fully object-oriented, easily extensible, and allows true component programming. Qt has formed the basis of many thousands of successful applications worldwide, including Google Earth, Adobe Photoshop Elements, and Skype. Qt is also the basis of the popular KDE Linux desktop environment, a standard component of all major Linux distributions.”[12] Qt Open Source Edition is provided to create free software that is licensed under the GNU GPL or a similarly recognized open source license. [13] Figure 2.7 shows the Qt designer window.

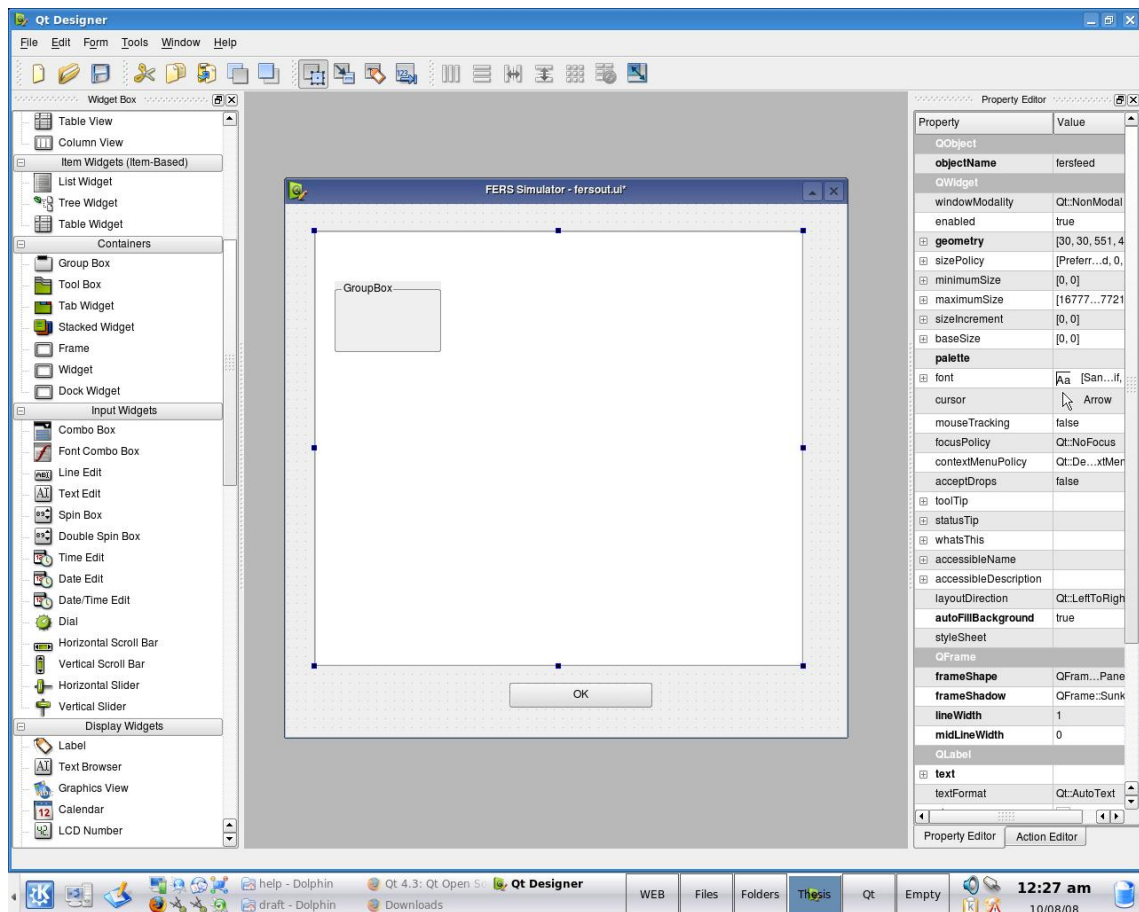


Figure 2.7: Qt4.3 Designer

Qt also provides the QtXml module which is a stream reader and writer for XML documents. It provides an interface to access and change the content and structure of an XML file. It creates a hierarchical view of the document (a tree view) and stores it in memory which makes manipulation easy.[17]

2.8 OpenGL

OpenGL was chosen to implement the 3D visualization. OpenGL is a standard API for rendering 3D graphics. OpenGL only deals with 3D rendering and provides little or no support for GUI programming issues. Fortunately the Qt OpenGL module makes it easy to use OpenGL in Qt applications. It provides an OpenGL widget class that can be used just like any other Qt widget.

2.8.1 Objects

Objects in OpenGL are specified by a set of vertices and the type of primitive. Primitives may be lines, triangles, rectangles or polygons. The primitive specified determines how the vertices will be joined. Take for example how to draw a line in OpenGL. If the primitive is specified as being a line then every pair of vertices specified will be joined by

a solid line. In the case of the triangle every 3 vertices will specify the points of a triangle.
[16]

2.8.2 Transformations

Scaling, orientation and positioning of objects in OpenGL are accomplished through modeling transformations. When transformations occur the current view matrix undergoes a series of matrix multiplications to change the scaling, orientation and positioning of the model. The order in which transformations are performed is therefore important since matrix multiplication is non-commutative.

Scaling

The `glScale` function multiplies the current view matrix by a matrix that stretches, shrinks or reflects an object along the axes.

Translation

The `glTranslated` command multiplies the current view matrix by a matrix that moves the object by the x,y and z coordinates specified.

Rotation

The `glRotated` command multiplies the current view matrix by a matrix that rotates the object by the angle specified about a vector starting at the origin in the direction specified by x,y and z.

[26]

2.8.3 Glut libraries

The Glut libraries are a series of pre-programmed drawing routines that can be used to create 3D models. This means that a large portion of code does not have to be reproduced every time by the programmer thus making coding in OpenGL easier. [15]

2.9 QWT

QWidgets for Technical applications(QWT) is a library created specifically for drawing graphs of scientific nature. Figure 2.8 is an example of a plot created using the QWT library. Since QWT is built around the existing Qt libraries, it was the perfect choice for plotting the results outputted by the FERS simulator.

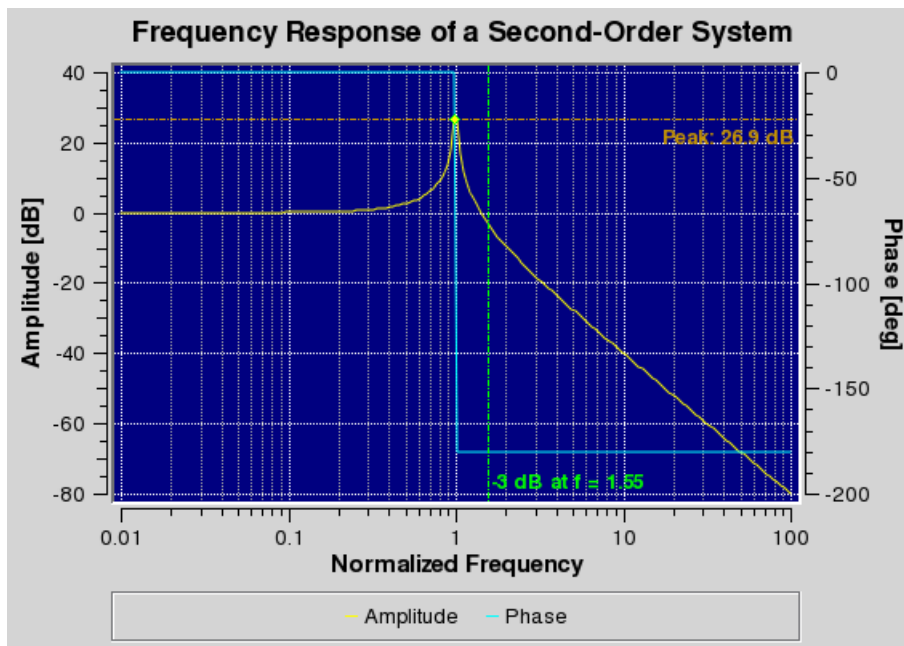


Figure 2.8: Example of a QWT plot
[16]

The basic procedure to plot a graph:

- Create a plot object to draw curves
- Create curves
- Get data for the curves
- Copy the data to the curves
- Set the style of the curves
- Attach the curves to the plot
- Show the plots

[18]

Chapter 3

Design

This chapter describes the design procedures followed in the development of the FERS GUI. The purpose of this phase is to determine the main components of the system so that they can be individually implemented. It begins by giving an overall description of the interaction between FERS, the GUI and the user. This is followed by determining the functionality of the GUI from the user requirements set out in section 2.3 of this report. Once these functional requirements are clarified the system components and their functions can be identified.

3.1 System overview

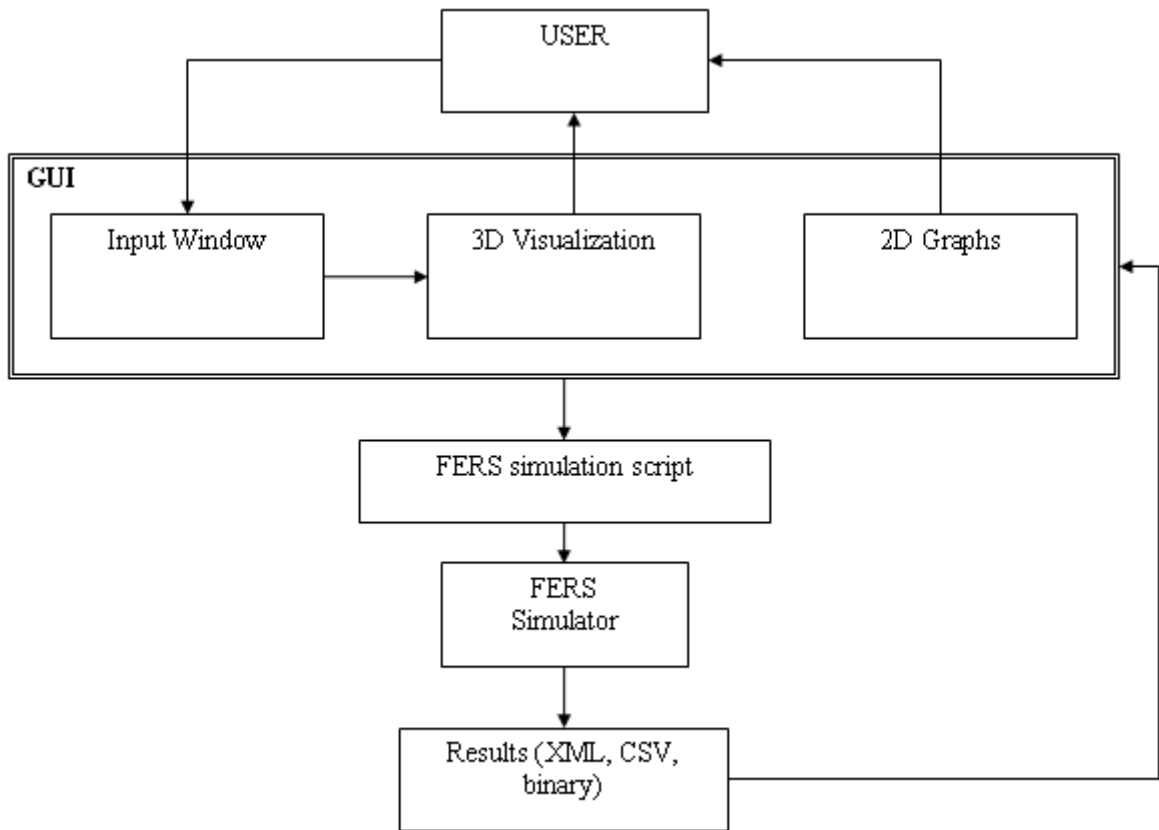


Figure 3.1: System overview

The overall system architecture in Figure 3.1 is highly abstracted, but this abstraction allows for easy understanding of how the GUI will function. The user enters the parameters of a scene to simulate and the transmitter, receiver and targets that the user specifies are displayed on a 3D plot. When the user chooses to simulate the scene the GUI creates an XML script of the parameters and invokes the FERS simulator with this file as its input. The FERS simulator executes this script and exports the responses of the receiver in 3 different file formats: XML, CSV and HDF5. The GUI then allows the user to view this output as 2D plots.

3.2 Functional requirements

The functionality that the GUI provides is determined from the user requirements established in Section 2.3 of this report.

Monostatic and Bistatic radar configurations

The user is given the option to choose between a monostatic or bistatic radar configuration. Depending on the choice of the user the appropriate input is enabled.

Ability to simulate several targets

The user must be given the choice of specifying more than one target. For each target the number of waypoints and RCS isotropic value must also be specified.

Visualization of the radar and targets

When the user specifies a radar or target these must be displayed on a 3D plot. These should be represented by different symbols. Users should be able to rotate and zoom into the plot.

Simple pulse compression

Monochrome, chirp and step pulse compression must be built into the GUI. Parameters for each pulse must also be inputted. Prior to simulation the chosen pulse compression will be written to file. This file is then referred to by the FERS input script.

Importing waveforms from file

The user should be given the choice of importing their own pulse file. This file is then referred to by the FERS input script.

Functionality similar to that of a previous FERS GUI

The flow of input will be similar to the previous FERS GUI. The user will enter the initial system parameters, the pulse to transmit, parameters for the radars and several targets. For each radar and target multiple waypoints can be inputted.

Duplicate some of the functionality of Sarsim2

The 3D visualization will be similar to that of SARSIM2. The user should be able to zoom and rotate the coordinate system. When objects are added to the simulation they should immediately be displayed.

Graphs

The graphs required will be drawn from the output files of the FERS simulator. The output files must therefore be read and the required information extracted. The user should be given the choice of which particular graph to display.

3.3 Additional features

Human Computer Interaction(HCI)

It was decided that the GUI would be more user-friendly if all possible inputs were displayed on a single window rather than the user being bombarded with new windows requiring input.

Waypoints

If multiple waypoints are specified they should be shown on the visualization. Lines should join them and each point should be labeled according to its waypoint number.

Working directory

The user must be given the option of choosing where to save the simulation data and results.

Visualization usability

During zooming extra grid-lines should be displayed.

Simulation messages

When the FERS simulator is completed a message should be displayed to notify the user if the simulation was successful or not.

3.4 System components

The system was broken down into the following components according to the required functionality. The role of each component is described below.

3.4.1 Input Window

This input window includes the graphical front-end as well as the C++ code that controls it. All user input is to be received by this component. Also the flow of input, depending on the choices made by the user, is directed by enabling or disabling the correct input options. The following must be implemented for this component to provide its function:

- Correct options are enabled when the platform or pulse type is changed
- Displaying the required number of waypoints or targets

- Options such as running the simulation or displaying graphs should only be enabled if all the system parameters are entered.

3.4.2 Pulse compression

This component will be responsible for creating the pulse that the user chooses from the input window. 3 types of pulse compression methods will be available: monochrome, chirp and step frequency. Therefore methods to create the required pulses must be implemented.

3.4.3 3D visualization

This component is responsible for creating the 3D display of the simulation using OpenGL. The following must be implemented:

- Displaying a 3D coordinate system
- Plotting of objects at coordinates specified
- Removing objects
- Zooming, rotating and panning of the coordinate system

3.4.4 Simulation

When the user has completed entering the required parameters and chooses to simulate, this component invokes the FERS simulator with the correct definition script. The methods outlined below must be available:

- calling the pulse compression component to create the correct pulse file
- creating the FERS definition script
- running the FERS simulator with the script created

3.4.5 Graphs

This component is responsible for displaying graphs from results produced by the FERS simulator. Options to choose from will be limited to power(dB) vs range(m), Doppler(Hz) vs range(m), power(dB) vs time(s) and Doppler(Hz) vs time(s). The following must be implemented

- reading the FERS output files to extract required data
- calling the QWT class to draw graphs with the extracted data

3.5 Component Diagram

The diagram below shows a modified collaboration diagram of the components which the GUI consists. The functions that each component will perform is included as well as the order in which the components can interact.

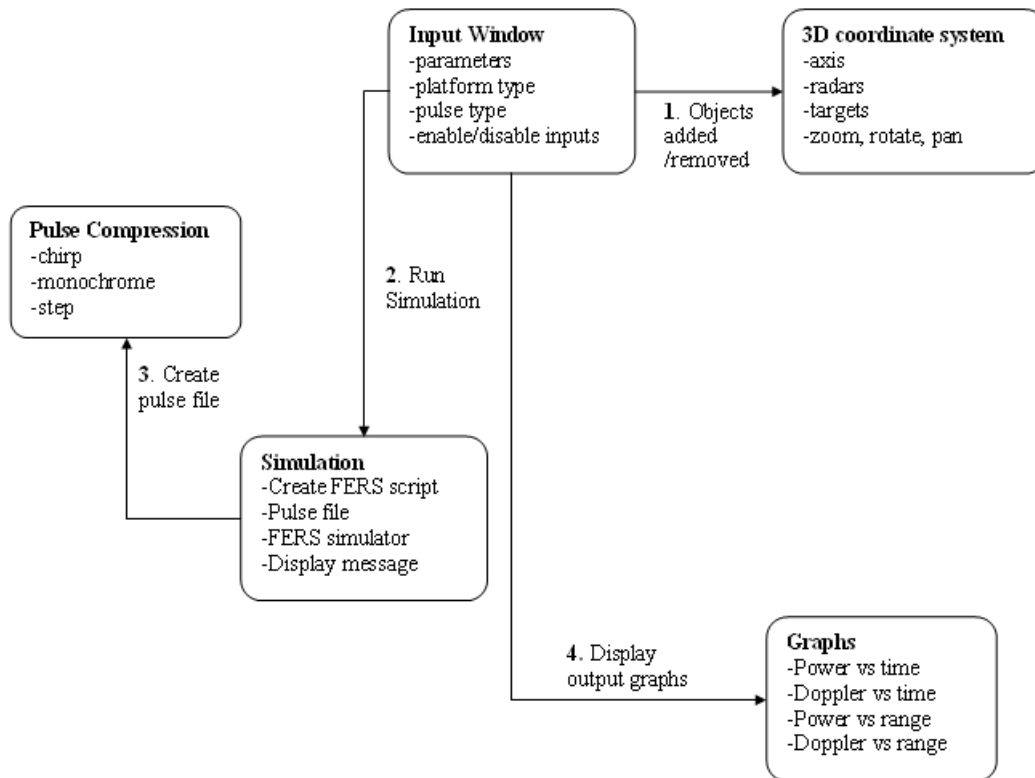


Figure 3.2: Component diagram

Chapter 4

Implementation

This chapter describes the procedures used to implement the required functionality. All implementation code can be found on the accompanying CD.

4.1 Input window

The input window is found on the left side of the main window. All inputs that the user can enter are displayed on this window. Obviously it is not possible to display all inputs options at once therefore tabbed windows were utilized to hide or display required inputs. The tabs are activated by combo boxes.

The input window is divided into 3 sections namely the system parameters, pulses and platforms.

Figure 4.1 is the completed input window.

If the PRF or propagation speed of the simulation is changed, the maximum non-ambiguous range which is displayed in the targets tab is updated.

When the OK button is pressed the radar or target and its waypoints are stored in a vector. This vector is then passed to the 3D visualization and displayed to the user.

4.2 Pulse compression

The pulse compressions incorporated were written in C++ and the resulting pulse written to a file. The pseudo code for each is described below. The Matlab code ported to C++ can be found in Appendix A.

4.2.1 Monochrome

```
variables : T,Fs //pulse width and sampling frequency  
time_step = 1/Fs;
```


Simulation name

Type

Parameters

Start time[s] Propagation speed[m/s]

End time[s] Sample rate[Hz]

Pulse

Choose pulse

Monochrome

Pulse width[s] Sampling freq[Hz]

Start freq[Hz]

Power[W] Carrier freq[Hz]

Timing Sources

Frequency[Hz] Jitter[Hz]

Monostatic **Targets**

Maximum Unambiguous Range($c/2PRF$) = 170.145m

Number of Targets

Target1

Name

RCS Isotropic Value

Waypoints

Number of Waypoints

Interpolation of waypoints

Waypoint1

X[m] Altitude[m]

Y[m] Time[s]

Figure 4.1: Input window

```

count to T in increments of time_step(t+=time_step)
    signal = cos(2*PI*(F1*t));2.3
    write signal to file;
end count

```

4.2.2 Chirp

```

variables:Fs,T,F1,F2 //sampling frequency, total time, start frequency, end frequency
chirp rate = (F2-F1)/T;
time_step = 1/Fs;
count to T in increments of time_step(t+=time_step)
    signal = cos(2*PI*((F1*t)+((alpha/2)*pow(t,2))));2.4
    write signal to file;
end count

```

4.2.3 Step frequency

```

variables : F1, Fn, Nf, Fs, T_freq //first frequency, last frequency, number of frequencies,
Sampling frequency,duration of each frequency
delta = (Fn-F1)/(Nf-1) //frequency step
create array of frequencies
calculate length of signal
loop through number of frequencies
    create signal for each frequency
    append signal to array
end loop
write signal array to file

```

4.3 3D visualization

The 3D visualization is displayed on the right-hand side of the main window. OpenGL was utilized for the implementation. The features that the visualization consists of are discussed below.

4.3.1 Coordinate system

The 3D coordinate system as shown in Figure 4.2 is drawn at the centre of the OpenGL display. The x,y and z axis extend 100 units each where each unit represents 1 km. Grids

are displayed on the xy, xz and yz planes. Major gridlines are drawn every 10 units. Depending on the zoom minor gridlines are drawn every 1 unit. At the edge of every major gridline the units are displayed.

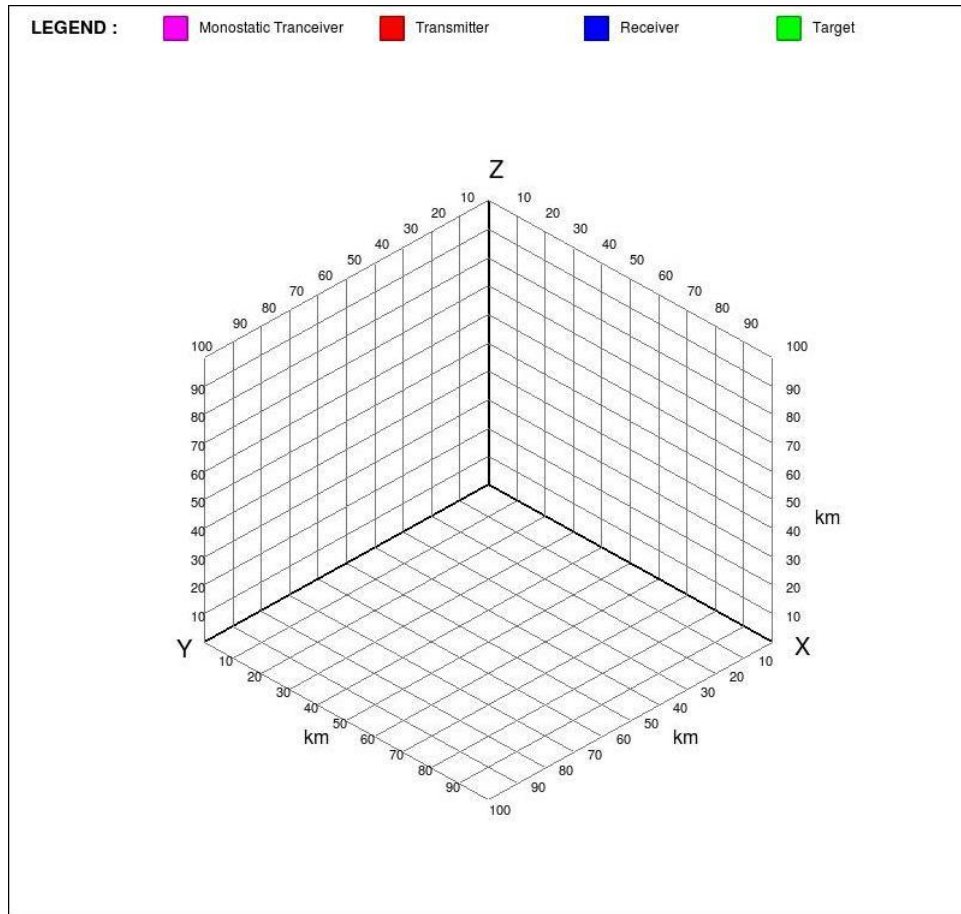


Figure 4.2: 3D coordinate system

4.3.2 Plotting objects

Two types of objects can be plotted. They are radars and targets. In the monostatic case a single transceiver is plotted and in the bistatic case a single transmitter and receiver is plotted. A maximum of six targets can be plotted. Radars are represented by a color coded antenna symbol as shown in Figure 4.3, magenta for the monostatic transceiver, red for the transmitter and blue for the receiver. Targets are represented by a green sphere.

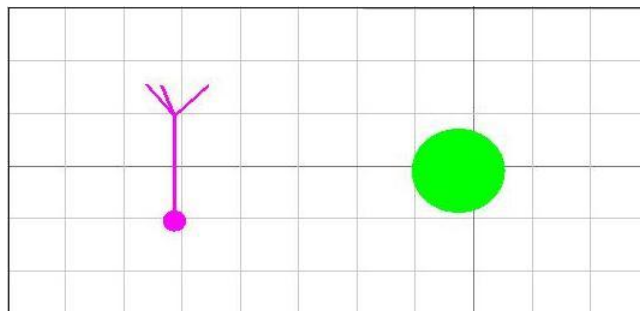


Figure 4.3: Object representations

Object Waypoints

Every object, together with its waypoints, that are added to the scene are stored in vector. When the objects are drawn on the OpenGL display, this vector is examined and the objects plotted at the position of the first waypoint. Any subsequent waypoints are represented by a sphere and labeled according to the waypoint number. The waypoints are joined by lines to indicate the linear interpolated path of the object. Figure 4.4 demonstrates how waypoints are plotted.

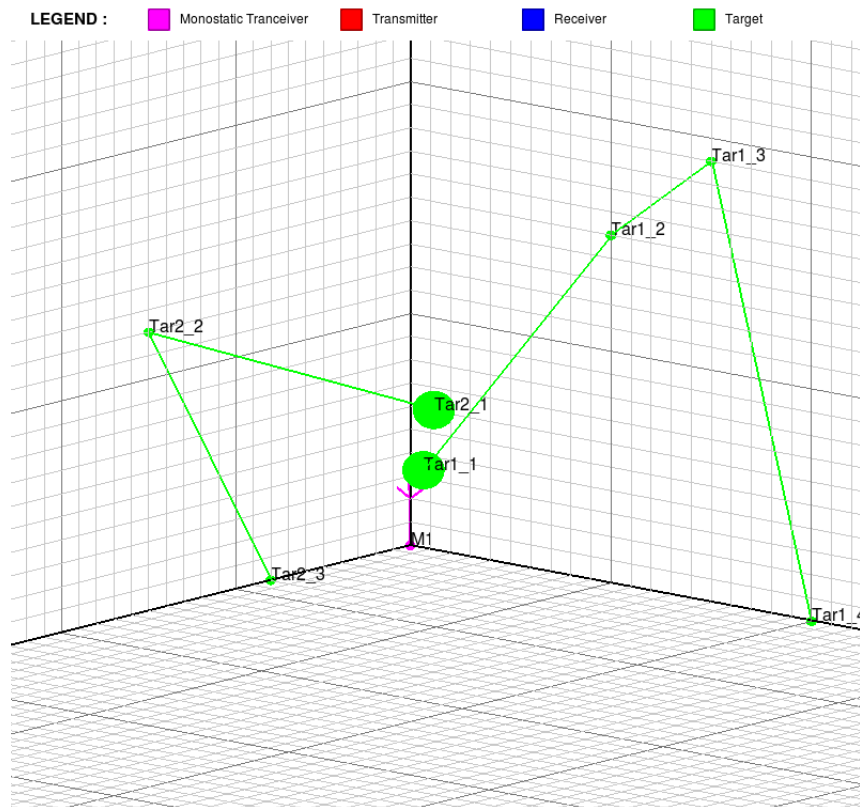


Figure 4.4: Objects' waypoints

4.3.3 Manipulating the visualization

The visualization can be zoomed, rotated and panned.

Zooming

Zooming occurs when the mouse scroll is activated. Zooming is achieved by storing the angle the mouse scroll is rotated. Scrolling up is in the positive direction and scrolling down is in the negative direction. This angle is then scaled and multiplied by the current view-port parameters.

Rotating

Rotating occurs when the left mouse button is clicked and the mouse is moved. The coordinate system is rotated in the direction that the mouse is moved. Rotating is achieved by scaling the distance the mouse was moved in a particular direction and then using the `glRotated` function to rotate the coordinate system accordingly.

Panning

Panning is similar to the procedure used to achieve rotation except that the right mouse button and the `glTranslated` function are utilized instead of the left mouse button and the `glRotated` function.

4.4 Running the Simulation

When the simulation is run the following occurs:

Pulse file is created if needed

If the user chooses any of the built-in pulse compression methods the pulses are stored in a file named according to the pulse chosen. No pulse files are created if the user chooses to browse for a pulse file.

Definition script is generated from inputs

The definition script created is similar to that in Figure 2.2. The `QXml` module is used to write the simulation parameters to a file named according to the simulation name.

FERS simulator invoked with definition script

- The `QProcess` module is used to run the FERS simulator with the definition script created.
- The FERS simulator writes the responses of the simulation to file.
- Any output messages from the FERS simulator are also stored to file.
- A message is displayed together with the output messages whether the simulation was successful or not.

Plot results enabled or disabled depending on success of simulation

- If the FERS simulator completes successfully the plot results options are enabled.
- If the FERS simulator is terminated due to any errors the plot result options are disabled.

4.5 Drawing Graphs of results

In order to create the required graphs the results of the FERS simulator is be read and the required values extracted. The XML results were chosen as the source of the results since experience in XML reading and writing is simplified by using the QXml module. The subsections below outline the pseudo code used to read the XML output and extract the needed values. Section 2.2 describes the format of the XML response file.

4.5.1 Power vs Time

Loop through responses

 Loop through interpolation points

 Store time

 Store power

 End loop

End loop

Call QWT plot with values of power and time

4.5.2 Doppler vs Time

Loop through responses

 Loop through interpolation points

 Store time

 Store Doppler

 End loop

End loop

Call QWT plot with values of Doppler and time

4.5.3 Power vs Range

Loop through responses

 Loop through interpolation points

```
    Get time
    Calculate range= ((receive_time mod PRF) - pulse_width)*propagation_speed
    Store range
    Store power
  End loop
End loop
Call QWT plot with values of power and range
```

4.5.4 Doppler vs Range

```
Loop through responses
  Loop through interpolation points
    Get time
    Calculate range= ((receive_time mod PRF) - pulse_width)*propagation_speed
    Store range
    Store Doppler
  End loop
End loop
Call QWT plot with values of Doppler and range
```

Chapter 5

Testing, Evaluation and Debugging

This chapter explains how the features of the GUI were tested for functionality as well as correctness. Any bugs that were discovered are noted. The chapter begins by detailing the tests carried out on the main input window to ensure correct display to the user. This is followed by evaluating the pulse compression built into the GUI. Next the script file generated for the FERS simulator is examined for correctness and this is followed by tests on the 3D visualization. Finally systems testing is carried out which tests all the components as a single application.

5.1 Main input window

Tests carried out on the main input window along with their results are outlined in Table 5.1.

5.2 Pulse compression

The pulse compression built into the GUI was tested by comparing results to that of equivalent Matlab code. The Tests were carried out as follows:

- The Matlab equivalent code was run and the results saved to file.
- The GUI was run with the same parameters as the Matlab equivalent code.
- The resulting files were compared.

5.3 FERS definition script

The definition files created by the GUI were tested for correct values and completeness. The scripts generated were run directly with the FERS simulator itself to ensure they were formatted correctly. The output of the simulator was examined and any errors noted. A

Aspect	Options	Expected Results	Pass /Fail	Bugs
Platform	Monostatic or bistatic	If monostatic is chosen the input to the transmitter and receiver of the bistatic platform is disabled and input to the monostatic platform is enabled. The opposite applies for the bistatic case.	Pass	
Pulse	Monochrome, chirp, step and browse for pulse	If a pulse is chosen only input related to that pulse is enabled and all other pulse inputs are disabled.	Fail	If the chirp pulse is selected after changing the selection from another pulse the monochrome input options are enabled as well.
Radar and target way-points	Increase or decrease the number of waypoints from 1 to 5	The number of waypoints allowed to be edited should be enabled and disabled accordingly	Pass	
Targets	Increase or decrease the number of targets from 1 to 6.	The number of targets allowed to be edited should be enabled and disabled accordingly.	Pass	
Browsers	Clicked	A window allowing the user to select a file or folder should appear.	Pass	
Simulation	Clicked	The FERS simulator will be run and a window displaying whether the simulation was successful or not is displayed	Pass	
Results	Clicked	Four graphs should be displayed	Pass	

Table 5.1: Testing and results for main input window

second test involved manually checking the generated scripts for the correct fields. The most effective method of accomplishing this was to generate a script from values inputted from an already existing script and compare the two. The existing script used can be found on FERS website [24].

Bugs

The noise temperature under the receiver platform was not included in the newly generated script.

5.4 3D Visualization

The plotting of objects is the most important aspect of the 3D visualization that required testing

5.4.1 Plotting

Objects added to the system had to be plotted at the coordinates specified. It was therefore necessary to ensure that this was the case. A problem is that objects in the 3D space cannot be verified that they are in the correct position by just visual inspection since the angle that the coordinate system is rotated by makes this very difficult. The positions of objects were therefore verified by examining its position in the xy, xz and yz planes. A single test case was sufficient for conclusions of the accuracy of object plotting.

Test case

3 objects, color coded red, green and blue were plotted at randomly chosen points : (30,45,60), (50,50,50) and (40,20,80) respectively.

Results

Figure 5.1 shows the output of the test case. Table 5.2 shows the results.

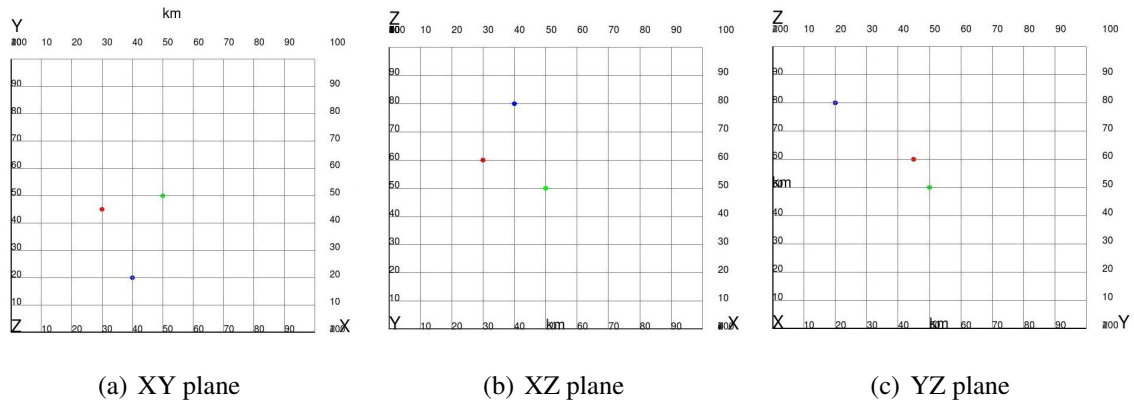


Figure 5.1: Plotting test results

Object	XY plane	XZ plane	YZ plane	XYZ coords
Red	(30,45)	(30,60)	(45,60)	(30,45,60)
Green	(50,50)	(50,50)	(50,50)	(50,50,50)
Blue	(40,20)	(40,80)	(20,80)	(40,20,80)

Table 5.2: Analysis of Plot results

As seen in Table 5.2 objects are plotted at the correct position and therefore the test was a success.

5.5 Systems Testing

Systems testing was essential to ensure that the individual components interacted in an desirable manner. The combinations of inputs are countless and therefore no single test case was sufficient. Systems testing is therefore an ongoing activity. Two major bugs that were found and fixed during the development of the GUI are noted below.

1. Removing targets out of order caused the GUI to crash. This occurred since targets were assumed to be added in order and therefore targets were removed according to their assumed index. For instance if target 3 was added first to the system and then removed, an attempt to remove the target at index 3 of the target vector resulted in a segmentation fault since target 3 was actually stored in index 1.
2. The range values of the output graphs were incorrect. It was found that the width of the pulse had to be subtracted from the delay time of the return signal before computation of the range.

Chapter 6

Command summary

This chapter provides a complete description of all available inputs and commands. This chapter is concluded with a walkthrough of a simple example.

6.1 Main Window

After starting the program a window similar to the one in figure 6.1 will be shown. On the left side of the window are the simulation inputs and on the right side of the window the 3D coordinate system can be found. Below the coordinate system are the current directory, simulation button and options to view results.

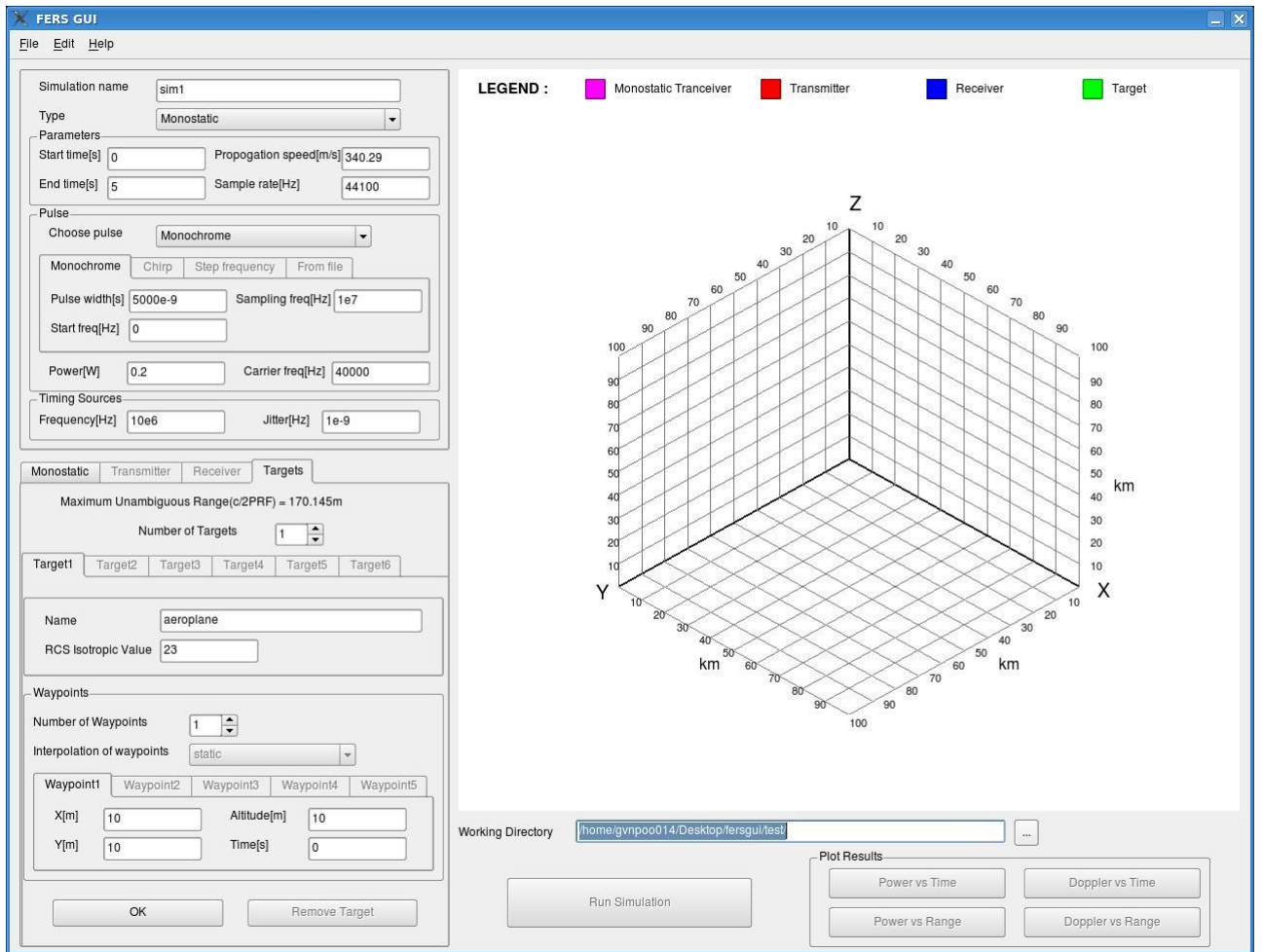


Figure 6.1: Main window

6.2 Simulation Inputs

The simulation inputs are divided into 3 parts: the simulation parameters, pulse type and platforms.

6.2.1 Parameters

These are the basic simulation parameters.

- simulation name
- type of platform : monostatic or bistatic
- start time in seconds
- end time in seconds
- propagation speed in metres per second
- sampling rate in Hertz

Figure 6.4 shows the parameter section of the simulation inputs.

6.2.2 Pulse Type

4 pulse inputs are available: monochrome, chirp step frequency and user defined. For each choice further parameters must be defined. The power and carrier frequency of the pulse chosen must also be defined. Figure 6.5 shows the inputs for a monochrome pulse.

Monochrome

- Pulse width
- Centre frequency
- Sampling frequency

Chirp

- First frequency
- Last frequency
- Total time
- Sampling frequency

Step frequency

- Start frequency
- End frequency
- Number of frequencies
- Duration of each frequency
- Sampling frequency

User defined

The user must include the path to a valid pulse definition file.

6.2.3 Platforms

The user may choose either from a monostatic or bistatic platform. If monostatic platform is chosen then the user must enter the parameters of the monostatic platform and 1 or more targets. If a bistatic platform is chosen then the user must enter the parameters for the receiver, transmitter and 0 or more targets. For each platform additional parameters are required.

Monostatic

Figure 6.6 shows the inputs for the monostatic platform

- Name
- PRF
- Number of waypoints
- Interpolation of waypoints
- The position(X, Y, altitude) and time of each waypoint
- Fixed rotation

Transmitter

Figure 6.2 a) shows the inputs for the transmitter

- Name
- PRF
- Window skip
- Window length
- Noise temperature
- Number of waypoints
- Interpolation of waypoints
- The position(X, Y, altitude) and time of each waypoint
- Fixed rotation

Receiver

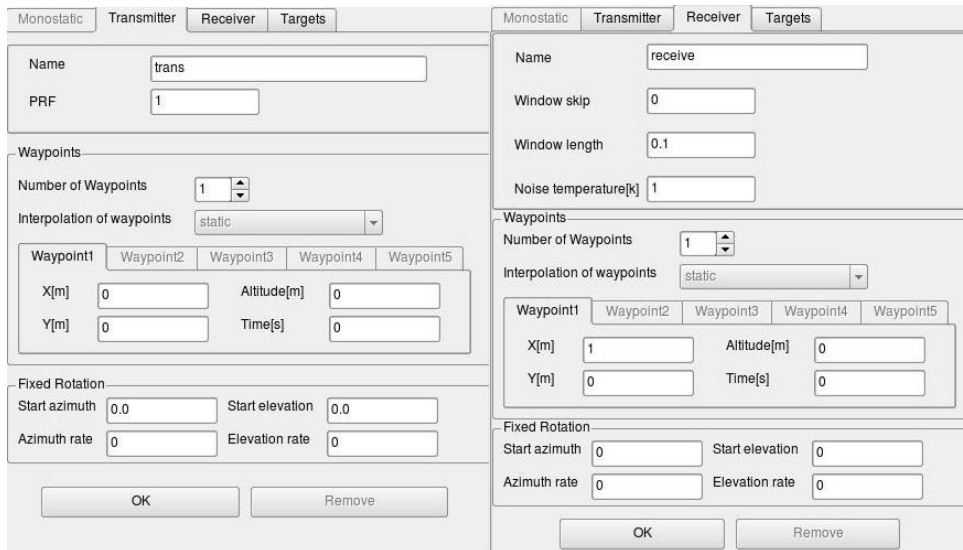
Figure 6.2 b) shows the inputs for the receiver

- Name
- Window skip
- Window length
- Noise temperature
- Number of waypoints
- Interpolation of waypoints
- The position(X, Y, altitude) and time of each waypoint
- Fixed rotation

Target

Figure 6.7 shows the target inputs

- Name
- RCS isotropic value
- Number of waypoints
- Interpolation of waypoints
- The position(X, Y, altitude) and time of each waypoint
- Fixed rotation



(a) Transmitter input

(b) Receiver input

Figure 6.2: Bistatic platforms

6.2.4 Working directory and results buttons

The working directory is the directory where the FERS definition script, pulse files and results of the simulation will be saved. The directory may be changed by choosing the browse option or by typing in a path to a folder. If the folder does not exist it will be created.



Figure 6.3: Working directory

6.3 A simple example

This simple step by step example models a monostatic radar, 2 moving targets and 1 stationary target.

6.3.1 Simulation setup

System parameters

The parameters as entered as shown in Figure 6.4. As seen the monostatic platform has been chosen. The total simulation time is 4 seconds with a propagation speed of $2.99 \cdot 10^3 m/s$ at a sampling rate of 44100 Hz.

Simulation name	simple_example		
Type	Monostatic		
Parameters			
Start time[s]	0	Propogation speed[m/s]	2.99e3
End time[s]	4	Sample rate[Hz]	44100

Figure 6.4: System parameters

Pulse

Figure 6.5 shows the parameters to be entered. The monochrome pulse has been chosen. The width of the pulse to be transmitted is 5ns at a sampling frequency of $10^7 Hz$ with a centre frequency of 0 Hz. The power has been set to 0.2 W at a carrier frequency of 40000 Hz.

Pulse	
Choose pulse	Monochrome
Monochrome	Chirp Step frequency From file
Pulse width[s]	5000e-9
Sampling freq[Hz]	1e7
Start freq[Hz]	0
Power[W]	0.2
Carrier freq[Hz]	40000

Figure 6.5: Pulse settings

Monostatic platform

Figure 6.6 shows the monostatic platform setup. The PRF is 10, window skip 0, window length 0.1 and noise temperature 0. The platform is stationary at the origin of the coordinate system with no rotation. Window skip is the time the receiver waits after the start of it's PRI to start sampling, and window length is the amount of time it samples for. The higher PRF means that more responses will be received within the simulation time frame and therefore it will be easier to distinguish the targets in the response graphs. Users must be aware that changing the PRF also changes the maximum non-ambiguous range.

The image shows a software interface for configuring a monostatic radar platform. It has four tabs: 'Monostatic', 'Transmitter', 'Receiver', and 'Targets'. The 'Targets' tab is selected. The interface contains several input fields and sections:

- Name:** radarsite
- PRF:** 10
- Window skip:** 0
- Window length:** 0.1
- Noise temperature[k]:** 0
- Waypoints:**
 - Number of Waypoints: 1
 - Interpolation of waypoints: static
 - Waypoint1: X[m]=0, Y[m]=0, Altitude[m]=0, Time[s]=0
 - Waypoint2: (empty)
 - Waypoint3: (empty)
 - Waypoint4: (empty)
 - Waypoint5: (empty)
- Fixed Rotation:**
 - Start azimuth: 0
 - Start elevation: 0
 - Azimuth rate: 0
 - Elevation rate: 0

At the bottom, there are 'OK' and 'Remove' buttons.

Figure 6.6: Monostatic platform settings

Targets

3 targets will be modeled. The setup for each is shown in Figure 6.7.

Table 6.1 shows the inputs for each target.

	Target 1	Target 2	Target 3
Name	moving_further	moving_nearer	stationary
RCS	3	6	9
No. of waypoints	5	5	1
Interpolation	Cubic	Cubic	Static
Waypoint 1(X,Y,Altitude,Time)	(80,0,0,0)	(9,0,0,0)	(50,0,0,0)
Waypoint 2(X,Y,Altitude,Time)	(60,0,0,1)	(17,0,0,1)	
Waypoint 3(X,Y,Altitude,Time)	(40,0,0,2)	(23,0,0,2)	
Waypoint 4(X,Y,Altitude,Time)	(20,0,0,3)	(31,0,0,3)	
Waypoint 5(X,Y,Altitude,Time)	(10,0,0,4)	(41,0,0,4)	

Table 6.1: Target setup

Maximum Unambiguous Range(c/2PRF) = 149.5m
Number of Targets: 3

Target1 Target2 Target3 Target4 Target5 Target6

Name: moving_further
RCS Isotropic Value: 3

Waypoints
Number of Waypoints: 5
Interpolation of waypoints: cubic

Waypoint1	Waypoint2	Waypoint3	Waypoint4	Waypoint5
X[m]: 10		Altitude[m]: 0		
Y[m]: 0		Time[s]: 4		

OK Remove Target

(a) Target 1 settings

(b) Target 2 settings

Maximum Unambiguous Range(c/2PRF) = 149.5m
Number of Targets: 3

Target1 Target2 Target3 Target4 Target5 Target6

Name: stationary
RCS Isotropic Value: 9

Waypoints
Number of Waypoints: 1
Interpolation of waypoints: static

Waypoint1	Waypoint2	Waypoint3	Waypoint4	Waypoint5
X[m]: 50		Altitude[m]: 0		
Y[m]: 0		Time[s]: 0		

OK Remove Target

(c) Target 3 settings

Figure 6.7: Target settings

6.3.2 3D visualization

Each radar or target added to the simulation appears on the 3D coordinate system. They are positioned according to their first waypoint and their linear interpolated path is also shown. Figure 6.8 illustrates this.

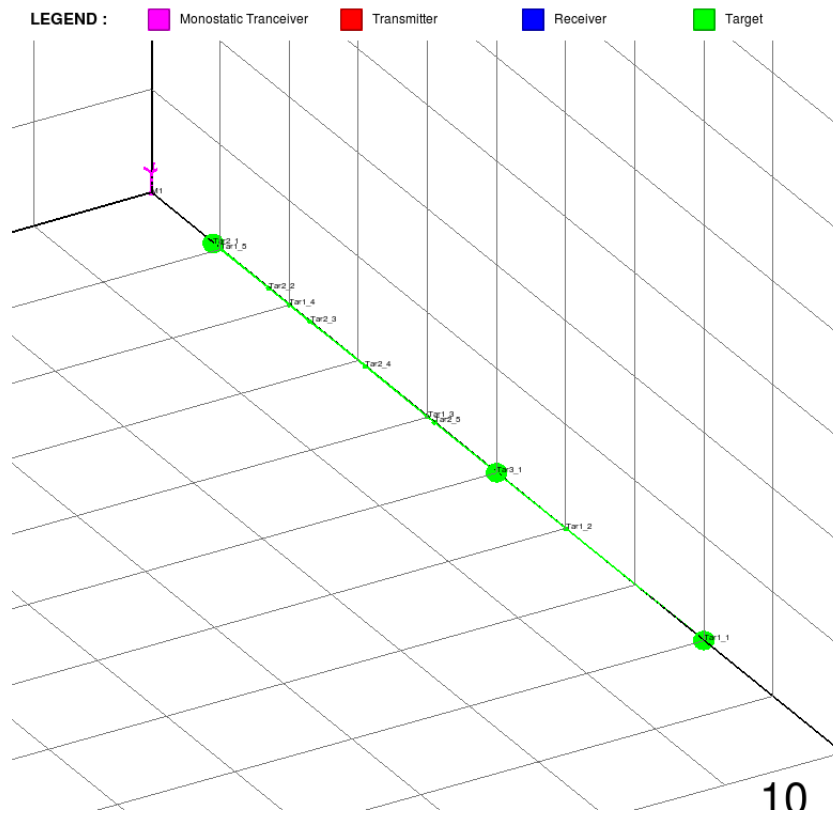


Figure 6.8: 3D visualization of system

6.3.3 Simulation

Once the above platforms have been added to the scene the simulation button will become enabled. Upon clicking this button the FERS simulator will be run. Upon completion a window as the one in Figure 6.9 will be displayed.

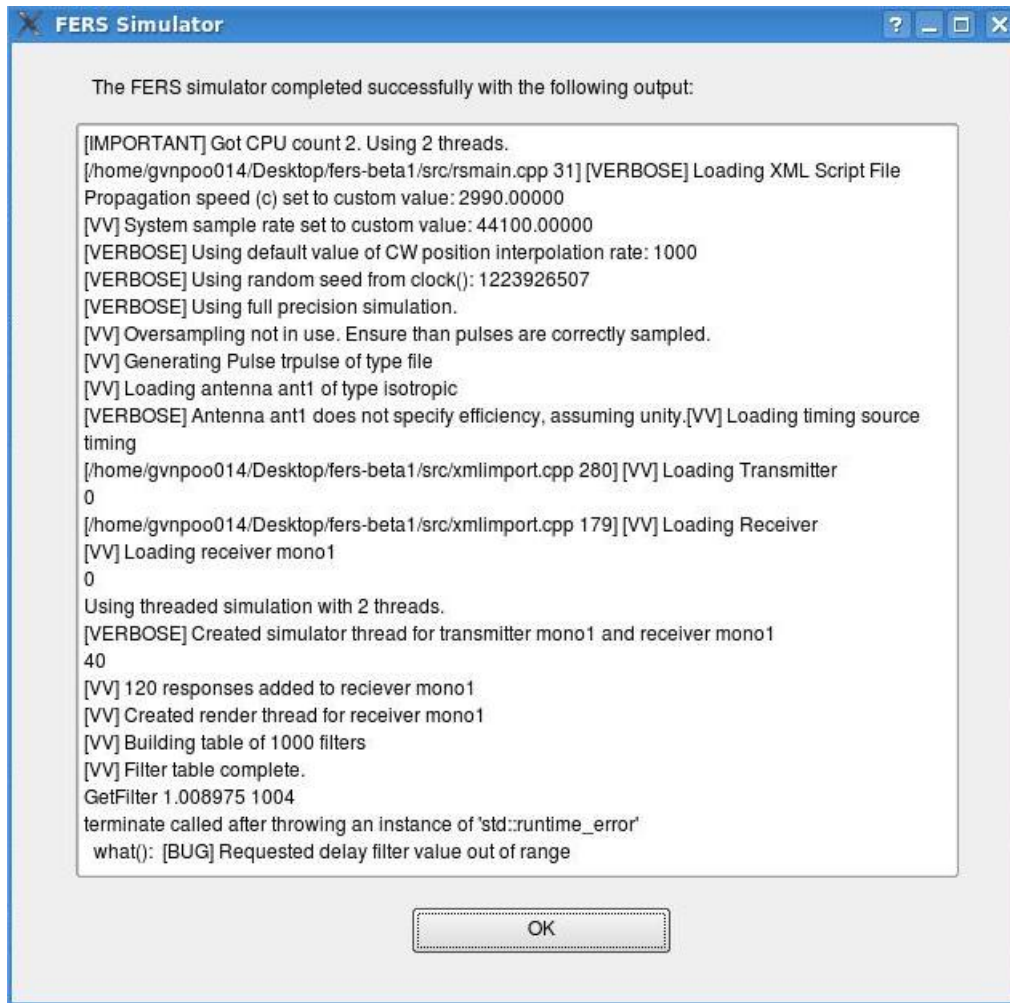


Figure 6.9: Simulation complete window

6.3.4 Results

Upon completion of a successful simulation the plotting options will be enabled. The user can choose from graphs of Power vs Time, Doppler vs Time, Power vs Range and Doppler vs Range. Together these graphs can tell the user the number of targets present together with characteristics of its motion and reflectivity.

Power vs Time

The results in Figure 6.10 shows that 3 targets are present. The Power vs Time graph can tell the user the direction in relation to the receiver that the target is moving. In this example one target is moving away from the radar since its power is decreasing, one target is moving towards the radar since its power is increasing and one stationary target is present since its power is constant.

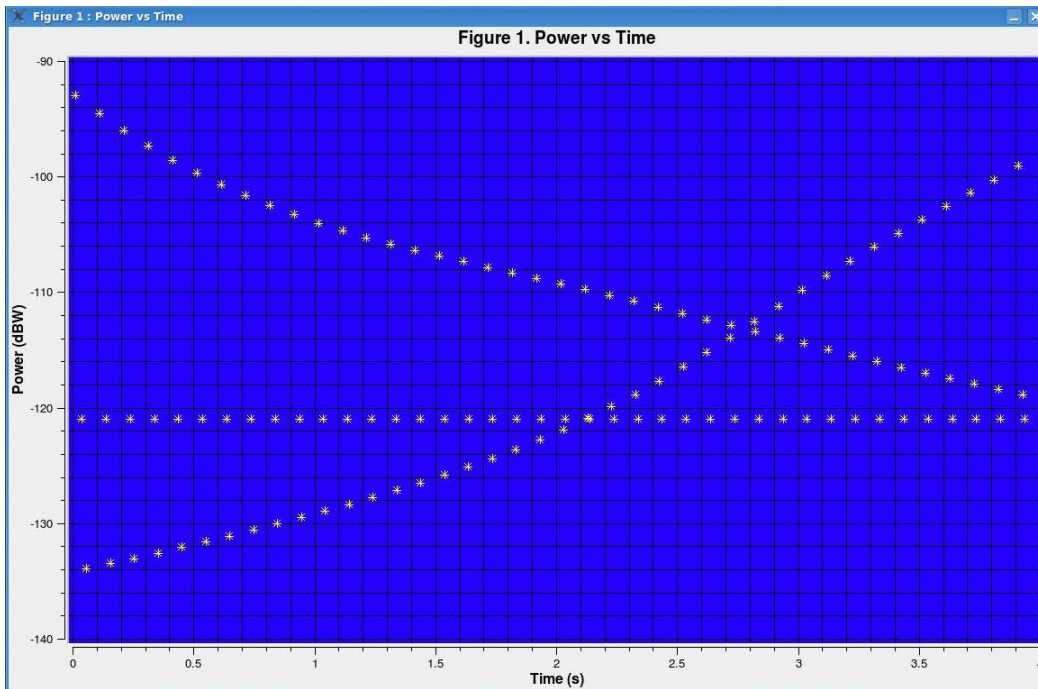


Figure 6.10: Graph of Power vs Time

Doppler vs Time

The Doppler vs Time graph gives an indication about the velocity of the targets. A constant Doppler means the target is moving at constant velocity. In this example two targets with varying velocity and one stationary target can be identified.

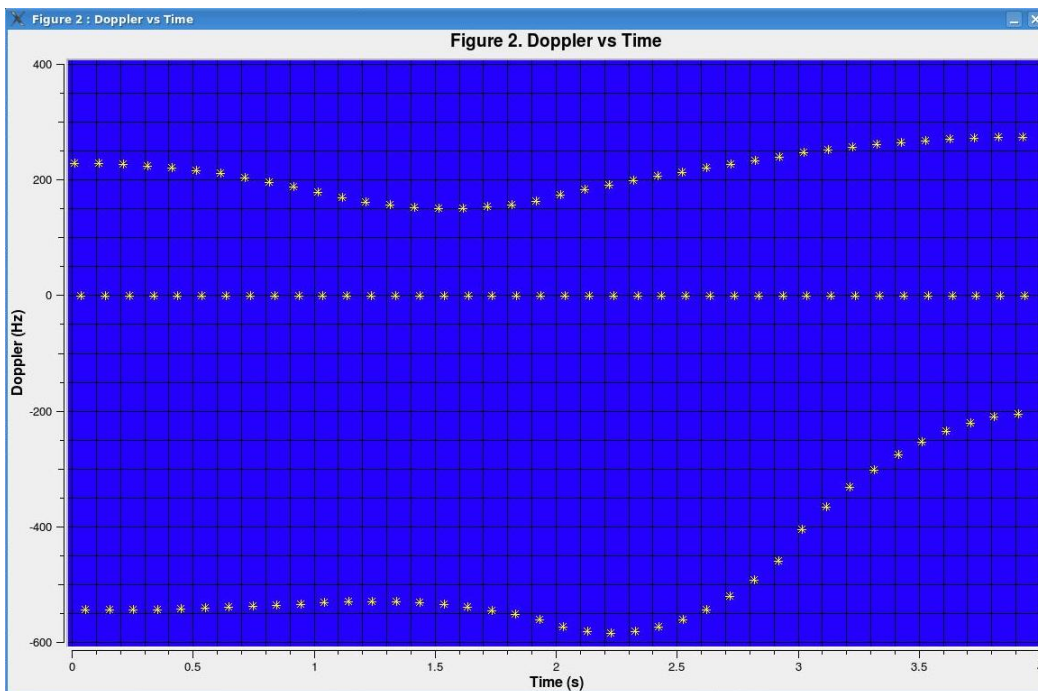


Figure 6.11: Doppler vs Time

Power vs Range

From the Power vs Range graph the viewer is shown the range through which the targets move as well as their reflectivity. Note that the direction is not given. In this example one target moves between 9km and 40km, a second target moves 10km and 80km, and a third target is stationary at 50km. It is also seen that some targets reflect more energy than others at the same range.

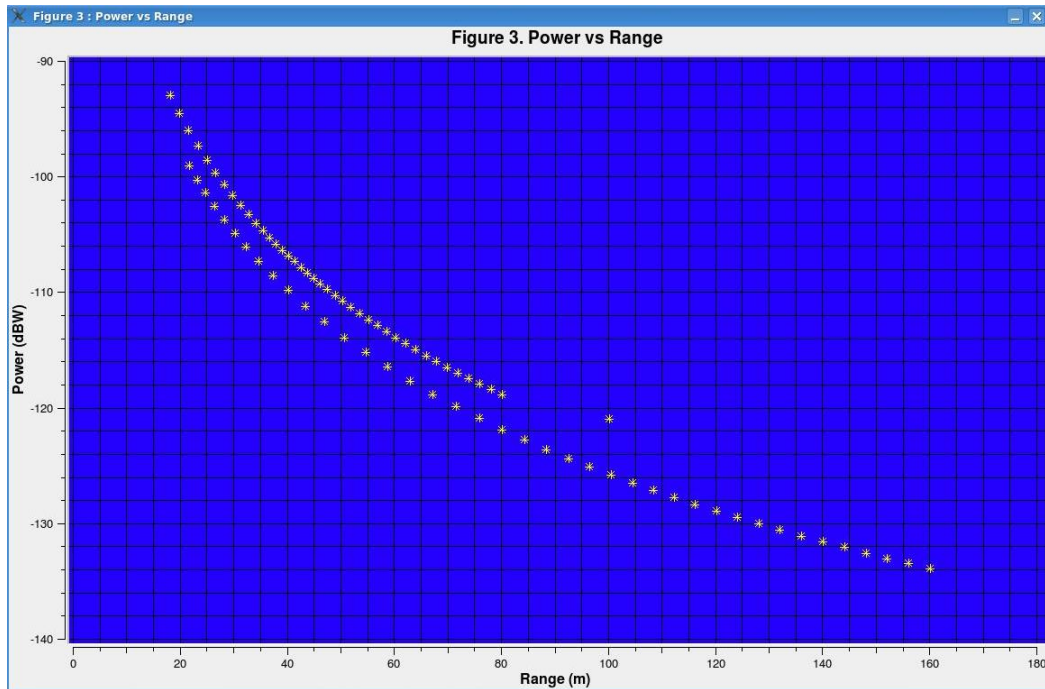


Figure 6.12: Power vs Range

Doppler vs Range

Figure 6.13 shows the Doppler of the targets over the range they travel. The results are similar to the Doppler vs Time graphs except now we know the Doppler of the targets at an exact range.

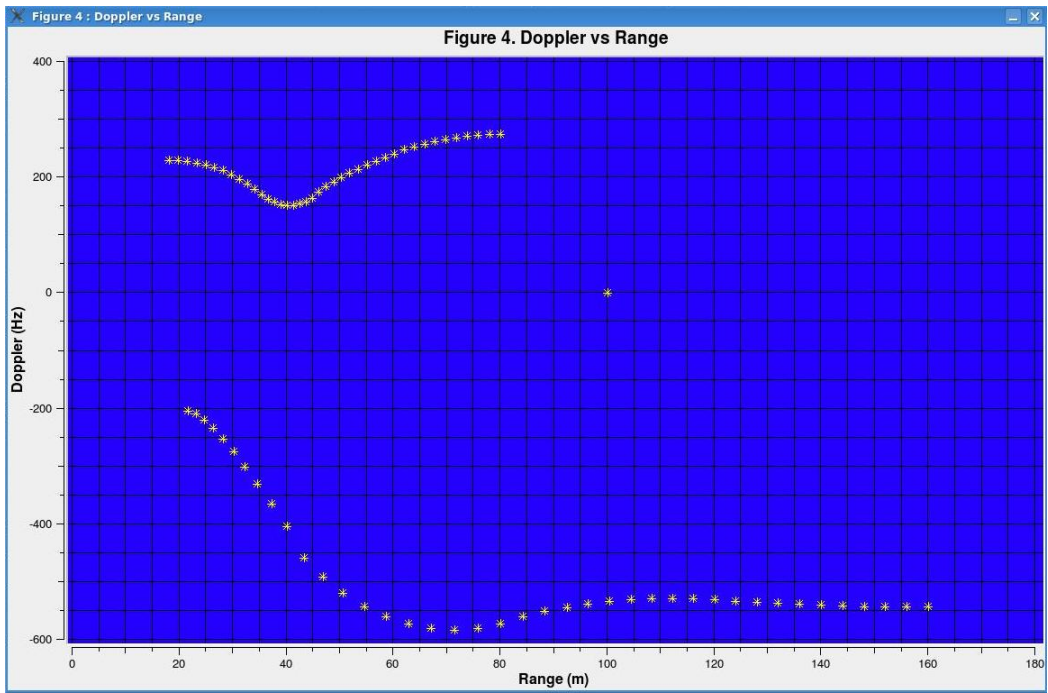


Figure 6.13: Doppler vs Range

Chapter 7

Conclusions

This chapter identifies the main objectives set out at the beginning of this project and compares them to the functionality of the completed GUI.

Objectives

The objectives of this project were to design and implement a Qt based GUI for the FERS simulator to be used for educational purposes.

The functionality that the GUI was required to provide:

- Simulating Monostatic and Bistatic radar configurations
- Importing FERS definition files
- Ability to simulate several targets
- Visualization of the radar and targets
- Simple pulse compression – monochrome, chirp and step frequency
- Importing waveforms from file
- Functionality similar to that of a previous FERS GUI
- Duplication of some of the functionality of Sarsim2
- Displaying results at a minimum
- Graphs of power(dB) vs range(km), Doppler(Hz) vs range(km), power(dB) vs time(s) and Doppler(Hz) vs time(s)

GUI capabilities

The GUI developed has the following capabilities:

- Modeling pulsed monostatic and bistatic radars with a maximum of 6 targets.
- Up to 5 waypoints can be specified for each object.
- The transmitting antenna and targets are considered isotropic.
- Monochrome, chirp and step frequency pulse compressions are built into the GUI.
- Waveforms can also be imported from file.
- A 3D visualization of the scene is displayed to the user. Each object along with its motion path is plotted on the display.
- The user can choose to display graphs of power vs range, Doppler vs range, power vs time and Doppler vs time.

Assessment of the GUI

The GUI has met the requirements of the project in most aspects. The only function that was not implemented due to time constraints is the ability to import FERS definition files. There are numerous inputs which can be defined in the FERS definition script. Many of these were either kept fixed or removed in order to make the GUI more usable. One of the main objectives of this project is to create a GUI for educational purposes and decreasing the complexity of the GUI was essential to achieve this.

In comparison to Sarsim2 this GUI is more user-friendly and less complex in terms of possible input parameters. Although the 3D visualization is not animated and real-time, the waypoints of each object are displayed allowing the user to visualize the entire scene to be simulated.

This GUI is simple and intuitive to students with a basic background of radar systems therefore making it ideal for students who are new to the subject of radars.

Chapter 8

Future work and recommendations

The GUI can be improved in several ways. Listed below are some of the improvements which may be implemented in the future as well as features that were not implemented due to time constraints.

- Improved error checking and handling
- Support of multistatic radar configurations
- Support of continuous waveforms
- 3D visualization animated in real-time when the simulation is run.
- Importing of FERS definition files
- Support of sinc and parabolic radar antennas
- Importing pulses from HDF5 files
- Adapting the GUI for advanced users

Bibliography

- [1] P. Z. Peebles. Radar Principles. Wiley-Interscience. 1998.
- [2] M. Skolnik. Introduction to Radar Systems. McGraw-Hill, NY, 2nd ed, 1980.
- [3] Bookrags staff. "Radar and Sonar". 2005. October 19 2008. [online]. Available : <http://www.bookrags.com/research/radar-and-sonar-woes-02/> .
- [4] FERS radar simulator [online]. Available : <http://www.brooker.co.za/fers/> .
- [5] Flexible Extensible Radar Simulator[online] Available : <http://sourceforge.net/projects/fers>.
- [6] B Fabien. Interface to create a script for a flexible, extensible radar and sonar simulator. Radar Remote Sensing Group, University of Cape town. July 2007.
- [7] R. Legenfelder. The Design and Implementation of a Radar Simulator. Master's thesis, University of Cape Town, September 1998.
- [8] W. Norman. A Technical Introduction to XML. World Wide Web Journal. October 03, 1998. [online]. Available : www.xml.com/pub/a/98/10/guide0.html.
- [9] The HDF Group. What is HDF5?. February 22nd 2008. [online] Available : www.hdfgroup.org/HDF5/whatishdf5.html.
- [10] FILExt - The File Extension Source. File Extension .CSV Details. [online]. Available : filext.com/file-extension/CSV .
- [11] Qt for Open Source C++ development on Linux/X11. [online]. Available : <http://trolltech.com/downloads/opensource/appdev/linux-x11-cpp> .
- [12] About Qt. Trolltech. 2008. [online]. Available : <http://doc.trolltech.com/4.3/aboutqt.html> .
- [13] Qt Open Source Edition. Qt. Trolltech. 2008. [online]. Available : <http://doc.trolltech.com/4.3/opensourceedition.html> .
- [14] OpenGL Overview. OpenGL.org organization. 2008. [online]. Available : <http://www.opengl.org/about/overview/> .

- [15] GLUT - The OpenGL Utility Toolkit. OpenGL.org organization. 2008. [online]. Available : www.opengl.org/resources/libraries/glut/ .
- [16] U Rathmann, J Wilgen. Qwt - Qt Widgets for Technical Applications. 2008. [online] Available : <http://qwt.sourceforge.net/index.html> .
- [17] QtXml Module. Trolltech. 2008. [online]. Available : <http://doc.trolltech.com/4.3/qtxml.html> .
- [18] U Rathmann, J Wilgen. QwtPlot Class Reference. 2008. [online] Available : http://qwt.sourceforge.net/class_qwt_plot.html .
- [19] R Croci. Radar Basics. October 2002. [online]. Available : http://www.alphaalpha.org/radar/intro_e.html .
- [20] About Radar. University of Washington Radar Remote Sensing Laboratory. [online]. Available : <http://rrsl.ee.washington.edu/radar.htm> .
- [21] M. Brooker. The Design and Implementation of a Simulator for Multistatic Radar Systems. Master's thesis, University of Cape Town, June 2008.
- [22] Introduction to HDF5. hdf5group.org. 2008. [online]. Available : <http://www.hdfgroup.org/HDF5/doc/H5.intro.html> .
- [23] A. J. Wilkinson. EEE4001F. Radar and Sonar course notes, Chapter 7, page 13, 2008.
- [24] Example FERS Radar Simulation. [online]. Available : <http://www.brooker.co.za/fers/example2.html>.
- [25] G. S. Gill and J. Huang. The Ambiguity Function of the Step Frequency Radar Signal Processor, pages 375-380. Naval Postgraduate School Monterey, California 93943, USA.
- [26] S. K. Semwal. OpenGL tutorial. [online]. Available : http://sabia.tic.udc.es/gc/practicas/opengl_tut_externos/semwal%20tutorial/geometric.html#Transform .
- [27] P. Wolfenden. The Physics of Waves and Oscillation. [online]. Available : http://people.finearts.uvic.ca/~aschloss/course_mat/MU207/wolfenden_article.html .

Appendix A

Matlab source code for pulse compressions

–Provided by Yoann Paichard

A.1 Chirp

```
1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %Variable definition
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 Fs = 1000; %Sampling frequency
5 T = 2; %Total time
6 F1 = 0; %Start frequency
7 F2 = 300; %End frequency
8 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
9 %Signal construction
10 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
11 %time
12 t = 0:0.001:2; % 2 secs @ 1kHz sample rate
13 alpha = (F2-F1)/T;
14 sgn = real(exp(2*j*pi*(F1*t+(alpha/2)*t.^2)));
15 plot(sgn)
16 %figure
17 plot(t,sgn)
```

A.2 Step frequency

```
1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %Variable definition
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 % F1 first frequency
5 F1= 1e6;
6 % Fn last frequency
7 Fn = 2e6;
8 %Number of frequency
9 Nf = 101;
10 % Sampling frequency
11 Fs=1e7;
12 %Duration of each frequency
13 T_freq=50e-6;
14 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
15 %Construction of the signal
16 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
17 % Frequency step
18 Delta_freq= (Fn-F1)/(Nf-1);
19 %Array of Frequencies
20 Fp = [F1:Delta_freq:Fn];
21 % Length signal (nb samples)
22 L_sgn=round(Nf*Fs*(T_freq));
23 %time
24 t=(1:L_sgn)/Fs;
25 %signal initialisation
26     jj = 1;
27 S = [];
28 % signal Computaion
29 for ii=1:Nf %100
30     t_int = t(jj:jj+Fs*T_freq-1); %time in the loop 500 values
31     Y= cos(2*pi*Fp(ii)*t_int); %signal for one frequency
32     jj=jj+floor(Fs*T_freq);
33     S = [S Y]; % horizontal signal concatenation with the new Y end;
34 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
35 %Graph
36 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
37 figure(1);
38 plot(t,S)
```


Appendix B

Software source code

The source code for the FERS GUI can be found on a CD at the back of the project report submitted. The source code consists of 3 classes, 2 user interfaces created with Qt4.3 Designer and a project file to build the program. The program is dependant on Qt4.3 Designer with OpenGL support and QWT5 in order to build successfully. The header files for the 3 classes are provided below.

B.1 Source header files

B.1.1 Fersgui

This class contains the C++ code to control the input window and implement the pulse compressions and output graphs.

```
#ifndef FERSGUI_H
#define FERSGUI_H
#include "ui_fersgui.h"
#include "glwidget.h"
class Fersgui : public QMainWindow, private Ui::fersgui
{ Q_OBJECT
public:
Fersgui(QMainWindow *parent = 0);
//OpenGL window containing 3D coordinate system
GLWidget *glWidg;
//Create Monochrome pulse with parameters specified and write to file
void monoFile(double F1, double Fs, double Pl);
//Create Chirp pulse with parameters specified and write to file
void chirpFile(double Fs, double T, double F1, double F2);
//Create Step frequency pulse with parameters specified and write to file
```

```

void stepFile(double F1, double Fn, double Nf, double Fs, double T_freq);
public slots:
void platype(); //Display the platforms that the user chooses
void writePulse(); //Call correct method to write the pulse that user has chosen
void writeXML(); //Create the FERS defination script from inputs
void tar1waysc(); //Display correct number of waypoints for target 1
void tar2waysc(); //Display correct number of waypoints for target 2
void tar3waysc(); //Display correct number of waypoints for target 3
void tar4waysc(); //Display correct number of waypoints for target 4
void tar5waysc(); //Display correct number of waypoints for target 5
void tar6waysc(); //Display correct number of waypoints for target 6
void pulfile(); //Display pulse inputs that user chooses to transmit
void pfileBrowse(); //Display a file browser so user can choose pulse
void mwaysc(); //Display correct number of waypoints for monostatic platform
void trawaysc(); //Display correct number of waypoints for bistatic transmitter
void rwaysc(); //Display correct number of waypoints for bistatic receiver
void tarAdd(); //Enable correct number of target tabs
void mAdd(); //Add monostatic platform to visualization window
void traAdd(); //Add bistatic transmitter to visualization window
void rAdd(); //Add bistatic receiver to visualization window
void aRange(); //claculate and display maximum non-ambiquous range
void tar1OK(); //Add target 1 to visualization window
void tar2OK(); //Add target 2 to visualization window
void tar3OK(); //Add target 3 to visualization window
void tar4OK(); //Add target 4 to visualization window
void tar5OK(); //Add target 5 to visualization window
void tar6OK(); //Add target 6 to visualization window
void mRemove(); //Remove monostatic platform from visualization window
void traRemove(); //Remove bistatic transmitter from visualization window
void rRemove(); //Remove bistatic receiver from visualization window
void tar1Remove(); //Remove target 1 from visualization window
void tar2Remove(); //Remove target 2 from visualization window
void tar3Remove(); //Remove target 3 from visualization window
void tar4Remove(); //Remove target 4 from visualization window
void tar5Remove(); //Remove target 5 from visualization window
void tar6Remove(); //Remove target 6 from visualization window

```

```

void outfolderBrowse(); //Display browser so user can choose working directory
void simulate(); //create the necessary files and invoke the FERS simulator
void powtimepl(); //Plot the power vs time plot using QWT
void dopplertimepl(); //Plot the Doppler vs time plot using QWT
void powrangepl(); //Plot the power vs range plot using QWT
void dopplerrangepl(); //Plot the Doppler vs time plot using QWT
};
#endif

```

B.1.2 Glwidget

This class creates and controls aspects of the visualization window.

```

#ifndef GLWIDGET_H #define GLWIDGET_H
#include <QGLWidget>
//#include "fersgui.h"
class GLWidget : public QGLWidget
{ Q_OBJECT
public:
GLWidget(QWidget *parent = 0);
~GLWidget();
//coordinates of waypoints
struct objectinfo
{
double x;
double y;
double z;
};
//vector of target waypoints
QVector< QVector< QVector<double > > > targetsv;
//number of waypoints for each target
int tar1ways;
int tar2ways;
int tar3ways;
int tar4ways;

```

```

int tar5ways;
int tar6ways;
//vector of target waypoints
QVector<QVector<double > > monov;
int mways;//number of waypoints for monostatic radar
//vector of target waypoints
QVector<QVector<double > > trav;
int traways;//number of waypoints for bistatic transmitter
//vector of target waypoints
QVector<QVector<double > > recv;
int rways;//number of waypoints for bistatic receiver
QSize minimumSizeHint() const;
QSize sizeHint() const;
//add targets to 3D display
//getTargets(target number,number of waypoints, waypoints*6)
void getTargets(int tnum, double waypoints, QVector<double> way1,QVector<double>
way2,QVector<double> way3,QVector<double> way4,QVector<double> way5);
void removeTarget(int tnum);
//add monostatic radar to 3D display
//getMono(number of waypoints, waypoints*6)
void getMono(double waypoints, QVector<double> way1,QVector<double> way2,QVector<double>
way3,QVector<double> way4,QVector<double> way5);
void removeMono();
//add bistatic transmitter to 3D display
//getTra(number of waypoints, waypoints*6)
void getTra(double waypoints, QVector<double> way1,QVector<double> way2,QVector<double>
way3,QVector<double> way4,QVector<double> way5);
void removeTra();
//add bistatic receiver to 3D display
//getTra(number of waypoints, waypoints*6)
void getRec(double waypoints, QVector<double> way1,QVector<double> way2,QVector<double>
way3,QVector<double> way4,QVector<double> way5);
void removeRec();
//Draw radars and targets and their waypoints
GLuint drawObjects();
//Draw Axis and grids

```

```

void drawMarkings();
void initializeGL();
public slots:
void setXRotation(int angle);
void setYRotation(int angle);
void setZRotation(int angle);
signals: //custom signals
void xRotationChanged(int angle);
void yRotationChanged(int angle);
void zRotationChanged(int angle);
protected:
void paintGL();
void resizeGL(int width, int height);
void mousePressEvent(QMouseEvent *event);
void mouseMoveEvent(QMouseEvent *event);
void wheelEvent(QWheelEvent *event);
private:
GLuint drawAxis(double zoomed);
void normalizeAngle(int *angle);
GLuint axis;
GLuint objects;
int xRot;
int yRot;
int zRot;
QPoint lastPos;
double centerX;
double centerY;
double pixmapScale;
double curScale;
};
#endif

```

B.1.3 Fersout

This class reads the display messages from the FERS simulator and displays them in a window when the simulation is completed.

```

#ifndef FERSOUT_H
#define FERSOUT_H
#include "ui_fersout.h"
class FersOut : public QDialog, private Ui::fersout
{ Q_OBJECT
public:
//window to display output of FERS simulator
FersOut(QWidget *parent = 0);
public slots:
//Write output of FERS simulator to window and return whether simulation was success-
ful or not bool displayOutput(QString file);
//exit window
void exit(); };
#endif

```

B.2 Build notes

When qmake is run on the project file, the header file, "ui_fersgui.h", that is created for the fersgui interface needs the following to be edited:

1. "#include <Qt3Support/Q3MainWindow>" changed to "#include <QMainWindow>"
2. Any appearance of "Q3MainWindow" changed to "QMainwindow"