Radiolocation Using AM Broadcast Signals

by

Timothy Douglas Hall

B.S. Electrical Engineering, University of Missouri, 1993
M.S. Electrical Engineering, University of Missouri, 1994

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

Massachusetts Institute of Technology

September 2002

© Massachusetts Institute of Technology
All Rights Reserved

Signature of Author

Department of Electrical Engineering and Computer Science
26 August 2002

Certified by

Charles C. Counselman III
Professor of Planetary Science
Thesis supervisor

Accepted by

Arthur C. Smith
Chairman, Department Committee on Graduate Student
Radiolocation Using AM Broadcast Signals

by

Timothy Douglas Hall

Abstract

I have designed, built, and evaluated a passive radiolocation system that uses only signals of opportunity, that is, signals that exist for purposes other than radiolocation. The system estimates the relative position vector between a base station, which is a navigation receiver at a known location, and a rover, which is like the base station but free to move about. The relative position vector, called the baseline vector, is determined by multilateration from observations of the carrier phases of signals received from AM broadcast stations. This system determines the horizontal components of the baseline with about ten-meter uncertainties for baseline lengths up to about 35 kilometers.

The navigation receivers are implemented as software radios on standard Intel®-based personal computers. The signals received by a one-meter vertical whip antenna are bandpass-filtered, amplified, and digitized. The entire AM band is digitized so simultaneous observation of all available signals is achieved. All further processing of the signals, including carrier-phase determination, is implemented in software run on the personal computer. The base station and rover record observed phase, frequency, and amplitude data on their local hard drives; and navigation algorithms are implemented in post-real-time.

The interpretation of a carrier-phase observation in terms of position is ambiguous because one cycle of a carrier wave is virtually indistinguishable from the next. Previous attempts at signal-of-opportunity navigation using carrier phase sidestepped the ambiguity problem by requiring that the initial position of the rover be known and that phase variations be tracked without interruption. I designed and implemented an ambiguity-function method that enables the phase ambiguity to be resolved instantaneously without position initialization or signal-tracking continuity.

I encountered several impediments to AM-broadcast-based radiolocation that, if not dealt with appropriately, reduce positioning accuracy, reduce ambiguity-resolution robustness, or both. AM transmitter position uncertainty directly causes receiver position-determination uncertainty. Since the error in published antenna positions sometimes exceeds 100 meters, I conducted sub-meter-accuracy geodetic surveys of 29 Boston-area
AM-broadcast antennas. The directional radiation patterns of the array antennas of many AM broadcast radio stations have phases that vary with azimuth angle. I developed and implemented a model for the phase of a directional antenna that nearly eliminated the errors caused by this effect. AM broadcast signals travel primarily as groundwaves, which propagate with phase velocities that depend on the electrical properties of the ground. Using simulations and empirical data, I designed and implemented a model for groundwave propagation that greatly reduced the errors caused by this effect over a broad geographic area. Proximate overhead and underground conductors, especially ones that are part of vast interconnected networks, can perturb phase locally by a radian or more, and in some cases can cause ambiguity-resolution failure. At night when the D-layer of the ionosphere recombines, signals in the AM band reflect off the ionosphere, which enables so-called skywave propagation. Since skywave can lead to interference with distant stations, regulations require many radio stations to significantly reduce power at night. Therefore, signals from far fewer AM radio stations are useful for nighttime navigation. Among signals that are still useful at night, skywave signals interfere with the desired groundwave signals and cause positioning performance accuracy to degrade by more than an order of magnitude.

AM radiolocation positioning performance varies greatly with the local environment of the navigation receivers. Outdoors in the open, 95% of positioning errors are smaller than 15 meters for baselines up to 35 kilometers long. In wooded areas, where GPS positioning performance drops significantly, AM positioning performance is not affected. However, significant challenges remain to make AM positioning useful near tall buildings in urban areas, or inside structures.

Thesis Supervisor: Charles C. Counselman III
Title: Professor of Planetary Science
Acknowledgements

I thank Prof. Charles C. Counselman III, my thesis advisor, for the countless hours he spent helping me with this work. Working with Chuck was not only educational but also enjoyable. I could not possibly have had a better advisor.

I also thank the other members of my thesis committee: Prof. John Tsitsiklis and Prof. John Kassakian. Both took time out of their vacations to read my thesis for which I am very grateful.

I am grateful to many people at MIT Lincoln Laboratory for their support. Specifically, I thank Dr. Pratap Misra for introducing me to the field of radiolocation and for being an outstanding mentor. Pratap’s unwavering encouragement while I was working on my thesis is greatly appreciated. I also thank Dr. Jay Sklar for providing me with a research assistantship after other funding at the lab dried up. Many thanks also go to Brian Adams for providing me with lab and office space and for helping me on countless occasions with experiment logistics.

I also thank the National Science Foundation for supporting my research through a Graduate Research Fellowship for the first three years of my program at MIT.

Finally, I thank my family and friends for their love and support. I especially thank my parents, Ron and Gayle Hall, for their unquestioning love and encouragement. I believe my success in life is due in no small part to their selfless commitment to family. My family is my anchor. Last, but certainly not least, I thank Kristin Little for her love and friendship.
# Table of Contents

Abstract.......................................................................................................................... 3  
Acknowledgements ......................................................................................................... 5  
Table of Contents ........................................................................................................... 7  
List of Figures .................................................................................................................. 11  
List of Tables .................................................................................................................... 17  

## Chapter 1: Introduction

1.1 Navigation Using GPS ................................................................................................. 19  
1.1.1 Navigation Using Differential GPS ...................................................................... 20  
1.1.2 Navigation Using Observations of GPS Carrier Phases ...................................... 20  
1.2 Navigation Using Signals of Opportunity ................................................................. 21  
1.3 Motivation .................................................................................................................. 21  

## Chapter 2: Navigation Receiver

2.1 Frequency Band ......................................................................................................... 23  
2.2 AM Navigation System Considerations .................................................................... 23  
2.3 AM Navigation Receiver Hardware .......................................................................... 24  
2.3.1 Antenna ................................................................................................................ 24  
2.3.2 Pre-Amp ............................................................................................................... 25  
2.3.3 Low Pass Filter ................................................................................................. 26  
2.3.4 Amplifier ............................................................................................................. 27  
2.3.5 Power Supply, Gain Control, and Peak Detection ............................................. 28  
2.3.6 Calibrator .......................................................................................................... 29  
2.3.7 A/D Converter .................................................................................................... 30  
2.3.8 Clock Circuity ..................................................................................................... 31  
2.3.9 Computer ........................................................................................................... 31  
2.3.10 Construction ..................................................................................................... 32
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4 AM Navigation Receiver Software</td>
<td>44</td>
</tr>
<tr>
<td>2.4.1 Data Collection Parameters</td>
<td>45</td>
</tr>
<tr>
<td>2.4.2 Data Processing Definitions</td>
<td>45</td>
</tr>
<tr>
<td>2.4.3 Data Processing Steps</td>
<td>45</td>
</tr>
<tr>
<td>2.4.3.1 Window Function</td>
<td>46</td>
</tr>
<tr>
<td>2.4.3.2 Fast Fourier Transform</td>
<td>46</td>
</tr>
<tr>
<td>2.4.3.3 Carrier Search</td>
<td>46</td>
</tr>
<tr>
<td>2.4.3.4 Carrier Frequency Refinement</td>
<td>47</td>
</tr>
<tr>
<td>2.4.3.5 Carrier Phase and Amplitude</td>
<td>48</td>
</tr>
<tr>
<td>Chapter 3: AM Navigation Algorithms</td>
<td>49</td>
</tr>
<tr>
<td>3.1 Incremental Navigation</td>
<td>49</td>
</tr>
<tr>
<td>3.1.1 Algorithm Description</td>
<td>49</td>
</tr>
<tr>
<td>3.1.2 Experimental Validation</td>
<td>50</td>
</tr>
<tr>
<td>3.2 Instantaneous Navigation</td>
<td>51</td>
</tr>
<tr>
<td>3.2.1 Algorithm Description</td>
<td>52</td>
</tr>
<tr>
<td>3.2.2 Experimental Validation</td>
<td>58</td>
</tr>
<tr>
<td>3.2.3 Experimental Validation for a Longer Baseline</td>
<td>61</td>
</tr>
<tr>
<td>3.3 Concluding Remarks</td>
<td>64</td>
</tr>
<tr>
<td>Chapter 4: AM Navigation Challenges</td>
<td>65</td>
</tr>
<tr>
<td>4.1 Transmitter Antenna Position</td>
<td>65</td>
</tr>
<tr>
<td>4.1.1 Effect on Navigation</td>
<td>65</td>
</tr>
<tr>
<td>4.1.2 Minimizing Effect on Navigation</td>
<td>66</td>
</tr>
<tr>
<td>4.1.3 FCC Position Coordinate Errors</td>
<td>67</td>
</tr>
<tr>
<td>4.1.4 Alternative Solutions</td>
<td>68</td>
</tr>
<tr>
<td>4.2 Transmitter Antenna Pattern</td>
<td>69</td>
</tr>
<tr>
<td>4.2.1 WRKO Simulation</td>
<td>69</td>
</tr>
<tr>
<td>4.2.2 Antenna Pattern Model</td>
<td>72</td>
</tr>
<tr>
<td>4.2.3 Model Sensitivity</td>
<td>73</td>
</tr>
<tr>
<td>4.2.4 Experimental Results</td>
<td>74</td>
</tr>
<tr>
<td>4.3 Imperfect Ground</td>
<td>78</td>
</tr>
<tr>
<td>4.3.1 Baseline NEC-4 Simulations</td>
<td>78</td>
</tr>
<tr>
<td>4.3.2 Baseline Experimental Results</td>
<td>81</td>
</tr>
<tr>
<td>4.3.3 Conductivity Estimation Using Amplitude</td>
<td>84</td>
</tr>
<tr>
<td>4.3.4 Permittivity Estimation</td>
<td>89</td>
</tr>
<tr>
<td>4.3.5 Ground Model</td>
<td>91</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.4 Overhead Conductors</td>
<td>94</td>
</tr>
<tr>
<td>4.4.1 Isolated Overhead Steel Guy Wire</td>
<td>96</td>
</tr>
<tr>
<td>4.4.2 Multiple Overhead Conductors</td>
<td>99</td>
</tr>
<tr>
<td>4.5 Underground Conductors</td>
<td>106</td>
</tr>
<tr>
<td>4.5.1 Underground Water Pipe</td>
<td>106</td>
</tr>
<tr>
<td>4.5.2 Briggs Field Underground Conductors</td>
<td>108</td>
</tr>
<tr>
<td>4.6 Skywave</td>
<td>113</td>
</tr>
<tr>
<td>4.7 Concluding Remarks</td>
<td>120</td>
</tr>
<tr>
<td>Chapter 5: Navigation Performance</td>
<td>123</td>
</tr>
<tr>
<td>5.1 Zero-Baseline Experiment</td>
<td>123</td>
</tr>
<tr>
<td>5.2 Outdoor Experiments</td>
<td>125</td>
</tr>
<tr>
<td>5.2.1 Hanscom Air Force Base Experiment</td>
<td>126</td>
</tr>
<tr>
<td>5.2.2 Route 2 Experiment</td>
<td>129</td>
</tr>
<tr>
<td>5.2.3 Route 128 Experiment</td>
<td>133</td>
</tr>
<tr>
<td>5.2.4 Urban Experiment</td>
<td>137</td>
</tr>
<tr>
<td>5.3 Forest Experiment</td>
<td>141</td>
</tr>
<tr>
<td>5.4 Indoor Experiments</td>
<td>146</td>
</tr>
<tr>
<td>5.4.1 Belmont, MA Wood-Frame House Experiment</td>
<td>146</td>
</tr>
<tr>
<td>5.4.2 Sudbury, MA Wood-Frame House Experiment</td>
<td>149</td>
</tr>
<tr>
<td>5.5 Concluding Remarks</td>
<td>152</td>
</tr>
<tr>
<td>Chapter 6: Conclusions and Future Work</td>
<td>153</td>
</tr>
<tr>
<td>6.1 Conclusions</td>
<td>153</td>
</tr>
<tr>
<td>6.2 Future Work</td>
<td>154</td>
</tr>
<tr>
<td>6.2.1 Underwater Navigation</td>
<td>154</td>
</tr>
<tr>
<td>6.2.2 Underground Navigation</td>
<td>154</td>
</tr>
<tr>
<td>6.2.3 Indoor Calibration</td>
<td>154</td>
</tr>
<tr>
<td>6.2.4 Outdoor Calibration</td>
<td>154</td>
</tr>
<tr>
<td>6.2.5 Transmitter Positioning</td>
<td>155</td>
</tr>
<tr>
<td>6.2.6 Sensing</td>
<td>155</td>
</tr>
<tr>
<td>6.2.7 H-field Antenna</td>
<td>155</td>
</tr>
<tr>
<td>Appendix A: Boston-Area AM Radio Stations</td>
<td>157</td>
</tr>
<tr>
<td>A.1 Radio-Station Power</td>
<td>157</td>
</tr>
<tr>
<td>A.2 Antenna Coordinates</td>
<td>158</td>
</tr>
<tr>
<td>A.3 Antenna Field and Phase</td>
<td>160</td>
</tr>
</tbody>
</table>
Appendix B: Source Code ................................................................. 163
  B.1 Software Radio Program .......................................................... 163
    B.1.1 Header Files ................................................................. 163
    B.1.2 Source Files ............................................................... 173
  B.2 Navigation Algorithm Program ................................................... 205
    B.2.1 Header Files ................................................................. 205
    B.2.2 Source Files ............................................................... 222

Bibliography .................................................................................... 293
List of Figures

Figure 2.1: Block diagram of AM Navigation Receiver.......................................................... 24
Figure 2.2: Pre-amp circuit with protection circuitry.............................................................. 25
Figure 2.3: Power supply bypassing circuitry...................................................................... 26
Figure 2.4: Switchable gain, three-stage, op-amp based amplifier..................................... 27
Figure 2.5: Power supply distribution, gain control signal distribution, and peak detector circuit. ............................................................... 28
Figure 2.6: Calibration signal generator............................................................................... 30
Figure 2.7: Pre-amp circuit board....................................................................................... 32
Figure 2.8: Low-pass filter board....................................................................................... 33
Figure 2.9: Amplifier circuit board..................................................................................... 33
Figure 2.10: Pulse generator board.................................................................................... 34
Figure 2.11: Receiver front-end motherboard.................................................................... 34
Figure 2.12: Receiver front end......................................................................................... 35
Figure 2.13: Assembled receiver front-end......................................................................... 36
Figure 2.14: Power supply, gain control, and peak detection circuitry............................... 37
Figure 2.15: Fully assembled power supply, gain control, and peak detector circuitry box ............................................................................... 38
Figure 2.16: Clock circuit board......................................................................................... 39
Figure 2.17: External connections to clock circuit............................................................... 40
Figure 2.18: Clock cable transformer................................................................................ 41
Figure 2.19: Signal cable wrapped around a ferrite toroid ................................................ 42
Figure 2.20: AM navigation receiver................................................................................ 43
Figure 2.21: Back of AM navigation receiver.................................................................... 44
Figure 3.1: AM and GPS position estimates. The AM position estimates are obtained using an incremental navigation algorithm.......................... 50
Figure 3.2: AM and GPS position estimates vs. time. The AM position estimates are obtained using an incremental navigation algorithm.............. 51
Figure 3.3: Ambiguity function value over a square kilometer area for the correct trial value of the receiver clock offset ................................................................. 54
Figure 3.4: Ambiguity function value over a 100-square kilometer area for the correct trial value of the receiver clock offset ...................................................... 55
Figure 3.5: Ambiguity function value over a 100-square kilometer area for an incorrect trial value of the receiver clock offset ............................................ 56
Figure 3.6: AM ambiguity function scatter plot ................................................................. 59
Figure 3.7: AM ambiguity function positioning time plot ......................................................... 60
Figure 3.8: Unweighted post-fit ambiguity function component residuals ................ 61
Figure 3.9: AM ambiguity function positioning results and DGPS positioning results ............................................................................................................. 62
Figure 3.10: AM ambiguity function positioning results and GPS positioning results plotted as function of time ................................................................. 63
Figure 3.11: Plot of an ambiguity function over a one square kilometer area for the correct value of the receiver clock offset. The base receiver is at the origin and the rover is near –200 east, 200 north ............................................. 64
Figure 4.1: Error in AM station position coordinates reported in the FCC database ................................................................. 68
Figure 4.2: Model of WRKO’s three-element antenna ......................................................... 70
Figure 4.3: Simulated phase vs. azimuth angle for WRKO for various radii circles centered on WRKO’s center tower ............................................................. 71
Figure 4.4: Model-predicted phase minus NEC-simulated phase for WRKO at various radii ..................................................................................................... 73
Figure 4.5: WRKO phase modeling error caused by incorrect model parameters. The amplitude and phase of the SW tower are increased five percent and decreased by three degrees, respectively ............................................. 74
Figure 4.6: Map showing receiver and transmitter positions during the antenna pattern validation experiment, conducted on 18 July 2001. The locations of AM radio transmitters are indicated with a small antenna symbol labeled with the station’s frequency in kHz ............................................. 75
Figure 4.7: Post-fit phase residuals when antenna pattern model is not used .................. 76
Figure 4.8: Post-fit phase residuals when antenna pattern model is used .......................... 77
Figure 4.9: Difference between NEC-4 simulated phase over various grounds and perfect ground for both ends of the AM broadcast band .............................. 80
Figure 4.10: Map of receiver and transmitter positions during experiment one, conducted on 19 January 2002. This experiment is used to develop a ground model ..................................................................................................... 81
Figure 4.11: Post-fit phase residuals from experiment one .................................................. 82
Figure 4.12: NEC simulated phase and post-fit phase residuals from experiment one for station 740. Various grounds simulated ................................................. 83
Figure 4.13: Map of receiver and transmitter positions during experiment two, conducted on 22 January 2002. This experiment is used to develop a ground model. 85

Figure 4.14: NEC-simulated amplitude and the amplitude reported by the rover in experiments one and two. The frequency is 740 kHz. Various grounds simulated. 86

Figure 4.15: Map of receiver and transmitter positions during experiment two, conducted on 25 January 2002. This experiment is used to develop a ground model. 87

Figure 4.16: NEC-simulated amplitude and amplitude reported by the rover in experiments two and three. The frequency is 1550 kHz. Various grounds simulated. 88

Figure 4.17: NEC-simulated amplitude and amplitude reported by the rover in the antenna pattern experiment and experiment 2. The frequency is 800 kHz. Various grounds simulated. 89

Figure 4.18: Difference between NEC-4 simulated phase for grounds with varying permittivity and perfect ground for both ends of the AM broadcast band. 90

Figure 4.19: Post-fit phase residuals from experiment one when ground model is used. 93

Figure 4.20: Post-fit phase residual histogram for 27.5 minutes into experiment one. 94

Figure 4.21: Map of receiver positions during an experiment in which the rover was placed in close proximity to overhead conductors. The experiment was conducted on 28 January 2002. 95

Figure 4.22: Wide-area map of receiver and transmitter positions during an experiment in which the rover was placed in close proximity to overhead conductors. The experiment was conducted on 28 January 2002. 96

Figure 4.23: Post-fit phase residuals from a portion of an experiment in which the rover is near an isolated overhead guy wire. 97

Figure 4.24: Ratio between the amplitudes observed at the rover and base station from a portion of an experiment in which the rover is near an isolated overhead guy wire. 98

Figure 4.25: Multiple overhead conductors along Route 2a near the Brooks Historical Area (Minute Man National Historical Park) in Concord, MA. 99

Figure 4.26: Post-fit phase residuals from a portion of an experiment in which the rover is near multiple overhead conductors. 100

Figure 4.27: Ratio between the amplitudes observed at the rover and base station during a portion of an experiment in which the rover is near multiple overhead conductors. 101
Figure 4.28: Frequency difference estimation error during a portion of an experiment in which the rover is near multiple overhead conductors.

Figure 4.29: Multiple overhead conductors along the Battle Road within the Minute Man National Historical Park in Lexington, MA.

Figure 4.30: Post-fit phase residuals from a portion of an experiment in which the rover is near multiple overhead conductors.

Figure 4.31: Map base station and water pipe positions during an experiment in which the rover is near an underground water pipe. The experiment was conducted on 25 January 2002.

Figure 4.32: Post-fit phase residuals from an experiment in which the rover is near a two-meter-diameter steel-lined water pipe.

Figure 4.33: Map of receiver positions during an experiment in which the rover is moved over underground conductors. The experiment was conducted 12 February 2002 on Briggs Field, which is located on the MIT campus in Cambridge, MA.

Figure 4.34: Ambiguity function value during an experiment in which the rover is moved over underground conductors. The experiment was conducted 12 February 2002 on Briggs Field, which is located on the MIT campus in Cambridge, MA.

Figure 4.35: Photograph of Briggs Field, which is located on the MIT campus in Cambridge, MA.

Figure 4.36: Post-fit phase residuals from an experiment in which the rover is moved over underground conductors. The experiment was conducted 12 February 2002 on Briggs Field, which is located on the MIT campus in Cambridge, MA.

Figure 4.37: Frequency difference estimation error from an experiment in which the rover is moved over underground conductors. The experiment was conducted 12 February 2002 on Briggs Field, which is located on the MIT campus in Cambridge, MA.

Figure 4.38: Map of receiver locations during two experiments designed to examine the effects of skywave. The daytime experiment was conducted on 7 March 2002. The nighttime experiment was conducted on 16 April 2002.

Figure 4.39: Amplitude reported by the rover during a daytime experiment designed to examine the effects of skywave. The experiment was conducted on 7 March 2002.

Figure 4.40: Amplitude reported by the rover during a nighttime experiment designed to examine the effects of skywave. The experiment was conducted on 16 April 2002.

Figure 4.41: Frequency difference estimation error during a nighttime experiment designed to examine the effects of skywave. The experiment was conducted on 16 April 2002.
Figure 4.42: Post-fit phase residuals from a nighttime experiment designed to examine the effects of skywave. The experiment was conducted on 16 April 2002.

Figure 4.43: Scatter plot of rover position estimates during a daytime experiment designed to examine the effects of skywave. The experiment was conducted on 7 March 2002.

Figure 4.44: Scatter plot of rover position estimates during a nighttime experiment designed to examine the effects of skywave. The experiment was conducted on 16 April 2002.

Figure 5.1: Scatter plot of position estimates for a zero-baseline experiment conducted on 2 May 2001.

Figure 5.2: Post-fit phase residuals for a zero-baseline experiment conducted on 2 May 2001.

Figure 5.3: Map of rover positions during an experiment conducted on Hanscom Air Force Base on 13 July 2001.

Figure 5.4: Difference between AM and GPS position estimates during an experiment conducted on Hanscom Air Force Base on 13 July 2001.

Figure 5.5: Histogram of the difference between AM and GPS position estimates during an experiment conducted on Hanscom Air Force Base on 13 July 2001.

Figure 5.6: Ambiguity function value during an experiment conducted on Hanscom Air Force Base on 13 July 2001.

Figure 5.7: Map of base station and rover positions during an experiment conducted along Route 2 on 22 January 2002.

Figure 5.8: Difference between AM and GPS position estimates during an experiment conducted along Route 2 on 22 January 2002.

Figure 5.9: Histogram of the difference between AM and GPS position estimates during an experiment conducted along Route 2 on 22 January 2002.

Figure 5.10: Ambiguity function value during an experiment conducted along Route 2 on 22 January 2002.

Figure 5.11: Map of base station and rover positions during an experiment conducted along Route 128 on 20 December 2001.

Figure 5.12: Difference between AM and GPS position estimates during an experiment conducted along Route 128 on 20 December 2001.

Figure 5.13: Histogram of the difference between AM and GPS position estimates during an experiment conducted along Route 128 on 20 December 2001.

Figure 5.14: Ambiguity function value during an experiment conducted along Route 128 on 20 December 2001.

Figure 5.15: Map of base station and rover positions during an experiment conducted in an urban area on 8 February 2002.
Figure 5.16: Map of rover positions during an experiment conducted in an urban area on 8 February 2002. .............................................................. 138

Figure 5.17: Difference between AM and GPS position estimates during an experiment conducted in an urban area on 8 February 2002. ............. 139

Figure 5.18: Histogram of the difference between AM and GPS position estimates during an experiment conducted in an urban area on 8 February 2002. ........................................................................ 140

Figure 5.19: Ambiguity function value during an experiment conducted in an urban area on 8 February 2002. ...................................................... 141

Figure 5.20: Map of rover positions during an experiment conducted in a forest on 6 February 2002. ............................................................................. 142

Figure 5.21: Difference between AM and GPS position estimates during an experiment conducted in a forest on 6 February 2002. ...................... 143

Figure 5.22: Histogram of the difference between AM and GPS position estimates during an experiment conducted in a forest on 6 February 2002. ...... 144

Figure 5.23: Ambiguity function value during an experiment conducted in a forest on 6 February 2002. ........................................................................ 145

Figure 5.24: AM position estimate errors during an experiment conducted in the garage of a wood-frame house in Belmont, MA on 9 May 2001. ....... 147

Figure 5.25: Ambiguity function value during an experiment conducted in the garage of a wood-frame house in Belmont, MA on 9 May 2001. ........ 148

Figure 5.26: Rover phase error during an experiment conducted in the garage of a wood-frame house in Belmont, MA on 9 May 2001. ................. 149

Figure 5.27: AM position estimate errors during an experiment conducted in the garage of a wood-frame house in Sudbury, MA on 7 June 2001. ........ 150

Figure 5.28: Ambiguity function value during an experiment conducted in the garage of a wood-frame house in Sudbury, MA on 7 June 2001. ....... 151

Figure 5.29: Rover phase error during an experiment conducted in the garage of a wood-frame house in Sudbury, MA on 7 June 2001. ................. 152
List of Tables

Table 4.1: WRKO geometry and daytime electrical parameters. ................................. 70
Table 4.2: Table of permittivity and conductivity for NEC simulations. ....................... 79
Table 4.3: Relative permittivity and conductivity for 740 kHz simulations. ................. 83
Table 4.4: Ground parameters for NEC simulations designed to study the effect of permittivity variation. ........................................................................................................ 90
Table 4.5: Ground-model parameters for ground with conductivity 0.0015 mhos/m and relative permittivity five.......................................................... 92
Table A.1: Daytime, critical hours, and nighttime power levels for Boston-area radio stations. .................................................................................................................. 158
Table A.2: FCC-database antenna tower heights and surveyed WGS-84 coordinates for Boston-area radio stations 590 through 1090................................. 159
Table A.3: FCC-database antenna tower heights and surveyed WGS-84 coordinates for Boston-area radio stations 1120 through 1600................................. 160
Table A.4: Daytime antenna field and phase for Boston-area radio stations............... 161
Table A.5: Critical-hours antenna field and phase for Boston-area radio stations. ..... 161
Table A.6: Nighttime antenna field and phase for Boston-area radio stations. .......... 162
Chapter 1: Introduction

Navigation technology progresses slowly with only occasional breakthroughs. In the 18\textsuperscript{th} century the problem of the day was determining longitude at sea. It took John Harrison most of his life to convince the British Board of Longitude that an accurate clock was the answer [26]. For over 200 years sailors navigated as suggested by Harrison; and, to this day timekeeping is essential to navigation. At the turn of the 20\textsuperscript{th} century, Reginald Fessenden began experimenting with continuous wave (CW) radio and amplitude modulation (AM) [1,2]. Little did he know that his work on CW not only would be the basis of most modern radio communication but also would revolutionize the field of navigation.

Radionavigation progressed rather slowly from radio direction finding [3], to LORAN [24], and eventually to the Global Positioning System (GPS) [17], each advance constraining yet another degree of freedom. The success of GPS has focused almost all attention on the refinement of GPS in particular rather than the advancement of navigation technology in general. This attention is easy to justify because GPS is an amazing system. In fact, GPS is being used to determine positions far more accurately than the system designers ever intended. By means of differential GPS (DGPS) and carrier-phase observations, sub-centimeter level positioning is possible [5].

1.1 Navigation Using GPS

The standard method of navigating by GPS is to observe the pseudorandom ranging codes that modulate the carrier-waves of the signals transmitted by the GPS satellites. The time-delay of the code modulation of the signal received from a satellite is measured by correlating the signal with a locally generated replica of the code. Observation of the delay (often called the code “phase”) gives an effectively unambiguous measure of the signal transit time, which is, of course, related to distance between the transmitter and receiver by the speed of light. The designers of GPS intended users to observe the code-phase of at least four satellite signals with what we now call a stand-alone receiver. “Stand-alone” simply refers to a single receiver operating with no external information. Stand-alone, code-phase GPS navigation yields position estimates that have a 2\(\sigma\) accuracy of approximately \(\pm 15\) meters. Through the 1990’s the U.S. Department of Defense maintained a policy of Selective Availability (SA) that effectively limited the
accuracy available to civilian users to ± 150 meters [16]. This feature is currently
disabled so civilian users realize ± 15 meter positioning, but the DOD reserves the right
to turn SA back on if necessary.

1.1.1 Navigation Using Differential GPS

Partly to eliminate the effects of SA, but also to reduce the effects of other errors, civilian
users began using what is called differential GPS, or DGPS. In a DGPS setup one
receiver is placed at a known location (the base station) and communicates with another
receiver whose position is to be determined (the rover). The position of the rover is
determined with respect to that of the base station, from differences between satellite
observations made simultaneously by both receivers. The SA effects are perfectly
correlated at both receivers so they cancel in the differencing between receivers. Also,
atmospheric refraction effects and the effects on receiver position determination of errors
in assumed knowledge of satellite positions are correlated at locations that are close
together, so these effects tend to cancel in the relative-position determination. Over a
distance of several kilometers, meter-level accuracy in relative positioning is possible by
DGPS when code-phase observations are used [16]. For such short distances, multipath-
propagation effects dominate all other sources of error. Multipath error results when a
signal reflected from the ground or another nearby object interferes with the desired,
directly received, signal.

1.1.2 Navigation Using Observations of GPS Carrier Phases

In addition to code-phase observations, observations of the phases of the radio-frequency
carrier-waves of GPS signals may be used for position determination. Multipath error in
carrier-phase observations is typically two to three orders of magnitude smaller than
multipath error in code-phase observations [7]. This advantage is not without cost,
however. The interpretation of a carrier-phase observation in terms of position is
potentially ambiguous because one cycle of the carrier wave is practically
indistinguishable from the next, and the GPS carrier wavelength is rather short (19 or 24
centimeters). But, because the wavelength is so short, position-determination from
carrier-phase observations can be exquisitely accurate, within about one millimeter.

Many techniques have been invented to deal with the carrier phase ambiguity problem.
Some techniques require the receiver to be stationary for a period of time while the
ambiguities are resolved. Kinematic ambiguity resolution, a more recent technique,
allows the receiver to move while the ambiguities are being resolved. Effectively
continuous tracking of the carrier is required to navigate. If the receiver misses a carrier
cycle, or “loses lock,” the ambiguity-resolution procedure must be repeated and the
potential accuracy of carrier-phase positioning (as opposed to the coarser accuracy of
code-phase) is lost until the ambiguities are resolved again.

A few techniques have been developed that resolve the carrier-phase ambiguities
instantly using observations from only one epoch. These techniques are superior in that
they require neither lengthy initialization nor continuous tracking. However, these techniques require simultaneous visibility of a large number of satellites. Unfortunately, the number of satellites needed is typically more than the number available, so instantaneous carrier-phase positioning is often not possible [22]. Because instantaneous positioning is important for real-time navigation problems, precise carrier-phase positioning is used mostly for precise geodetic surveying.

### 1.2 Navigation Using Signals of Opportunity

Carrier-phase observations can be made of any CW signal including those that exist for purposes other than navigation. Such signals include those from the ubiquitous (and often very strong) radio and TV broadcast stations. Unfortunately, navigating using these signals of opportunity is not as straightforward as navigating using GPS carrier-phases for the following fundamental reasons:

1. A broadcast transmitter’s frequency and phase are not synchronized with any other transmitter’s.
2. A broadcast transmitter’s nominal frequency is distinct from the nominal frequencies of all other transmitters in the same geographic area.
3. Broadcast signals are not designed for navigation.

Every GPS satellite has several onboard atomic frequency standards that maintain accurate time and frequency synchronization among satellites. Signals of opportunity, on the other hand, do not generally require synchronization for their intended purpose.

GPS achieves channel separation using “code division multiple access” (CDMA), and all satellites transmit with the same carrier frequency. This equality enables carrier-phase ambiguities in position determination to be represented by a modest number of integers whose determination solves the ambiguity problem [16]. Most signals of opportunity use “frequency division multiple access” (FDMA), i.e., different transmitters transmit signals on different frequencies. Due to the frequency inequality, most if not all ambiguity-resolution techniques that have been developed for GPS are inapplicable.

### 1.3 Motivation

GPS is such an effective navigation tool that one might argue, there is no need to develop other ways to navigate. There are, however, some significant concerns regarding the use of GPS:

1. GPS is highly susceptible to jamming [27].
2. GPS does not work well indoors [21].
3. GPS does not work well under dense foliage.
4. GPS does not work under water.
5. GPS is a quite complicated system requiring sophisticated receiver technology.
6. GPS is basically a military system, controlled by the U.S. Department of Defense.
GPS is easy to jam because all GPS satellites broadcast on the same frequency and because GPS signals are exceedingly weak when they reach users on the ground. Signals of opportunity, on the other hand, often have very high signal strengths and are spread out in frequency, making them much harder to jam. This relative immunity to jamming is of particular interest not only to the military but also to users who rely on navigation systems as a matter of safety.

GPS does not work well indoors because the signals are weak, and because the GPS signals that are available for civilian use, having a frequency of 1575.42 MHz and corresponding free-space wavelength of approximately 19 centimeters, are absorbed or reflected by most building materials. Even when GPS reception is possible indoors, position estimates are highly degraded by multipath. Similar problems are encountered while using GPS under dense foliage. Signals at this wavelength also have a practically zero skin depth in both fresh and saltwater, so underwater navigation is nearly impossible.

Because of higher signal-to-noise ratios and lower frequencies, a navigation receiver using signals of opportunity could be considerably cheaper than a GPS receiver. Economies of scale have made GPS receivers very affordable, but from a technology standpoint, one should be able to make a signal-of-opportunity receiver more cheaply. Regarding cost, it should also be mentioned that little infrastructure is necessary to deploy a signal-of-opportunity navigation system because the transmitters already exist for other purposes.

GPS is controlled by the U.S. Department of Defense. While it is unlikely, because of political pressure, that the DoD would arbitrarily deny GPS service to the civilian community, the DoD explicitly states that it can deny or degrade service to civilians as it deems necessary. Most GPS users in the U.S. don’t give the possibility of denial or degradation a second thought, but elsewhere in the world it is a big enough issue that several European countries and Japan are considering launching satellites for their own system.
Chapter 2: Navigation Receiver

To explore the effectiveness and performance of a navigation system that uses only signals of opportunity, I designed and built two navigation receivers.

2.1 Frequency Band

I chose to use signals from the AM broadcast band for navigation. In the US, this band spans frequencies from 540 kHz to 1700 kHz with corresponding wavelengths from 555 meters to 175 meters [10]. This band is attractive for a number of reasons:

1. In most parts of the country, signals from many AM stations are available.
2. The long wavelengths may be more suitable for indoor and underwater navigation.
3. Low frequencies and low bandwidths simplify receiver design.

The first commercial radio broadcasting was in the AM band; and to this day, AM radio remains popular. In the Boston metropolitan area there are over 30 stations whose signals are usable for navigation.

Shorter wavelength signals such as those used for FM and TV broadcasting do not penetrate buildings as well, and are more susceptible to multipath error. The skin depth of AM signals in freshwater is about 10 meters, so navigation may be possible down to 30 meters or so. The skin depth in sea water is only about 20 centimeters [25]. This may at first seem too shallow to be useful, but in some military applications, navigating with an antenna slightly below the surface may be far more attractive than navigating with an antenna slightly above the surface.

The AM band is centered at about 1 MHz and spans only slightly more than 1 MHz. Electronic design and circuit board layout at these frequencies is straightforward. Also, the entire band can be sampled at a reasonable sampling rate without first downconverting.

2.2 AM Navigation System Considerations

A navigation system that uses unsynchronized transmitters must employ a reference receiver, in other words a “base station,” as with DGPS. Otherwise, phase variation with time at a fixed position cannot be distinguished from phase variation caused by position change. Therefore, I built two receivers. One serves as a reference and is placed at a
known location. The position of the other is determined from differences between the AM carrier phases observed locally and at the reference receiver.

Thus, determining the position of the roving receiver requires data from the base receiver. In a production system this information could be provided via a radiotelemetry link. To keep my system simple, I did not implement a radio link. Instead, each receiver stored time-tagged measurement data locally, for post-processing.

2.3 AM Navigation Receiver Hardware

In the design of these receivers, my primary objectives were to minimize complexity and to minimize sources of error. These two design objectives were often in conflict, in which case a suitable balance was sought.

Figure 2.1 shows a block diagram of each navigation receiver:

![Block diagram of AM Navigation Receiver](image)

In the following sections, the design and construction of each block is discussed.

2.3.1 Antenna

The antenna is a simple vertical whip whose base voltage is sensed with respect to the relatively high-capacitance “counterpoise” of the receiver chassis; in other words, it is a vertical E-field probe. The antenna is less than 1 meter long so it is much shorter than the wavelength of the signals it is intended to receive. An electrically short antenna is not only convenient; it also does not significantly perturb the phase or amplitude of the received signals.

A small loop antenna (in other words, a small B-field probe) could also be used in this application. However, because AM broadcast signals are vertically polarized, only one vertical E-field probe is required for an azimuthally omnidirectional sensor, whereas two orthogonal horizontal B-field probes would be required.
2.3.2 Pre-Amp

Because the whip antenna is much shorter than a wavelength, the source impedance of the antenna is very large, and essentially pure-negative-imaginary, i.e., capacitive. The input impedance of the receiver should also be large and pure-negative-imaginary, so that the gain and phase shift of the system is frequency-independent. The circuit in Figure 2.2 provides a suitably high-impedance input for the antenna.

![Pre-amp circuit with protection circuitry.](image)

The components labeled “PSB” in Figure 2.2 are power supply bypassing circuits. The following schematic shows the power supply bypassing that is used throughout the receiver.
The components labeled FB1 and FB2 in Figure 2.2 and Figure 2.3 are ferrite beads made by Panasonic, part number EXCELDR35. At 1 MHz, each bead has an inductance of about 3 µH. At 100 MHz, the beads are almost purely resistive with a resistance of about 150 ohms. The purpose and operation of the bypassing is straightforward. Further bypassing is provided where power is brought onto the circuit board as depicted in the lower left hand corner of Figure 2.2.

U1 in Figure 2.2 is a BUF04 IC buffer made by Analog Devices. It has unity gain and a very high input impedance. The BUF04 has good linearity up to 10 MHz which ensures good performance in the AM band.

The BUF04’s high impedance input needs to be protected from high-voltage spikes like those caused by static discharge, and from the high RF voltages that occur when the receiver is situated near a high-power transmitter. High-speed diodes D1 and D2 limit the input voltage to protect BUF04. Neon bulb B1, fuse F1, and resistor R1 protect the diodes in the case of a prolonged or excessively large voltage spike.

Capacitor C1 couples a calibration signal into the antenna input circuit. Calibration will be discussed in a subsequent section.

### 2.3.3 Low Pass Filter

The low pass filter prevents aliasing in the subsequent A/D conversion. I decided the maximum aliasing error I was willing to tolerate was 5 milliradians, which corresponds to about 15 cm at 1700 kHz. If one assumes that a potentially aliased signal has the same power and is 90 degrees out of phase with the desired signal, then the required attenuation is about 46 db. The maximum frequency of interest is 1700 kHz and the subsequent A/D conversion is at 5 megasamples/second. Therefore the minimum frequency that will alias into the band of interest is 3300 kHz. The low pass filter will have its cutoff at 1700 kHz so the filter needs to roll off at 160 db/decade to satisfy the 46 dB attenuation requirement.
2.3 AM Navigation Receiver Hardware

An 8-pole Butterworth filter meets this specification. Other filter types were considered for this application. The Bessel filter was particularly attractive because it has maximally flat group delay in the passband. Unfortunately, no Bessel filter can provide the required attenuation.

The filter also includes one high-passing pole-zero pair, with a zero at the origin and a negative-real pole at 50 kHz. High-pass filtering helps preserve the dynamic range of the receiver for the signals of interest. A 50 kHz cutoff was used instead of the more obvious 540 kHz cutoff because there are potentially useful signals below the AM band.

I had the filter built by Allen Avionics of Mineola, NY to the above specifications.

2.3.4 Amplifier

An amplifier with switchable gain is needed to fully utilize the dynamic range of the A/D converter. Figure 2.4 shows a schematic of the amplifier:

![Figure 2.4: Switchable gain, three-stage, op-amp based amplifier.](image)

The amplifier consists of 3 op-amp stages. Each op-amp is an Analog Devices type AD844. The AD844 was chosen for its excellent linearity at frequencies up through 10 MHz. Multiple stages were used to limit the gain required for each stage, which further enhances the linearity of the overall circuit.
The gain is controlled in each stage by shorting out selected feedback resistors with reed relays. Reed relays were chosen over semiconductor switches to ensure maximum linearity. Linearity is important to minimize errors due to internal generation of spurious signals such as harmonics and second- and third-order intermodulation products.

The overall gain of the amplifier is $2^n$ where $n$ can take on any integer value from 0 through 7. This yields a good range of gain while ensuring that no more than one bit of dynamic range is given up in the A/D converter.

### 2.3.5 Power Supply, Gain Control, and Peak Detection

Power supply distribution, gain control signal distribution, and a peak detector circuit are all provided on a single circuit board. The schematic is shown in Figure 2.5.

![Figure 2.5: Power supply distribution, gain control signal distribution, and peak detector circuit.](image)

An Analog Devices 925 supplies the power for all the circuits. The 925 is a linear power supply that converts 110 volts AC to +/- 15 volts DC. The power from the 925 is delivered to the board via J4 and J5. The board delivers power to local circuits and to J7, which provides power to other circuits.
A rotary 8-position switch provides the gain control signals to J6. These signals are filtered and sent on to J8, which connects to the amplifier.

An analog meter is used to display the peak voltage of the amplifier output signal. The peak detector circuit shown in Figure 2.5, comprised of the four op-amps of U1, provides the peak voltage signal. U1 is an OP467 made by Analog Devices. The output of the amplifier is connected via J1 to both the peak detector and to an AD844 op-amp that is configured to provide a gain of 2. The output of U2 drives the A/D converter. The gain of two compensates for the effect of the transmission line termination.

### 2.3.6 Calibrator

The high end of the AM band has a frequency that is over three times the frequency at the low end. The phase delay of the electronics will vary significantly through this wide frequency range. To the extent that both receivers are identical, this effect will cancel out. However, electronic parts have finite tolerances and each receiver will be in a different environment so some error caused by this effect could persist. To counteract this effect, I built some circuitry that can inject a calibration signal into the antenna terminals. The calibration signal is a periodic train of very narrow pulses in the time domain. The Fourier transform of an impulse train is an impulse train in the frequency domain. The phase of each of these impulses in the frequency domain is well defined so observations of this signal by the A/D converter can be used to determine the phase delay vs. frequency of the electronics. The high- and low-pass filter disperses the time-domain pulses so the pulses do not saturate the A/D converter. The pulse repetition frequency of the pulse train is chosen to minimize interference with broadcast AM stations but still provide enough harmonics throughout the band to fully characterize the phase delay vs. frequency of the circuit. Figure 2.6 shows the calibration circuit.
The circuit in Figure 2.6 is designed to take an external 10 MHz sine wave input at terminals J1. The sine wave is squared up with comparator U1 and then flip-flop U2a divides the frequency by two. The frequency of the 5 MHz square wave is further divided by the counters U3 and U4. The divisor is set by DIP switch SW1. Then U2b converts the pulse-train output of the counters to a square wave. Finally, the output transistor circuit converts the positive-going transitions of the square wave into extremely narrow (< 10 nanoseconds) negative-going pulses. This pulse train is loosely coupled into the antenna terminals of the receiver through a 10-pF capacitor as shown in Figure 2.2.

2.3.7 A/D Converter

I purchased the A/D converter from Datel in Mansfield, MA. The model number is PCI-416D. This model is capable of 5 Megasample/second sampling with 12-bit resolution and simply plugs into the PCI slot of a PC. Datel provides a Windows dynamically linked library to facilitate control and data acquisition.
2.3.8 Clock Circuitry

The A/D converter has an on-board crystal-oscillator frequency standard from which both a sampling clock and a burst-trigger clock are derived. The sample clock runs at 5 MHz. It is not necessary for the A/D converter to sample continuously because periodic bursts of samples are adequate for navigation purposes. The trigger clock controls when these bursts of samples are collected; it runs at 0.2 Hz. Thus, a burst of samples is taken every 5 seconds.

For testing purposes, it is useful to have a truth source available to evaluate performance. Position coordinate truth is obviously required to evaluate a navigation system. Less obviously, the epoch difference, or time delay, between the clock in the base receiver and the clock in the roving receiver, called the receiver clock offset, must also be estimated along with the position coordinates. While position truth is easily provided with a GPS receiver or a tape measure, clock truth requires more work.

To provide clock truth, highly stable clocks must be used for both receivers or the clocks in both receivers must be derived from the same source. Unfortunately, the clocks on board the Datel PCI-416D cards are not very stable and they are useless for providing clock truth. Even less fortunately, the Datel cards do not allow derivation of the sample clock and trigger clock from a single external frequency standard. So, if a more stable external clock is desired, signals for both the trigger clock and sample clock must be applied to the board.

I built a clock circuit that provides a 5 MHz sample clock and a 0.2 Hz trigger clock from a 10 MHz source. I used rubidium frequency standards for the 10 MHz sources since they were freely available for me to use. This circuit also allows both receiver clocks to be driven by the same 10 MHz frequency standard. Therefore, both methods of providing clock truth mentioned above can be implemented with this circuit: stable independent clocks and clocks derived from a common source.

I omit the schematic for this circuit because the design is similar to that of the pulse generator circuit described above. Because of the high number of binary stages required to divide a 10 MHz input down to 0.2 Hz, I used a programmable ASIC instead of discrete counters.

2.3.9 Computer

All radio functions, other than those described above, were implemented in software run on a standard Intel® PC. This arrangement is called a software radio. The advantage of this type of radio is that it is extremely flexible. The disadvantage is that it requires a considerable amount of computing power. Fortunately, the limited bandwidth of the AM band makes a software radio practical on a standard PC.

I built PC’s to run the radio software. To maximize performance while minimizing cost, I chose to use dual Intel Celeron processors. The processors were clocked at 550 MHz and each system had 256 MB of memory.
2.3.10 Construction

Good radio design requires not only good circuit devices and topology but also good physical layout. This section documents the physical layout of the navigation receivers and also some noise-suppression techniques that were employed.

The pre-amp, filter, amplifier, and calibration circuit were implemented on separate daughter boards. The pre-amp board is shown in Figure 2.7. This board is 144 millimeters by 62 millimeters including the tabs near the edge connector.

![Pre-amp circuit board.](image)

**Figure 2.7 : Pre-amp circuit board.**

The pre-amp board layout, like the board layout for all circuits in the receiver, is designed to minimize trace length. Although it is not visible in the picture, the entire back side of the board is a ground plane. Most components on the pre-amp board are easily identifiable so no labeling is overlaid. The SMA connector near the bottom of the figure is used to connect to the pulse generator circuit. The SMA connector in the upper left corner of the figure connects to the low pass filter.
As mentioned in a previous section, the low-pass filter was purchased rather than built from discrete components. The filter merely needed to be mounted onto the copper-clad board (144 x 62 millimeters) shown in Figure 2.8. The SMA connector on the left connects to the pre-amp circuit and SMA connector on the right connects to the amplifier circuit.

The three stages of the amplifier are easily recognizable in Figure 2.9. The four long, black boxcar-shaped components are the reed relays. This board is 144 millimeters by 62 millimeters including the tabs near the edge connector.
The pulse generator is shown in Figure 2.10. To further reduce trace length because of the high-speed digital signals on this circuit board, IC sockets were not used. The red DIP switch on the right side of the board is used to set the pulse repetition frequency of the calibration signal. This board is 144 millimeters by 62 millimeters including the tabs near the edge connector.

The pre-amp, filter, amplifier, and pulse-generator circuit boards all plug into the motherboard shown in Figure 2.11. The motherboard provides power and ground to all...
circuit boards and also provides signal connections to the outside world. The motherboard is 94 millimeters by 57 millimeters.

![Figure 2.12: Receiver front end.](image)

With all circuit boards plugged into the motherboard, the entire assembly slides into a slotted aluminum box (Pomona Model 3743) shown in Figure 2.12. The box is 156 x 105 x 68 millimeters, not including the connectors. The “BASE” label on the box indicates that this particular front-end box is used for the base receiver. The piece of aluminum with the threaded stud protruding out of it, on the extreme right side of the picture, is the antenna terminal. The antenna terminal is as close as possible to the pre-amp circuit to minimize the capacitance of this high-impedance connection.

The motherboard/daughter board design provides modularity. Also, the daughter boards are arranged to minimize unwanted capacitive coupling between the various stages in the front-end circuit. Unwanted capacitive coupling is arguably not much of a problem in the band of interest because the impedance of the unwanted capacitance is high. However, the antenna receives signals from all frequency bands. Also, the pulse generator circuit produces harmonics of the pulse repetition frequency up to 100 MHz. Capacitive coupling of higher frequency signals around the low-pass filter would be probable without careful circuit layout. Obviously, these higher frequency signals could cause problems if aliased into the band of interest.
In Figure 2.12, the daughter boards from top to bottom are the pulse generator, pre-amp, filter, and amplifier. This daughter board sequence was used so that the ground planes would act as shields between the various stages. In particular, the pre-amp and the pulse generator, which carry unwanted high frequency signals, are shielded from the amplifier, which only has signals in the band of interest.

To further prevent unwanted capacitive coupling, only low impedance 50-ohm transmission lines were used throughout the circuit.

Figure 2.13: Assembled receiver front-end.

The assembled receiver front-end is shown in Figure 2.13 with the whip antenna attached. The connector in the upper-left corner is used to supply power and gain-control signals. The lower-left connector is the output and the lower-right connector is used to supply a 10 MHz sine wave signal to the pulse generator. The two tabs that can be seen attached to the case and antenna terminal are used to inject a synchronization pulse at the beginning of a navigation experiment.
2.3 AM Navigation Receiver Hardware

Figure 2.14: Power supply, gain control, and peak detection circuitry.
Chapter 2: Navigation Receiver

The power supply, gain control, and peak detection circuitry is shown in Figure 2.14. The aluminum box (LMB/Heeger Model KAB-3743) is 188 x 119 x 78 millimeters, not including the components on the outside of the box. The Analog Devices 925 power supply is easily visible in the lower-left hand corner of the box. A two-section, LC, AC-line filter, attached to the right side of the box, prevents unwanted RF signals from entering the box through the power line. The plug housing in the lower-right hand part of the box also contains a line filter. The three spade connectors that are visible hanging over the lower side of the box connect to a lighted power switch on the box cover.

The circuit board in Figure 2.14 contains the peak detection circuitry and filtering for both power and gain control signals. The 4-pin header connector in the lower-right part of the circuit board connects to an 8-position gain control switch on the box cover. The 2-pin header connector in the upper-middle part of the circuit board connects to an analog meter on the box cover.

The left SMA connector on the upper side of the box carries the output signal to the A/D board. The right SMA connector is the input from the front-end circuitry. The other connector on the upper side of the box is a DB9 that provides power and gain control signals to the front-end circuitry. The fully assembled box is shown in Figure 2.15.

Figure 2.15: Fully assembled power supply, gain control, and peak detector circuitry box.
The circuitry that converts a 10 MHz sine wave into both sample clock and trigger clock signals is shown installed in its enclosure in Figure 2.16.

**Figure 2.16: Clock circuit board.**

The IC’s on the board in Figure 2.16 are, from left to right, a comparator, a programmable ASIC, and a line driver. The square-shaped component in the lower-left part of the board is a switching DC-DC converter. The connector on the right connects board signals to external connectors. The various signals are discussed below. This box also contains a switching power supply and a 10 MHz rubidium frequency standard.
The clock circuitry is designed so that sample clock and trigger clock signals can be generated from either internal or external 10 MHz sine-wave sources. The switch marked “INT” and “EXT” selects the source. The DB9 marked “10MHz REF” provides a 10 MHz sine-wave output from the internal clock on two pins and accepts a 10 MHz sine-wave input on another two pins. Likewise, the DB9 marked “RESET” provides a reset signal on two pins and accepts a reset signal on 2 different pins. This design allows either clock to be slaved to the other clock. The reset signal is simply a 1 Hz pulse that is used to simultaneously zero the internal clock counters on both clock circuits. This ensures synchronous triggering in both receivers. The DB9 connector marked “CLOCK OUT” provides a 5 MHz sample clock on two pins and a 0.2 MHz trigger clock on another two pins. These signals drive the A/D conversion.

Slaving one clock to the other is more difficult than one might assume. A cable used to connect the two clocks will adversely affect the phase measurements at both receivers. The cable acts like an antenna and provides an undesired signal path into the receiver. In other words, the cable extends the counterpoise of a receiver’s antenna, so that the effective position of the E-field sensing is displaced. To overcome this obstacle, I broke the electrical connection at each end of the clock cable with a transformer, shown in Figure 2.18. The board to which the transformer is attached is 60 millimeters by 42 millimeters.
The transformer core is K-type ferrite, which is good for 10 MHz transformers. The primary and secondary windings each have 10 turns of wire. The capacitor value was chosen so that the circuit formed by the capacitor and the transformer leakage inductance resonates near 10 MHz.

The clock cable is particularly insidious because of its length, but every cable acts like an antenna and can offer unwanted signal paths into the receiver. It is impractical and luckily unnecessary to install transformers on every cable. The next best thing to a transformer is to put several turns of the cable around a ferrite toroid as shown in Figure 2.19.
Figure 2.19: Signal cable wrapped around a ferrite toroid.

The toroidal core (Amidon Model FT-240-77) is made of type 77 ferrite which is good for attenuating unwanted signals in the AM band. The outside diameter of the core is 70 millimeters and the height is 12.7 millimeters. The winding is designed to maximize the common-mode impedance which requires a balance between the conflicting goals of maximizing the number of turns and minimizing capacitive coupling [20]. A ferrite core like the one in Figure 2.19 was placed on every cable.

A complete AM navigation receiver is shown in Figure 2.20.
The box on the left side of the teacart’s lower shelf is a uninterruptible power supply (UPS) that is used to power the receiver. All system components are secured to the
teacart with rope to prevent shifting during transportation. The back of the receiver is shown in Figure 2.21.

Figure 2.21: Back of AM navigation receiver.

A close look at Figure 2.21 reveals that every power cord has both a 2-stage line filter and a ferrite toroid. Every other cable has a ferrite toroid. The method used to take up excess cable length minimizes not only length but also the cross-sectional area of any loops that are created.

2.4 AM Navigation Receiver Software

The radio receiver software converts the data collected by the A/D converter into estimates of frequency, phase, and amplitude for every AM station. The mathematics of this conversion is discussed in this section. A majority of the software development effort was spent dealing with board interface issues. These issues are omitted from this
discussion because they are not interesting in the context of AM navigation. However, the source code for the radio software is included in Appendix B for those who are interested in these details.

### 2.4.1 Data Collection Parameters

A number of data collection parameters are relevant to the following discussion. As mentioned earlier, the trigger rate is 0.2 Hz. This value was chosen based on some basic assumptions about the dynamics of the receivers. The base receiver is assumed to be stationary. The maximum acceleration of the rover is assumed to be 0.5 g. The maximum unpredictable position change that can be tolerated during incremental navigation (to be discussed in a later chapter) is about 90 meters. If an unpredicted 0.5-g step in acceleration occurs immediately after one epoch, then the next observation epoch must occur within 6 seconds to satisfy the above requirements. A 0.2-Hz trigger comfortably satisfies this requirement.

The number of samples collected at each trigger is $2^{22}$. A power of two is chosen to reduce the computational burden of the FFT (discussed below). Given the 5 MHz sample clock, $2^{22}$ samples results in a coherent integration time of about 0.8 seconds and a corresponding discrete Fourier transform resolution of just over 1 Hz. This resolution is more than adequate to reject adjacent-channel interference. As it turns out, this is also near the maximum number of samples that the computer can process in 5 seconds.

### 2.4.2 Data Processing Definitions

In this section I define a number of terms to clarify the following discussion on data processing. The data are simply the real, time-series samples digitized by the A/D converter. The DTFT (discrete-time Fourier transform) is the complex spectrum of the data. An FFT (Fast Fourier Transform) is any of a number of methods that efficiently compute equally spaced samples of the DTFT. The SCS (sampled complex spectrum) is the result of the FFT operation. The SCS is a sampled version of the DTFT.

### 2.4.3 Data Processing Steps

The data processing steps are as follows:

1. Apply a window function to the data.
2. Perform an FFT on the windowed data.
3. Find the peaks of the magnitude of the SCS near AM station nominal frequencies.
4. Cross correlate the SCS with a pre-computed high-resolution Fourier transform of the window function.
5. Find the peaks of the cross correlation to refine the AM station frequency estimates.
6. Estimate the phase and amplitude at each of these frequencies.
2.4.3.1 Window Function

The first data processing step is to multiply the length-N time-series data by a length-N window function. A Blackman window was chosen because its Fourier transform has very small sidelobes. The downside of the Blackman window is that its Fourier transform has a wide mainlobe. This tradeoff is well suited to the problem at hand because there is no interference very close in frequency to the target carrier. The formula for a Blackman window is:

\[ w(n) = 0.42 + 0.5 \cos\left(\frac{2\pi}{N-1} n\right) + 0.08 \cos\left(\frac{4\pi}{N-1} n\right) \]

where \( N \) is the total number of samples in the data, \( 2^{22} \), and \( n \) ranges between 0 and \( N-1 \).

2.4.3.2 Fast Fourier Transform

After the window is applied to the data an FFT is performed. It is not obvious that an FFT is necessary. After all, the complex amplitude is only needed in a few spots in the AM band. The problem is that the frequencies of interest are not known a priori because of poor receiver and transmitter frequency accuracy and stability. Therefore, one must search for the amplitude peaks in the spectrum near the transmitter nominal frequencies to determine the apparent carrier frequencies. The FCC only requires AM station carrier frequencies to be within 50 ppm of nominal [10]. The receiver clock is specified by Datel to be within 20 ppm of nominal. Therefore, to determine the carrier frequency of an AM station that has a nominal frequency of 1 MHz, one should search a 140 Hz wide window centered on 1 MHz. If a step size of 1 Hz is used for the search, 140 steps are required. So even if the processing of only one station is considered, an \( N(\log_2 N) \) FFT algorithm should be more efficient.

I chose to use a freely available FFT package called FFTW [14]. FFTW is implemented in a C library. This library includes routines that actually measure the performance of various FFT algorithms. These routines allow the library to optimize itself for the particular application and platform. I found FFTW to be extraordinarily fast and easy to use. After just a few simple tests, it was immediately apparent that calculating an entire FFT using FFTW was much faster than evaluating the DTFT at only the necessary frequencies.

2.4.3.3 Carrier Search

After the FFT is computed, the magnitude of the SCS is searched near the AM station nominal frequencies for maxima which correspond to the AM carriers. The indices of these maxima correspond to estimates of the carrier frequencies. The samples of the DTFT that the FFT provides do not necessarily fall directly on top of the AM carriers. So, the frequency estimate derived from the index of the maximum of the amplitude spectrum can be in error by as much as one half the resolution of the FFT, 0.625 Hz. For
reasons that will become apparent in another chapter, a better frequency estimate is needed.

2.4.3.4 Carrier Frequency Refinement

One way to refine the frequency estimate is to apply a local maximization algorithm to the DTFT near the maxima of the magnitude of the SCS. This will arguably produce the best frequency estimate. However, the computational burden of evaluating the DTFT (and perhaps its derivative) at even one frequency is quite significant. Local maximization algorithms are therefore impractical for this purpose.

Another way to refine the frequency estimate is to fit a curve to the SCS near the maxima. The computational burden of maximizing the fitted curve is trivial. Furthermore, the frequency estimate should be quite good because an a priori functional shape of the SCS near the maxima is known: it is the Fourier transform of the window function.

The Fourier transform of a Blackman window can be expressed as:

\[ \mathcal{F}_R(e^{j\omega}) = 0.42 \mathcal{F}_R(e^{j\omega}) - 0.25 \mathcal{F}_R(e^{j(\omega - \frac{2\pi}{N-1})}) - 0.25 \mathcal{F}_R(e^{j(\omega + \frac{2\pi}{N-1})}) + 0.08 \mathcal{F}_R(e^{j(\omega \frac{4\pi}{N-1})}) + 0.08 \mathcal{F}_R(e^{j(\omega + \frac{4\pi}{N-1})}) \]

where \( N \) is the total number of samples and:

\[ \mathcal{F}_R(e^{j\omega}) = e^{-j\omega \frac{N-1}{2}} \frac{\sin(\omega N)}{2} \sin(\omega \frac{N}{2}) \]

is the Fourier transform of a rectangular window [19].

Since the Fourier transform of a Blackman window can be calculated explicitly, it can be cross-correlated with the complex spectrum computed from the windowed data. The index of the maximum of the magnitude of the cross-correlation can be used to enhance the frequency estimate. Since the Fourier transform of the window is available as a continuous function of frequency, the resolution of the resulting frequency estimate is not constrained.

In practice a couple of simplifications are made. Since the sidelobes of a Blackman window spectrum are so small, only the mainlobe is considered in the cross-correlation. Also, the mainlobe of the Fourier transform of the window is sampled such that the resulting number of samples is some multiple of the number of samples that would result if an FFT was taken of the window. This simplifies the indexing. It also limits the resolution of the resulting frequency estimate but if the number-of-samples multiplier is high enough the loss of resolution is acceptable.
2.4.3.5 Carrier Phase and Amplitude

Finally, the amplitude and phase are determined. Since the Blackman window is symmetric, applying the window does not affect the phase spectrum. However, FFTW does not apply a symmetric FFT. The result is that the SCS computed from the data is multiplied by the linear phase factor $e^{j\pi n}$ where $n$ is the index of the SCS. This linear phase factor is easily undone by changing the sign of the SCS samples that have odd indices. To calculate the phase, I use the formula:

$$\Phi = \arg(X[k-1] - X[k] + X[k+1]) \text{ for } k \text{ odd}$$

$$\Phi = \arg(-X[k-1] + X[k] - X[k+1]) \text{ for } k \text{ even}$$

where $X[k]$ is the SCS sample with maximum magnitude. The amplitude is computed by fitting a parabola to the amplitude of the SCS and evaluating the fit parabola at the frequency estimated from the cross-correlation.
Chapter 3: AM Navigation Algorithms

In this chapter, two different methods of navigation using AM broadcast signals are presented. The first method is called incremental navigation and involves computing position increments from observed, unambiguous carrier-phase increments. The second method is called instantaneous navigation, which as the name implies, involves computing positions from a snapshot of ambiguous carrier-phase measurements. After each technique is described, the results of applying it to experimental data are shown. Most of this chapter has been published in [15].

3.1 Incremental Navigation

A phase measurement is ambiguous by an unknown whole number of cycles. This ambiguity is not a problem if the initial position of the receiver is known and continuous tracking is possible throughout the time navigation is required. Continuous tracking simply implies that the phase is sampled often enough that the evolution of the phase over time is unambiguous. That is, no integer-cycle change could go undetected.

3.1.1 Algorithm Description

Unambiguous phase increments can be constructed from the fractional-cycle phases observed at each epoch. This process is called phase connection and simply requires combining the measured fractional phase and a whole number of cycles calculated by multiplying the observed frequency by the epoch period and adding any predicted motion effects. The dynamics of the roving receiver and the stability of the transmitter and receiver oscillators imply a maximum epoch period for which the phase connection will be successful. As shown in a previous chapter, a 5-second epoch period is sufficiently small for FCC-specified AM transmitter stability and less than 0.5-g dynamics.

A distance increment can be formed by multiplying the phase increment by an appropriate wavelength divided by $2\pi$. For a given transmitter signal, a single difference of distance increments can be formed by subtracting the distance increment computed from measurements made by the base receiver from the corresponding roving receiver distance increment. A double difference of distance increments can then be formed by differencing single differences corresponding to different transmitter signals. Doubly-
differenced distance increments are related to the relative position increment in a straightforward way. A simple least-squares algorithm can be used to estimate the position increment from the doubly-differenced distance increments [16].

### 3.1.2 Experimental Validation

I tested this method of navigation on the ramp at the Lincoln Laboratory Flight Facility in Bedford, MA. This site was chosen for its flat surface and lack of overhead conductors. The experiment was conducted on 25 September 2000. The base receiver was placed on the ground near the edge of the ramp at a GPS-surveyed location (WGS-84: N42.46536 W71.29919). I also surveyed four waypoints to visit with the roving receiver. The rover was placed in the back of a truck and driven to each of the four waypoints. This route was driven three times. Figure 3.1 and Figure 3.2 show the results. In the scatter plot (Figure 3.1) the blue diamonds represent the AM tracking results and the green squares represent the GPS waypoints. There is no GPS track in this plot.

The waypoints were surveyed using a Garmin GPS III handheld receiver with an RDS FM subcarrier DGPS receiver. This system was very convenient but not very accurate. Repeated position estimates from this system scatter by 5-10 meters. The Garmin receiver was almost as accurate without the DGPS receiver.

The figure shows the path of the truck as it visited each of the four waypoints. A small cluster of blue diamonds can be seen near each GPS waypoint. The following time plot...
(Figure 3.2) shows more clearly the fact that the truck stopped at each waypoint. The GPS track is constructed from just the four waypoints.

![Graph of AM and GPS position estimates vs. time.](image)

**Figure 3.2:** AM and GPS position estimates vs. time. The AM position estimates are obtained using an incremental navigation algorithm.

The figures show that the repeatability of the AM position estimates is quite good. The estimates differ from the GPS survey by as much as ten meters. Not all of this discrepancy is due to the AM estimates. As mentioned previously, the DGPS system used for the survey is only repeatable within 5-10 meters. The repeatability of the GPS position estimates cannot be judged from Figure 3.2 because the three waypoints were only surveyed once. In the figure, the GPS position estimates are merely repeated for each set of waypoint visits.

### 3.2 Instantaneous Navigation

Incremental navigation is an effective way of sidestepping the ambiguous nature of phase observations. The obvious weakness of such a technique is the required signal tracking continuity. Also, the initial position of the rover is required. Neither requirement is always practical to satisfy. Clearly, a navigation system that does not have such restrictions is much more desirable.

Fortunately, there are alternatives. This ambiguity problem is of course not unique to AM navigation. Researchers have studied GPS carrier-phase ambiguity resolution.
extensively since the extreme precision of such measurements was demonstrated in 1980 [8]. Since then, many ambiguity-resolution methods have been developed for GPS. All of these methods fall into essentially two classes: measurement-space methods and position-space methods. The first class of methods involves searching measurement space for an optimal set of doubly-differenced carrier phase integer ambiguities. The ambiguities in doubly-differenced GPS carrier phase observations turn out to be integers because all GPS satellites transmit on the same frequency. Because this class of methods relies on all transmitters to broadcast on the same frequency, this class of methods can, in general, be ruled out for use in multi-frequency systems. Clever researchers did, however, figure out a way to preserve the integer nature of the ambiguities for the GLONASS system, which does not use a single frequency [9]. This technique may be suitable for adapting to AM navigation but we have not yet tried it. The other class of GPS ambiguity resolution methods involves searching position space to optimize a cleverly designed function. We adapted one such method [10, 11] for use with the AM system.

3.2.1 Algorithm Description

At each epoch, the phase, frequency, and amplitude of the signal received from each transmitter are measured at the base receiver, \( b \), which is placed at a known position. These transmitters’ signals are also observed \( t \) seconds later by a roving receiver, \( r \). The epoch offset, \( t \), is the unintentional departure from synchronization of the receiver clocks. The phase, frequency, and amplitude data are collected and stored by each receiver. After an experiment has been completed, the data from the base and roving receiver are brought together for processing. The 2-dimensional position (\( x, y \)) of the roving receiver and the epoch offset, \( t \), are determined by maximizing an ambiguity function, which is a function of these unknowns and of the known, measured, phases. The basic idea is to design the ambiguity function such that it is maximal only for the correct values of \( x, y \) and \( t \).

To simplify the following equations, all positions (including transmitter positions) are expressed in a coordinate frame centered on the known base receiver position. Phase is defined to increase as distance from the transmitter decreases. The observed fractional phases are differenced between receivers for each transmitter, \( j \):

\[
\theta^j = \theta^j_r - \theta^j_b
\]

For trial values \( \hat{x}, \hat{y}, \) and \( \hat{t} \) of \( x, y, \) and \( t \), we also calculate a theoretical phase difference:

\[
\hat{\theta}^j(\hat{x}, \hat{y}, \hat{t}) = \hat{\theta}^j_r(\hat{x}, \hat{y}, \hat{t}) - \hat{\theta}^j_b
\]

where

\[
\hat{\theta}^j_b(\hat{x}, \hat{y}, \hat{t}) = -k^j_b \sqrt{(\hat{x} - x^j)^2 + (\hat{y} - y^j)^2} + \omega^j_b \hat{t}
\]
and

\[ \hat{\theta}_b^j = -k_b^j \sqrt{(x^j)^2 + (y^j)^2} \]

In the above equations, \( \omega_b^j \) is the radian frequency of transmitter \( j \) as observed by the base receiver.

\[ k_b^j = \frac{\omega_b^j}{c} \]

is the wavenumber of transmitter \( j \) where \( c \) is the speed of light. Notice that the “theoretical” phase difference depends on a measured quantity, the frequency, \( \omega_b^j \), of the signal from transmitter \( j \) observed by the base receiver. When the trial values \( \hat{x}, \hat{y}, \) and \( \hat{t} \) equal the actual values of \( x, y, \) and \( t \), the measured phase difference will equal the theoretical phase difference modulo \( 2\pi \), except for (hopefully small) measurement error. An ambiguity function is formed from the measured and theoretical phase differences from \( J \) transmitters:

\[ R(\hat{x}, \hat{y}, \hat{t}) = \sum_{j=1}^{J} W^j \cos(\theta^j - \hat{\theta}^j(\hat{x}, \hat{y}, \hat{t})) \]

The weights, \( W^j \), are normalized so they sum to one. In principle, the ambiguity function will equal one when the correct trial values are chosen. The argument of the cosine function is the difference between the measured and theoretical phase differences for transmitter \( j \). So, if the theoretical phase difference equals the measured phase difference modulo \( 2\pi \), the cosine function will equal one. For any transmitter \( j \), the cosine function will equal one for many trial values \( \hat{x}, \hat{y}, \) and \( \hat{t} \), not just the correct trial values. However, in the absence of measurement errors, the cosine functions for all transmitters will equal one if and only if the correct trial values are chosen.

I experimented with several weighting schemes including schemes based on received power and signal-to-noise ratio. Experimental data show, however, that SNR and received power are not correlated with measurement errors as long as there is adequate SNR to distinguish the AM carrier from potential interfering signals. Therefore, I use equal weights for all transmitter signals that have adequate SNR (usually 6 dB) at both the rover and the base.

I also experimented with variations on the ambiguity function described above in an attempt to deal with some of the challenges that will be discussed in subsequent chapters. I tried the alternative ambiguity function,

\[ R(\hat{x}, \hat{y}, \hat{t}) = \sum_{j=1}^{J} W^j (0.5 + 0.5 \cos(\theta^j - \hat{\theta}^j(\hat{x}, \hat{y}, \hat{t})))^\alpha \]
Chapter 3: AM Navigation Algorithms

for various values of the parameter $n$. In principle, as $n$ increases this ambiguity function becomes more sensitive to small deviations from the correct trial values. In general, I found that this variation did not significantly improve positioning performance. Additionally, I tried an “odd-man-out” technique in which the signal with the smallest cosine value is eliminated and the ambiguity-function search is repeated. This technique also did not significantly improve positioning performance.

In Figure 3.3, I plot values of the ambiguity function over a square kilometer area in the $\hat{x}, \hat{y}$ plane for the correct trial value of $\hat{t}$. The data used in this plot are from a 2-meter baseline experiment. The baseline is simply the distance between the rover and base. Negative ambiguity function values are plotted in blue. Positive values of the ambiguity function are plotted as indicated in the colorbar.

![Ambiguity function value over a square kilometer area for the correct trial value of the receiver clock offset.](image)

The plot shows a towering peak in the ambiguity function over the correct position of the roving receiver (approximately the origin). The peak value is 0.99. Other local maxima do not exceed 0.4, so the correct position is quite clear. The plot is generated using measurements from a single epoch. Single-epoch ambiguity resolution is the holy grail of carrier phase navigation. For the skeptics, I include Figure 3.4, which extends the position search area to 100 square kilometers.
3.2 Instantaneous Navigation

Figure 3.4: Ambiguity function value over a 100-square kilometer area for the correct trial value of the receiver clock offset.

Extending the position search area does reveal local maxima that are near 0.6, but the correct maximum near the origin is still very distinct. Figure 3.5 shows the ambiguity function for a wrong value of the receiver clock offset, $t$. 
As desired, no peaks above 0.6 are found in Figure 3.5. However, it should be noted that the ambiguity function is quasi-periodic in $\hat{t}$ with period 100 $\mu$s. The quasi-periodicity is due to the fact that the nominal AM broadcast frequencies are all harmonics of 10 kHz. The 100 $\mu$s periodicity is slightly modulated due to transmitter frequency deviation from nominal 10 kHz harmonics. In practice, this modulation does not sufficiently distinguish the correct value of $\hat{t}$ from values of that are 100$n$ $\mu$s from the correct value, where $n$ is an integer. Therefore the receiver clock offset must be known a priori to within 100 $\mu$s of the correct value.

In a real-time system, the required a priori synchronization knowledge could easily be provided by the data link. In a post real-time system, like the one I built, the problem is more difficult.

One solution to the synchronization problem is to have both receivers record the modulation of one pre-determined transmitter. The transmitter’s modulation observed at one receiver is just a time-shifted version of the modulation observed at the other receiver. If the receiver clock offset does not exceed the duration of the sample burst (~0.8 seconds), the index of the maximum of the cross-correlation of the modulation observed at each receiver will yield an unambiguous estimate of the receiver clock offset. Better estimates will result if the receiver clock offset is small. There is, of course, no
3.2 Instantaneous Navigation

reason why the modulation of multiple pre-determined transmitters can’t be used to improve the estimate of receiver clock offset. 

Another possible solution to the synchronization problem is to record, at each receiver, the frequency, phase, and amplitude of some number, say 10, of strongest peaks in the modulation sidebands of one pre-determined radio station. The long correlation times and distinct spectral features that characterize music and voice modulation insure that a reasonable number of the strongest modulation peaks observed at one receiver will be among the strongest modulation peaks reported at the other receiver. If both receivers record an observation of the same spectral feature, the observations can be used in the ambiguity function precisely as additional carrier observations would be used. A measured phase difference can be formed from the phase observed by each receiver. The theoretical phase difference for every modulation spectral feature will be the same as the theoretical phase difference for the associated carrier. Because the frequencies of the modulation spectral peaks will be randomly spaced, the observations will not contribute to the 100 µs periodicity and therefore, t can be unambiguously estimated. Modulation spectral peaks from more than one pre-determined radio station can be used in the same manner to improve estimates of x, y, and t.

I chose a simpler solution to the synchronization problem. At the beginning of each experiment, a short pulse is injected simultaneously into both antenna terminals. Each receiver records the time, according to its sample clock, when the pulse appears in the sample stream. The receiver clock offset, t, at the time the short pulse occurs, can easily be computed post real-time from these observations. Subsequent values of the clock offset can be predicted well within the required precision by using the ratio of the observed frequencies of a particular transmitter at each receiver.

It is not practical to search all possible positions for the maximum of the ambiguity function. The problem of determining an appropriate search space is not unique to AM navigation. Many GPS carrier-phase ambiguity resolution techniques also require searching either position or measurement space. With GPS, the problem is usually solved by initializing and bounding the search variables using the C/A-code position estimate and C/A-code position estimation error statistics, respectively. For AM navigation, other possibilities exist. Because of the close proximity of the transmitters, received power differences can be used to initialize the search. I have not tested this method at the time of writing. For the validation experiments described in this chapter, I simply searched a square kilometer area centered on the base station position. The receiver clock offset, t, was searched over a 10 µs window centered on the a priori estimate. I use a step size equivalent to ten percent of the shortest wavelength in the AM band.

For the longer baseline and higher dynamics experiments described in subsequent chapters, I used a more complex algorithm for initializing and bounding the search space. I initialized the search variables using a Kalman predictor. The acceleration of each variable (x, y, and t) is modeled as a first-order Gauss-Markov process. The inputs to the predictor are the ambiguity function position estimates. The measurement covariance matrix is dynamically adjusted using the ambiguity function value associated with the
input position estimate. It is important for the reader to understand that the Kalman predictor is used only to initialize the ambiguity function search. None of the AM navigation positioning results presented in this thesis are filtered in any way. I refer the reader to the source code found in Appendix B.2 for more details.

Starting at the estimate provided by the Kalman predictor, I use a local optimization algorithm to find a local maximum of the ambiguity function. Since a closed-form expression for the derivative of the ambiguity function is easily derived and requires little additional computation, I use a quasi-Newton optimization algorithm that makes use of the derivative. Specifically, I use the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [23]. I refer the reader to the source code in Appendix B.2 for more details.

If the value of the ambiguity function at the local maximum found by the BFGS algorithm exceeds 0.8 then it is assumed to be the global maximum. I determined empirically that maxima of the ambiguity function that are greater than 0.8 have a very high probability of being associated with the correct trial values. If the value of the ambiguity function at the local maximum is less than 0.8, then the BFGS algorithm is executed starting from an equally spaced grid of points centered on the Kalman predictor estimate. The global maximum is then chosen from the set of local maxima resulting from this procedure. The grid spacing was chosen to be 0.4 times the smallest wavelength in the AM band. I determined empirically that this grid spacing is the largest that can be used while maintaining a low probability that local maxima are overlooked. The overall dimensions of the grid are determined from the error covariance matrix of the Kalman predictor. More details can be found in the source code.

3.2.2 Experimental Validation

In Figure 3.6 and Figure 3.7 I present results from an experiment in which positions are determined by maximizing an ambiguity function at each epoch, independent from all other epochs. The purpose of this experiment is to test my ability to resolve the ambiguity at each epoch. In the interest of simplicity in this preliminary experiment, I limited the distance between the base and rover to 50 meters. The experiment was conducted on 5 January 2002. The base receiver was placed at a DGPS surveyed location (WGS-84: N42.40860 W71.18851) in the driveway of a suburban house. The rover was placed on a teacart and rolled to three pre-surveyed waypoints in the street in front of the house. The three rover waypoints were surveyed, relative to the base station position, using a tape measure and astronomical observations. Figure 3.6 is a scatter plot of the results.
Figure 3.6: AM ambiguity function scatter plot.

The blue diamonds show the path of the rover as estimated by maximizing the ambiguity function. The clusters are evidence of the stops at each waypoint and differ from the survey by as much as five meters. Figure 3.7 illustrates the same data as a time plot.
Figure 3.7: AM ambiguity function positioning time plot.

The survey trace is constructed from the three waypoints. The figure clearly shows the receiver stopping at each waypoint and that the AM position estimates are within five meters of the survey. Figure 3.8 shows the individual cosine terms that make up the ambiguity function for a portion of this experiment. Individual cosine terms are computed for each transmitter $j$ using,

$$
\cos(\theta^j - \hat{\theta}^j(x^*, y^*, t^*))
$$

where $\theta^j$ and $\hat{\theta}^j$ are the measured and theoretical phase, respectively. The arguments of the theoretical phase $x^*$, $y^*$, and $t^*$ are the trial values of $x$, $y$, and $t$ that maximize the ambiguity function. In the absence of any errors, all cosines would be equal to one.
Figure 3.8: Individual cosine terms that make up the ambiguity function.

The legend in Figure 3.8 indicates the nominal frequency of the observed radio station in kHz. The distance equivalent of each individual cosine can be calculated using:

$$\frac{\arccos(cv)}{2\pi} \frac{c}{f}$$

where $cv$ is the cosine value from the chart, $c$ is the speed of light, and $f$ is the frequency. The plot shows that no cosine value was lower than 0.982. This cosine value, corresponding to the largest error represented in the chart, occurs for the radio station that has a nominal frequency of 1600 kHz. Using the above formula, the distance equivalent of this individual cosine term is about five meters.

### 3.2.3 Experimental Validation for a Longer Baseline

I conclude this chapter with the results of a longer baseline experiment conducted at Hanscom Air Force Base. The baselines in this experiment were longer than the others in this chapter, but still short enough to avoid the errors associated with the effects I will address in the following chapters. This experiment was conducted on 8 January 2001. The base receiver is placed at a DGPS-surveyed location (WGS-84: N42.45485 W71.26996) in a parking lot at the Air Force Research Laboratory. The rover is driven to four waypoints in the back of a truck. The receiver is removed from the truck at each epoch to ensure that the truck itself does not affect the positioning results. Results are shown for all times, not just those when the receiver is removed from the truck.
position of the truck is also tracked using our DGPS system. Figure 10 shows the results from the AM and DGPS tracking.

![Graph showing AM and GPS positioning results](image)

**Figure 3.9: AM ambiguity function positioning results and DGPS positioning results.**

The AM navigation system reports position estimates every 5 seconds according to the base station sample clock. An AM position estimate refers to the time (according to the base station sample clock) of the middle of the base station data burst. The GPS receiver adaptively controls the time at which it records position estimates in memory. Therefore, the GPS and AM epochs are not synchronous. Even so, the plot clearly shows that the tracking results agree quite well for longer baselines. Figure 3.10 shows the same AM data as a time plot. The GPS traces in this figure are constructed only from the four waypoints rather than from the tracking data.
At each GPS-surveyed waypoint, the rover was removed from the truck, placed directly over the waypoint (5-10 meters from the truck), and left undisturbed for several minutes. Figure 3.10 clearly shows five distinct intervals in which the position estimates are nearly constant. The first corresponds to the time in which the rover was motionless and quite close to the base station. The remaining four constant intervals correspond to the stops at the four waypoints. The receiver was only motionless in the middle of the constant intervals. The variation near the ends of the intervals is due to real motion of the receiver as I unloaded and loaded it. Since the discrepancy between the GPS and AM position estimates near the ends of the constant intervals can be completely explained by the (un)loading of the rover, the plot does not indicate much, if any, distortion caused by the truck itself.

The individual cosine terms that make up the ambiguity function for this experiment are similar to those from the previous experiment so a plot is omitted. Instead I include a plot of the ambiguity function for an epoch near the end of this experiment.
Figure 3.11: Plot of an ambiguity function over a one square kilometer area for the correct value of the receiver clock offset. The base receiver is at the origin and the rover is near -200 east, 200 north.

Figure 3.11 again shows a towering peak over the correct position of the roving receiver. The peak does indeed move appropriately as the roving receiver moves. Comparing Figure 3.3 with Figure 3.11 reveals no significant changes other than the shift in position away from the origin. The value of the peak over the correct position exceeds 0.9 while other local maxima are all less than 0.4. The figure provides excellent validation for the ambiguity function algorithm on longer baselines.

3.3 Concluding Remarks

In this chapter, I have described and demonstrated two algorithms for navigating using AM broadcast signals. The incremental navigation algorithm works but is inferior because it requires both initialization and continuous tracking. I have shown that robust instantaneous positioning is possible on baselines up to 280 meters using an ambiguity function. The ambiguity function method is superior to incremental navigation in all respects except for computation time. Since computing is cheap these days, the ambiguity function algorithm will be used exclusively for the remainder of this thesis.
Chapter 4: AM Navigation Challenges

In this chapter I explore some issues that present impediments to using AM broadcast signals for navigation. Fortunately, navigation errors caused by some of these impediments can be minimized using models of the relevant physical effects. Where this is possible I document the model development that I did, and evaluate its effectiveness at reducing navigation errors. Where modeling is not practical, or at least was not done by me, I simply characterize the errors and evaluate implications for navigation performance.

4.1 Transmitter Antenna Position

As we saw in the previous chapter, AM navigation requires a database of radio station antenna locations. Position errors in this database lead directly to errors in the calculation of theoretical phase. For GPS, the corresponding problem is satellite ephemeris error. AM navigation is more sensitive than DGPS to transmitter position errors because of the close proximity of the transmitters.

4.1.1 Effect on Navigation

Recall that AM navigation uses observations of phase differences. So, navigation is affected only if one receiver is affected differently than the other. To quantify the error in the theoretical phase difference caused by an incorrect antenna position, I define three vectors. Let $a$ be the vector from the base receiver to the actual transmitter antenna position. Let $b$, the baseline vector, be the vector from the base receiver to the roving receiver. Let $e$ be the vector from the actual transmitter antenna position to the incorrectly assumed transmitter antenna position. To first order in $\|e\|$, the error in the theoretical phase difference is:

$$\Delta \hat{\theta} = \left( \frac{(b - a)^T}{\|b - a\|} + \frac{a^T}{\|a\|} \right) e$$

We can make a number of useful observations about this equation. As expected, to first order, the theoretical phase error scales linearly with the transmitter antenna position.
error. Also, when the distance to the transmitter antenna is much larger than the baseline the equation reduces to:

\[ \Delta \hat{\theta} = \frac{b^T}{\|a\|} e \]

This approximation holds for most DGPS work because the distances to the satellites are so great. It also holds for AM navigation on short baselines. However, for longer baselines the more general equation must be used.

Consider the scenario in which both the base receiver and rover are on the line defined by the error in antenna position. This scenario can be either best-case or worst-case. If the base and rover are on the same side of the antenna then the error in the theoretical phase is zero. However, if they are on opposite sides of the antenna then the error is twice the error in antenna position. It should be noted that this worst-case scenario is not all that far-fetched for AM navigation. For baselines over 10km it is common for the rover to be on the other side of a transmitter antenna from the base receiver.

The FCC requires AM transmitter antennas to be surveyed to the nearest arcsecond [11] and the FCC database only reports positions to the nearest arcsecond [10]. So, lack of numerical precision alone can result in antenna position errors up to about 15 meters. As discussed above, the resulting error in theoretical phase can be as large as 30 meters.

Errors in theoretical phase as large as 30 meters are clearly not acceptable for a system that we have already seen can produce meter-level results. Better transmitter antenna surveys are needed.

### 4.1.2 Minimizing Effect on Navigation

To satisfy this need I surveyed the antenna towers of 29 AM radio stations in eastern Massachusetts and southern New Hampshire. From the above analysis it is clear that a meter-level positioning system needs meter-level surveys of at least the transmitter antennas that are close to the receivers. I used a pair of Ashtech Z-12 GPS receivers and a theodolite to do geodetic surveys to determine the positions of the AM transmitter towers in the WGS-84 coordinate system. I placed one of the GPS receivers on a post on the roof of Lincoln Laboratory Lexington, MA. The WGS-84 coordinates of the post (N42.4594972, W71.2651564) had been determined within about five centimeters. I drove to each of the 29 stations with the other GPS receiver and the theodolite.

For each radio station I performed the following steps:

1. Set base GPS receiver to collect data.
2. Drove to radio station with roving GPS receiver and theodolite.
3. Set up and leveled two tripods with the general goal of getting them close to the AM transmitting antennas, far from each other, and placed such that the lines through them and any particular transmitting antenna are close to orthogonal.
4. Placed the GPS antenna on one tripod and set the receiver to collect data.
5. Placed the theodolite on the other tripod and measured azimuth angles in the locally horizontal plane to the center of each AM antenna tower and to the GPS antenna on the other tripod.
6. Swapped the location of the GPS receiver and the theodolite and repeated the measurements.
7. Repeated steps 2 through 6 for other radio stations.
8. Downloaded the GPS data from both receivers.
9. Post-processed the GPS data to get tripod positions.
10. Used angle measurements and tripod positions to compute antenna tower positions.

The steps are straightforward and I encountered no significant problems conducting the surveys. The antenna position calculation is also straightforward. A theodolite by itself is only capable of measuring relative bearings. However, since I measured an angle from one GPS-surveyed location to another (the tripods), the relative bearings can easily be converted to true bearings. True bearings from a surveyed location define a line. The lines from each tripod intersect at the AM transmitter antenna positions. I wrote a program to perform these calculations.

As a check, I collected three complete sets of angle measurements for each AM tower. The corresponding measurements from these sets never differed by more than 15 arcseconds and the distance from the towers to the tripods never exceeded 300 meters. Since the tripods were purposely placed in orthogonal directions from the towers, the antenna positioning error due to angle inaccuracy was less than five centimeters. Furthermore, I used each of the angle measurement sets independently to determine the positions of the towers relative to the tripods. In all cases, the resulting tower position for each measurement set differed by no more than 15 centimeters.

I used Ashtech software to post-process the GPS data. It calculates not only position estimates but also uncertainty estimates. In all cases the two-sigma error was estimated to be less than 50cm and as expected the worst estimated errors were on the longest baselines. From this and the above analysis, I believe that the surveys produced sub-meter AM tower positions for all the stations I visited. The results of this survey can be found in Appendix A.

### 4.1.3 FCC Position Coordinate Errors

Figure 4.1 shows the RSS of differences between the north and east coordinates from the FCC database and those from my surveys for each AM station.
Most of the stations have position errors that are at least close to the 15-meter numerical precision limit. There are a few exceptions. In defense of the FCC and those reporting information to the FCC, some of the position coordinate errors may be due to a misunderstanding about what is to be surveyed. The FCC only requires geodetic position coordinates for the antenna array, not for individual towers. Stations with multiple tower antenna arrays are required to report geodetic positions for the center of the array and the relative positions of the antennas in the array. The relative antenna positions are relative to each other, not necessarily relative to the geodetic station coordinates [11]. To me the “center of the array” means the geometric center. I suspect some station engineers interpret this differently, perhaps assuming that the tower nearest to the geometric center is the center of the array. Whether the errors in Figure 4.1 are caused by this ambiguity or by imprecise surveys is really irrelevant for our purposes. The errors are significant and the survey was clearly necessary.

### 4.1.4 Alternative Solutions

Surveying the AM transmitter antennas using DGPS and a theodolite was not the only option. It is entirely possible to compute the transmitter antenna positions from geographically diverse observations of AM carrier-phases if the receiver positions are known. This technique has been used to determine GPS satellite orbits using observations of the GPS signal from known locations [6]. In fact, this is precisely how the GPS ephemerides broadcasts are calculated. These techniques are almost certainly
adaptable for use with AM signals. However, the use of these techniques is not appropriate in the context of this thesis because the propagation of AM signals is not yet clearly understood. In fact, one of the goals of this thesis is to examine propagation issues. Surveying the antennas using independent means helps isolate errors and ensures better understanding of propagation.

It is also possible to determine the relative excitation of the towers in each array from a suitably distributed set of AM phase observations from known locations. I did not explore this technique either, but in the following section I analyze multiple-tower antennas and suggest a model to use in navigation.

### 4.2 Transmitter Antenna Pattern

Some AM transmitters have just a vertical monopole antenna but most have multiple-tower antennas. Multiple towers are used to make the antenna gain pattern directional. Usually the antenna pattern includes one or more nulls that are engineered to protect a distant station from interference. For the navigator, an undesired side effect of the directional antenna is phase that varies with azimuth angle.

#### 4.2.1 WRKO Simulation

To understand how a transmitter antenna pattern affects phase measurements I modeled WRKO’s antenna using NEC-4 [4]. WRKO is in Burlington, MA and operates on 680 kHz. WRKO’s three-tower antenna is clearly visible to the northwest as one drives by the Middlesex Turnpike on Route 128. NEC-4 (Numerical Electromagnetics Code) is a widely used method-of-moments electromagnetics-modeling program developed by Lawrence Livermore National Laboratory and others.

The FCC provides relative tower positions, heights, and excitation information on its website [10]. One is left to speculate on the geometry of ground radials that are almost always used in AM transmitting antennas to improve efficiency. Using the FCC information and an educated guess on ground radial geometry, a model of WRKO’s antenna was constructed in NEC-4 as shown in Figure 4.2.
Chapter 4: AM Navigation Challenges

Figure 4.2: Model of WRKO’s three-element antenna.

I included 8 quarter-wavelength ground radials for each tower. Table 4.1 dimensions the rest of the drawing and gives the electrical excitation of each tower for the daytime pattern.

<table>
<thead>
<tr>
<th>Tower</th>
<th>Bearing (true degrees)</th>
<th>Spacing (electrical degrees)</th>
<th>Height (electrical degrees)</th>
<th>Field</th>
<th>Phase (degrees)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW tower</td>
<td>0</td>
<td>0</td>
<td>108</td>
<td>0.5</td>
<td>104</td>
</tr>
<tr>
<td>Middle tower</td>
<td>75</td>
<td>155</td>
<td>97</td>
<td>0.9</td>
<td>40</td>
</tr>
<tr>
<td>NE tower</td>
<td>75</td>
<td>310</td>
<td>108</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.1: WRKO geometry and daytime electrical parameters.

As the table shows, the towers are collinear and equally spaced. “Field” indicates the relative amplitude of the current in each tower. Unfortunately, the FCC does not give the height where the field and phase are specified. I assume that the field and phase are specified at the height of the current maximum in each tower since the current here most closely relates to far-field radiation pattern. WRKO is a 50 kW station, although its power is irrelevant here.

NEC-4 allows modeling of the ground as a lossy dielectric half-space using a Sommerfeld ground model. The model allows specification of ground permittivity and conductivity. For my model of WRKO, I chose values of 5 and 0.002 mhos/meter for relative permittivity and conductivity, respectively. These values are typical for New England ground.

When modeling or operating a multiple-element antenna, getting the desired current distribution is not just a matter of driving the antenna elements at their bases with currents of the desired amplitude and phase. Mutual coupling among the antenna elements makes life more complicated. AM stations typically use loops part way up the
antenna towers to sample the amplitude and phase of the currents at those points in the antenna. The amplitudes and phases of the base currents can be tweaked to get the desired currents at the loops. This is an iterative process because tweaking the input to one tower in a multiple-tower antenna changes the current distributions in the other towers.

The development of my model of WRKO was likewise iterative. I first modeled the antenna using base currents that match the fields and phases in Table 4.1. I then found the height where the maximum amplitude of the current distribution occurs on each tower. I tweaked the input currents so that the amplitude and phase at these heights match those in Table 4.1. Only two iterations were needed to achieve good convergence.

I asked NEC to calculate the field for five concentric circles centered on the center tower. The radii of the circles varied from 500 meters to 10 kilometers. The phase of the z component of the electric field as a function of azimuth on each circle is plotted in Figure 4.3. I shifted the curves so that the phases were equal for zero azimuth and I manually removed any 360-degree phase wraps.

![Figure 4.3: Simulated phase vs. azimuth angle for WRKO for various radii circles centered on WRKO’s center tower.](image)

Figure 4.3 shows dramatic phase variation as the azimuth angle changes. The same plot for a monopole antenna would be flat for all radii. The dramatic phase variation implies the potential for large differential phase errors. In fact, if the base and rover were off opposite ends of the array (75 and 255 degrees azimuth), a phase difference error of over 200 degrees (245 meters) would result. An AM navigation system clearly needs to account for this effect.
4.2.2 Antenna Pattern Model

Using NEC simulation results directly to model antenna patterns presents problems. First, it would be tedious to carefully model every multiple tower antenna. Twenty of the twenty-nine stations that I use for navigation in the Boston area have multiple-tower antennas and most of these have different day and night antenna patterns. Therefore, close to 40 simulations would be required. Using the data from the simulations would also be challenging. Fitting curves to the data in Figure 4.3 would require a high-order curve fit because of the complex shape. Also, many curves would be needed for each antenna pattern to account for all radii. None of these obstacles are show stoppers but they provide good motivation to seek an alternative.

One way to compute the phase of the field produced by a multiple-tower antenna is to consider the fields caused by the individual towers. The complex amplitude of the field at a distant point is the complex sum of the field caused by each tower at that point. Specifically,

\[ E_z = \sum_j E_z^j \]

where \( E_z^j \) is calculated for each tower as if it were the only tower. By expressing the field this way, I have reduced the problem to calculating the field due to a monopole. The field produced by a monopole does not vary with azimuth but does depend on ground properties. I could simulate how the field varies as a function of distance from the monopole. This is a much easier simulation task than the one described above because the field only varies with distance from the tower. Furthermore, the specifics of the antenna arrays need not be considered and the tweaking procedure I describe above is not necessary. Simulations of this nature will be explored in the next section. For the purpose of calculating the antenna pattern, however, the effect of varying ground properties should not make that much difference. Therefore, I will model antenna pattern as if the antenna were over perfectly conducting ground.

Over perfect ground, the amplitude of the field falls off as one over distance and phase decreases by the distance times the free-space wavenumber.

\[ E_z' = m \frac{f^j h^j}{d^j} \angle (\phi^j - kd^j) \]

where \( m \) is a scale factor; \( d \) is the distance to the antenna; \( f, \phi, \) and \( h \) are field, phase and height from Table 4.1; and \( k \) is the free-space wavenumber. Since \( m \) is constant for all \( j \), and we are only interested in phase, \( m \) is ignored.

I simply model the antenna array phase variation as the angle of \( E_z \). In Figure 4.4, I compare my model with the NEC simulation results for WRKO. I took the model phase estimate and subtracted the NEC simulated phase for different distances from the center tower.
4.2 Transmitter Antenna Pattern

The model does quite well despite not considering the imperfect ground. For distances greater than one kilometer, the maximum phase difference error that can occur due to antenna pattern mis-modeling is less than 20 degrees (25 meters). This is a factor-of-ten improvement over no model. The 0.5-kilometer trace in Figure 4.4 shoots off the chart and up to 37 meters at 195 and 315 degrees azimuth. As we will see in the next section, the effect of the ground increases near the antenna. This fact explains the generally improving performance of the model as distance from the antenna increases.

Figure 4.4 suggests a navigation algorithm rule: don’t use measurements from stations that are within five kilometers of any receiver. This rule limits the worst-case phase difference error to ten degrees. Because AM stations are typically well distributed geographically, a very small number of measurements are disregarded as a result of this rule.

4.2.3 Model Sensitivity

The FCC requires radio stations with multiple tower antennas to maintain relative field strength to within five percent of nominal and relative phase to be within three degrees of nominal [18]. To test the effect of an error of this magnitude, I compared the outputs of my antenna pattern model using correct parameters and parameters that were slightly skewed: I increased the amplitude of the SW tower by five percent and decreased the phase of the SW tower by 3 degrees. I compare modeled phase estimates for these sets of parameters in Figure 4.5.
Chapter 4: AM Navigation Challenges

Figure 4.5: WRKO phase modeling error caused by incorrect model parameters. The amplitude and phase of the SW tower are increased five percent and decreased by three degrees, respectively.

Figure 4.5 shows that even if a radio station is operated within specifications, significant phase error can result. The phase error caused by this effect can be minimized by adherence to the rule suggested above: don’t use measurements from stations that are within five kilometers of any receiver. The results in Figure 4.5 validate not only the rule, but they also validate the use of the antenna pattern model. The modeling errors in Figure 4.4 are similar to those in Figure 4.5. So even if a better model is used, similar errors remain due to circumstances beyond my control.

I also investigated phase error caused by varying field parameters on other towers and on multiple towers simultaneously. The results are omitted because the results in Figure 4.5 are representative.

4.2.4 Experimental Results

Finally, I compare experimental results using the antenna pattern model with the same experimental results not using the model. The experiment was conducted on 18 July 2001. I set up the base receiver at a DGPS-surveyed location (WGS-84: N42.45644 W71.26673) in MIT Lincoln Laboratory’s Annex 6 parking lot. MIT Lincoln Laboratory is located in Lexington, MA within the confines of Hanscom Air Force Base. Annex 6 is near the top of Katahdin hill. The base receiver was placed well away from buildings and overhead power lines. I drove the rover out Hartwell Ave., north on Route 128 (I-95), north on I-93, and finally south on I-495. I include results from only the first 30 minutes
of this experiment because shortly after this time the ambiguity function method fails when the antenna pattern model is not used. At 30 minutes, the rover is still heading north on I-93, about two kilometers south of I-495.

The yellow line in Figure 4.6 shows the path traversed by the roving receiver during the experiment. The positions of the base receiver and that of WRKO are also shown marked “Base” and “680”, respectively. The waypoints marked “12” and “15” on the yellow line mark the location of the rover at the elapsed time in minutes indicated by the number. The significance of these elapsed times will become apparent shortly. The base receiver is about five kilometers from WRKO bearing 225 degrees true. The rover starts near the base receiver and ends up over 19 kilometers from WRKO bearing 300 degrees true. Since the rover passes through a bearing of 195 degrees true from WRKO, we should see the results of the large phase discontinuity predicted by the simulation and shown in Figure 4.3.
Chapter 4: AM Navigation Challenges

In Figure 4.7, I show post-fit phase residuals for the experiment when the antenna pattern model is not used. A post-fit phase residual is the difference between the actual measured phase difference and the theoretical phase difference computed at the position that maximizes the ambiguity function. The phase residual is computed for every station whether or not it is actually used when maximizing the ambiguity function. The phase residuals are scaled by the appropriate inverse wavenumber so that the quantity is expressed in equivalent meters. Only the results from the first 30 minutes of this experiment are included here because without the use of an antenna-pattern model, ambiguity resolution failed for the longer baselines encountered during the experiment.

As expected, a number of the stations have significant phase residuals when the rover is far from the base. WRKO, represented by the turquoise X’s, is the most serious offender. At about 12 minutes the phase residual for WRKO increases dramatically and at 15 minutes it reaches its maximum. As mentioned before, the “12” and “15” on the map indicate the position of the rover at the indicated elapsed time. The bearings from WRKO to the 12 and 15-minute marks are approximately 195 and 90 degrees true, respectively. The discontinuity occurred exactly where one would predict from the NEC simulation results. The maximum, however, occurred a little south of where one would predict. Also, using the simulation results, the maximum error should be about 225 degrees or 275 meters. Figure 4.7 shows that the maximum phase residual is actually about 350 meters. At the 15-minute mark, a number of stations have large phase residuals so the resulting position estimation error could be quite large. Since the position estimate is used to calculate post-fit residuals, position error affects the residuals. The only way to remedy this problem is to obtain better position estimates.
4.2 Transmitter Antenna Pattern

accomplish this, I apply the antenna pattern model. The resulting phase residuals are shown in Figure 4.8.

![Figure 4.8: Post-fit phase residuals when antenna pattern model is used.](image)

Using the antenna pattern model dramatically reduces the post-fit phase residuals. A few large residuals persist. There are a number of single epochs that have large residuals on a large number of stations. These errors are almost always attributable to overhead power lines. This effect will be investigated in detail in a subsequent section.

The phase residual for WBZ (1030) exceeds 100 meters. The main WBZ transmitter is near Bumpkin Island in Boston Harbor, but WBZ also has an auxiliary transmitter in Brighton, MA. I suspect the large phase residuals that occur on WBZ between three and eight minutes are due to testing of this auxiliary transmitter. A step in signal strength, observed by both the base and the rover, provides evidence for this theory.

The phase residual for WRKO (680) still gets nearly as high as 100 meters despite the use of the model. The residual reaches these high values between 12 and 16 minutes, which corresponds to the time when the rover is close to the transmitter. Figure 4.4 indicates poor model performance when a receiver is close to the transmitter, but the magnitude of the error far exceeds that predicted. Furthermore, the residuals for the other stations are quite small so position estimation error is not the likely culprit. The large error is most likely caused by a combination of mis-modeling and incorrect amplitude and phase information. I will attack the mis-modeling part of this error in the next section. The incorrect amplitude and phase information could be caused by errors in the FCC database or by WRKO not controlling the array to specifications.
Finally, WEZE (590) has phase residuals that exceed 70 meters. Because WEZE is over ten kilometers from the nearest approach of the rover and over fifteen kilometers from the base (see Figure 4.6), the mis-modeling component of this error is likely quite small. The amplitude and phase information I got from the FCC’s website for WEZE’s directional antenna are likely incorrect.

4.3 Imperfect Ground

In this section I explore the effect of imperfect ground on navigation performance and document the development of a model to account for this effect. The model I developed treats the ground in the entire Boston area as a homogeneous medium. Much of this section is devoted to documenting the selection of electrical parameters for this medium.

I start by presenting results from some rather generic NEC-4 simulations. The goal of these simulations is to motivate model development and to quantify the effect of imperfect ground on phase at both ends of the AM band.

I then proceed to show the post-fit phase residuals from a real experiment in which a ground model is not used. I select a single station from this experiment and compare phase simulation results to post-fit phase residuals for a specific AM radio station. The goal of this comparison is to show general agreement between simulation and reality. More importantly, using this comparison I argue that post-fit phase residuals are subject to noise and systematic errors that make them largely unsuitable for ground parameter estimation.

I show that comparing simulations and observations of amplitude, as opposed to phases, is quite useful and appropriate for selecting ground parameters. I demonstrate good consistency for several widely-spaced, in geometry and frequency, radio stations.

Finally, I report model parameters derived from the ground parameters that were deduced from observing amplitude. I then show the post-fit phase residuals from the same experiment mentioned above, but using a ground model.

4.3.1 Baseline NEC-4 Simulations

I first conduct some NEC simulations to quantify the effects of imperfect ground and to motivate model development. NEC-4 allows simulation of the ground using a Sommerfeld model. The model allows specification of the relative permittivity and the conductivity of the ground. The surface of the earth is, of course, not homogeneous. At one extreme is saltwater with very high conductivity and permittivity. At the other extreme is very poor ground that one would expect to find in mountainous regions. I simulated these two extremes and three types of ground that fall in between. Table 4.2 shows the five types of ground that I simulated along with specific values of relative permittivity and conductivity.
Table 4.2: Table of permittivity and conductivity for NEC simulations.

<table>
<thead>
<tr>
<th></th>
<th>Permittivity</th>
<th>Conductivity (mhos/meter)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saltwater</td>
<td>80</td>
<td>5</td>
</tr>
<tr>
<td>Freshwater</td>
<td>80</td>
<td>0.001</td>
</tr>
<tr>
<td>Good ground</td>
<td>13</td>
<td>0.005</td>
</tr>
<tr>
<td>New England ground</td>
<td>5</td>
<td>0.002</td>
</tr>
<tr>
<td>Very poor ground</td>
<td>3</td>
<td>0.001</td>
</tr>
</tbody>
</table>

The electrical properties of water and soil vary with their chemical and physical composition. The conductivity of saltwater, for instance, varies greatly with the salinity of the water. The Dead Sea has better conductivity than the brackish waters of Lake Pontchartrain. The parameters for saltwater in Table 4.2 are typical of ocean water. The conductivity of freshwater is likewise affected by the amount of dissolved solids it contains. The parameters for freshwater in Table 4.2 are meant to be representative of freshwater in New England that has very little dissolved minerals. The New England ground parameters are taken from an FCC chart [12] for the area. The good ground and very poor ground are meant to represent ground that might be encountered in midwest farmland and mountainous regions, respectively. There is actually very little of either in the Boston area but I include these simulations for comparison.

In the simulations, I model the transmitting antenna as a half-wavelength monopole with eight quarter-wavelength ground radials. To ensure that the details of the transmitting antenna do not affect the simulation results, I also simulate perfect ground to serve as a baseline. I subtract the phase predicted in the perfect-ground simulation from the phase predicted in each of the imperfect-ground simulations.

I simulate phase over perfect ground and the grounds described in Table 4.3 at both ends of the AM band: 550 kHz and 1700 kHz. The results are shown in Figure 4.9.
The phase difference is expressed in equivalent meters so that the results are easily related to navigation performance and so that the results from the different frequencies can be compared on the same axes. The saltwater traces for both 550 and 1700 are identical.

The results in Figure 4.9 are strange at first because it is not obvious that the phase velocity should vary with distance from the transmitter. This is especially troubling since we explicitly eliminated any transmitter antenna effects by subtracting the phase over perfect ground. Only the linear saltwater traces indicate a constant phase velocity for any distance from the transmitter. All the other traces indicate a much slower phase velocity near the transmitter antenna.

Understanding the shapes of the curves in Figure 4.9 requires a better understanding of groundwave propagation. A groundwave can be described as the superposition of three separate waves: one that travels in the air, one that travels on the surface, and one that travels in the ground. The wave that travels in the air travels with a phase velocity that is near the speed of light in a vacuum. The surface wave component is small and can be neglected. The wave that travels in the ground travels with a phase velocity that varies with ground properties but is always slower than the speed of light. Because the ground is lossy, the groundwave component that travels in the ground dissipates much faster than the wave that travels in the air [28]. The faster dissipation of the wave that travels in the ground explains the results in Figure 4.9. The phase velocity is slow near the transmitter where the wave that travels in the ground is still strong. As one moves away from the...
transmitter, the phase velocity increases to near the speed of light in a vacuum because
the wave that travels in the ground dies out leaving only the wave that travels in the air.

At lower frequencies, the ground is less lossy so the groundwave component that travels
in the ground dissipates more slowly. The result of this effect can be clearly seen in
Figure 4.9. The 1700-kHz traces become nearly straight lines after the distance from the
transmitter exceeds only 20 kilometers. Most of the 550-kHz traces, however, never
quite become straight lines, indicating the continuing influence of the wave that travels in
the ground.

When considering the navigation error that the results in Figure 4.9 might imply, it is
important to remember that the AM navigation system that I have developed is a
differential system. So even if the propagation effect is ignored, the errors can be quite
minimal if both the base and rover are kept far from the transmitters. This is certainly
ture for frequencies near the upper end of the AM band with very reasonable restrictions
on transmitter proximity. However, in the lower-frequency end of the band, the
necessary restrictions on transmitter proximity become troublesome. A ground model is
clearly needed.

4.3.2 Baseline Experimental Results

Now I turn to experimental results that illustrate the effects of imperfect ground. I
conducted an experiment on 19 January 2002 that involves the close proximity of
transmitters and receivers. The base station was placed at a GPS-surveyed location
(WGS-84: N42.41687 W71.19398) in a parking lot near the intersection of Dow Avenue
and Route 2 in Belmont, MA. A map of receiver and transmitter positions during the
experiment, which I will call experiment one, is in Figure 4.10.

Figure 4.10: Map of receiver and transmitter positions during experiment one, conducted on
19 January 2002. This experiment is used to develop a ground model.
Chapter 4: AM Navigation Challenges

The yellow line in Figure 4.10 shows the path of the roving receiver along Route 2. The rover actually traversed this route twice during the experiment. The location of the base receiver, marked with “Base”, can be seen right next to the path of the rover. The base receiver was placed in an open parking lot.

I show the post-fit phase residuals from this experiment in Figure 4.11. I used the antenna pattern model, described above, for this experiment.

![Figure 4.11: Post-fit phase residuals from experiment one.](image)

Figure 4.11 shows that 590, 680, 740, 1150, 1360, and 1510 all have significant systematic errors. Not surprisingly, the stations with the largest errors have low frequencies, close proximity to the receivers, or both. 1360 is off the map in Figure 4.10, approximately 15km to the northeast of 740. Although the phase residuals are plotted for the stations with the largest errors, the measurements from these stations were not considered when computing the position estimates. Therefore, the post-fit phase residuals are not corrupted by the larger measurement errors.

Station 740 has the largest post-fit phase residuals in Figure 4.10 and it also has a single-tower antenna. Since 740 is immune to antenna modeling errors, it is an ideal candidate for investigating propagation errors.

I conducted some NEC simulations, similar to those described above, for 740. I simulated various grounds likely encountered in the Boston area. The specific parameters for relative permittivity and conductivity are in Table 4.3.
Table 4.3: Relative permittivity and conductivity for 740 kHz simulations.

The values were chosen around those published by the FCC for the Boston area [12]. Interestingly, the FCC actually only gives the conductivity in [9]. In a much earlier publication, they roughly give the relative permittivity for cities to be 5 [13]. The FCC’s recent lack of concern for specifying the permittivity is well justified. Although Figure 4.9 clearly shows that the phase depends on both the permittivity and conductivity, it turns out that the amplitude is mostly insensitive to permittivity. Since the FCC is primarily concerned with keeping stations from interfering with each other, they are only interested in how the ground affects the amplitude.

I show the simulation results in Figure 4.12. I also include the post-fit phase residuals for 740 (previously shown in Figure 4.11) in the figure. This time the phase residual is not plotted versus time, but versus distance from the 740 transmitter.

For each ground simulated, the NEC-simulated phase is shifted so that the phase is equal to zero at the distance the base receiver was from the 740 transmitter. This accounts for...
the differential nature of the post-fit phase residuals and allows for easy comparison with the experimental results.

The good news from Figure 4.12 is that the phase is not that sensitive to likely ground variation in the Boston area for this frequency. The bad news is that none of the simulation results agree that well with the measured results. Near the transmitter, a conductivity of 0.004 and a relative permittivity of 5 fit the data best. Far from the transmitter, 0.002 and 2.5 are better parameters. Realistically though, the data are inconclusive.

I looked at similar data from other experiments for a variety of radio stations. The results were similarly inconclusive. The problem is that post-fit phase residuals are almost always noisy and have systematic errors that are related to inaccurate position estimation. There is a chicken-and-egg problem. A ground model is needed to get more accurate position estimates (necessary for accurate post-fit phase residuals). But accurate post-fit phase residuals for longer baselines are necessary to develop the model. I could proceed using a bootstrapping approach but fortunately there is an alternative. Amplitude is much more sensitive than phase to variation in ground conductivity.

4.3.3 Conductivity Estimation Using Amplitude

I conducted a number of additional experiments and NEC-4 simulations to determine an appropriate value for Boston-area ground conductivity by looking at amplitude. We are not particularly interested in the amplitude, but it is much more sensitive to ground conductivity than phase.

I begin by presenting simulated and real amplitude data for WJIB, 740 kHz. The simulated data are from the NEC simulations with the ground parameters described in Table 4.3. In the NEC simulations, I model the antenna as a half-wavelength monopole with eight quarter-wavelength ground radials. This geometry does not represent the geometry of WJIB’s antenna. I also used an arbitrary-valued current source to excite the antenna in NEC. I explain my reasons for these actions below. The experimental amplitude data are from experiment one, described above. I also include amplitude results from another experiment, conducted on 22 January 2002, that includes rover positions that are significantly farther from the 740 transmitter. I show the relevant geometry for this experiment, which I will call experiment two, in the following map.
4.3 Imperfect Ground

Figure 4.13: Map of receiver and transmitter positions during experiment two, conducted on 22 January 2002. This experiment is used to develop a ground model.

The rover gets as close as one kilometer and almost as far as 50 kilometers from 740 in this experiment. The base receiver, labeled “Base” on the map, is placed at a GPS-surveyed location (WGS-84: N42.45450 W71.27003) in the Air Force Research Laboratory parking lot, which is located on Hanscom Air Force Base in Bedford, MA. The parking lot is large, flat, and free from overhead conductors. NEC-simulated amplitude and the amplitude observed by the rover for experiments one and two are shown in Figure 4.14.
Figure 4.14: NEC-simulated amplitude and the amplitude reported by the rover in experiments one and two. The frequency is 740 kHz. Various grounds simulated.

In experiments one and two, like most experiments, I periodically adjusted the roving receiver’s gain to ensure full utilization of the A/D converter’s dynamic range. I adjusted the experimental results to account for the gain changes. This was an easy and exact task since I had recorded when I changed the gain. Also, since the gain is adjusted in powers of two, the gain changes are quite obvious in the data: they appear as 6db jumps in amplitude for all stations. Otherwise, the experimental data are unaltered.

As I mentioned above, for the NEC simulations, I did not attempt to make the antenna excitation and antenna structure geometry resemble the real excitation and real geometry of station 740. Since 740 has a monopole for an antenna, the only difference between the simulations that I did and simulations that include a careful model of the antenna and excitation is a scale factor. Since the absolute gain of my receivers is not known a priori, an unknown scale factor exists anyway. Therefore, no additional unknowns are introduced by not carefully modeling the antenna and excitation. The antenna and excitation are, of course, constant for all simulations.

To account for the factor that separates the NEC simulations from my experimental results, the NEC curves in Figure 4.14 are shifted, all together, to align them with the experimental amplitudes. This alignment is rather straightforward near the antenna where all the simulations have similar results. However, my solution to accounting for this factor is a bit dubious so I refrain from making conclusions only based on the data in Figure 4.14. Rather, I defer until presenting similar results from other experiments and other radio stations.
WNTN (1550 kHz) is another station with a monopole for an antenna. Amplitudes from this station are especially interesting because the frequency is at the opposite end of the AM band from 740. 1550 can be seen near the bottom of the map in Figure 4.13, near the intersection of Interstates 90 and 95 (Route 128), almost due south of the base receiver. The rover gets as far as 40 kilometers and only as close as seven kilometers in experiment two. To supplement the data from experiment two, I include results from another experiment, conducted on 25 January 2002, in which the rover gets as close as 2 kilometers and as far as 20 kilometers. A map of this experiment, which I will refer to as experiment three, is in Figure 4.15.

![Figure 4.15: Map of receiver and transmitter positions during experiment two, conducted on 25 January 2002. This experiment is used to develop a ground model.](image)

The base receiver, marked with “Base” in Figure 4.15, is again located at a GPS-surveyed location (WGS-84: N42.45443 W71.26997) in the AFRL parking lot. The rover travels down I-95 (Route 128), west on I-90 (the Mass Pike), and then back east on some secondary roads.

I simulated 1550 kHz over the same grounds described in Table 4.3. I kept the excitation the same as in the 740 kHz experiments but I scaled the height of the monopole so that it is still a half wavelength tall. Therefore, simulations over perfect ground at 740 kHz and 1550 kHz produce identical results.

The 1550-kHz results from experiments two and three, the NEC simulation results, and results from experiment two are shown in Figure 4.16.
Again, the experimental results are adjusted to account for gain changes during the experiments. The NEC-simulated amplitude curves are also shifted as before, but they are shifted by exactly the same amount as they were for 740 except that I account for the difference in transmitter power. I do not account for transmitter power directly in the NEC simulations. Instead, I account for differences in transmitter power by scaling the NEC amplitude output. 740 is a 250-watt station and 1550 is a 10-kilowatt station. Therefore, 1550 has 40 times the power or about 6.3 times the voltage of 740. I added 16db to the 1550 NEC curves in addition to what I added to the 740 NEC curves to account for the difference in transmitter power.

The 1550 results are astonishingly consistent with the 740 results. Although the amplitude measurements are quite noisy, the results for both frequencies suggest a ground conductivity between 0.001 and 0.002 mhos/meter. Also evident from the results for both frequencies is that relative permittivity has very little affect on amplitude: the 2.5, 5 and 10 relative permittivity curves are nearly on top of each other for both frequencies.

Finally, I present amplitude results for WCCM (800 kHz). WCCM’s transmitter is located near the intersection of I-93 and I-495. It can be seen near the top of the maps in Figure 4.6 and Figure 4.13. Results for station 800 are interesting because the antenna is over 30 kilometers from both 740 and 1550 and therefore can give insight into ground parameter variation that may occur in the Boston area. The results are shown in Figure 4.17.
4.3 Imperfect Ground

Again, I compensate for the gain changes in the experimental results and I shift the NEC curves by the same amount they were shifted for 740 except that I account for the difference in transmitter power. I omit the NEC simulations for the different values of permittivity because previous results show that amplitude is not sensitive to permittivity.

Figure 4.17 reveals a slight inconsistency between the two experiments. The results from the antenna pattern experiment seem to favor a ground conductivity that is closer to 0.001 mhos/meter than the other results. This could be due to seasonal ground variation. The antenna pattern experiment was conducted in July while experiments one, two, and three were all conducted in January. Since the ground is generally drier in July than in January, the lower conductivity could be the result of less ground water. More data, however, is necessary to confirm this.

The results from four different experiments and three different radio stations are all astonishingly consistent, especially considering the large area involved. All results suggest a 0.0015 mhos/meter ground conductivity for the Boston area. The consistency of the results suggests that a homogeneous ground model could be quite effective.

4.3.4 Permittivity Estimation

Now that I have a good estimate of the ground conductivity in the Boston area, I turn to estimating the permittivity of the ground. Figure 4.14 and Figure 4.16 show that the
amplitude is not very sensitive to changes in permittivity. Therefore, amplitude observations and simulations will not help resolve permittivity.

Figure 4.9 shows that the phase is sensitive to large changes in permittivity. Referring to Table 4.2, freshwater and poor ground differ only in their permittivity: 80 and 3, respectively. The difference between the freshwater phase curve and the poor ground phase curve is quite dramatic. A majority of the ground in the Boston area, however, does not have such a large variation.

So the question is whether or not phase is sensitive enough to the subtle variation in permittivity that represents the possible permittivity in the Boston area. To answer this question, I include the results of more NEC simulations.

I simulated the phase at 550 kHz and 1700 kHz for the following ground parameters.

<table>
<thead>
<tr>
<th>Relative Permittivity</th>
<th>Conductivity (mhos/meter)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.0015</td>
</tr>
<tr>
<td>2.5</td>
<td>0.0015</td>
</tr>
<tr>
<td>10</td>
<td>0.0015</td>
</tr>
</tbody>
</table>

Table 4.4: Ground parameters for NEC simulations designed to study the effect of permittivity variation.

The results of these simulations are plotted below.

Figure 4.18: Difference between NEC-4 simulated phase for grounds with varying permittivity and perfect ground for both ends of the AM broadcast band.
The NEC simulation results in Figure 4.18 show that small variations in permittivity are not likely to cause significant navigation errors. The maximum difference between the three curves representing the 550-kHz results is about 15 meters. At the other end of the AM spectrum, the maximum difference is only about 10 meters. If the base and rover are reasonably far from the AM transmitters, the error caused by this effect is quite small, probably below the noise.

Since the likely error caused by using a ground model that assumes an incorrect value of ground permittivity is small, I do not attempt to refine the FCC’s estimate. I use a relative permittivity of five.

### 4.3.5 Ground Model

Now that I have a good number for the ground conductivity and an appropriate guideline for the permittivity, I can proceed with developing a ground model. First, I simulate the phase for every frequency used by a Boston-area radio station using NEC. I use 0.0015 mhos/meter for ground conductivity and five for relative permittivity in the simulations. I also simulate the phase for the same frequencies over perfect ground. I subtract the perfect-ground phase from the imperfect-ground phase to produce curves like the ones in Figure 4.18. These curves are well described by exponentials so I fit an exponential to the data for each frequency. I then use these exponentials directly in the calculation of theoretical phase in the navigation program.

Exponentials of the form,

$$be^{-mx} + a$$

are added to the theoretical phase for both the rover and the base. The parameters $a$, $b$ and $m$ are determined for each station from the NEC simulations and $x$ is the distance from the transmitter. The values for $a$, $b$ and $m$ that I determined for Boston-area radio stations are in Table 4.5.
Table 4.5: Ground-model parameters for ground with conductivity 0.0015 mhos/m and relative permittivity five.

The parameters $a$ and $b$ in Table 4.5 are scaled so that the model produces phase corrections in degrees. While the parameter $a$ was a necessary free parameter for the curve fit, it does not need to be included in the model: a constant added to the phase from both the rover and the base has no affect.

To wrap up this section, I compare post-fit ambiguity function residuals, both with and without the ground model. Figure 4.11 shows post-fit phase residuals for experiment one when the model is not used. Figure 4.19 shows results from the same experiment but using the model.
Comparing the results in Figure 4.19 with those in Figure 4.11 reveals the utility of the ground model. The large residuals for 740, 1150, and 1510 all improve dramatically. The residuals for 590 improve dramatically between 20 and 30 minutes but only marginally for times around 10 and 45 minutes. The large residuals that still remain on 590 are probably the result of incorrect antenna drive parameters that are used in the antenna pattern model. The residuals for 680 and 1360 only improve slightly with model use but were not too large when the model was not used.

I also include a histogram of phase residuals both using and not using the ground model. Comparing the phase residuals in Figure 4.19 with those in Figure 4.11 clearly shows how the large residuals for most stations improve with the use of the ground model. However, comparing the two phase residual plots does not reveal more subtle improvements. Figure 4.20 contains a histogram of phase residuals for the time when the rover is farthest from the base, which occurs 27.5 minutes into experiment one. The histogram reveals the more subtle improvements of the model.
Chapter 4: AM Navigation Challenges

4.4 Overhead Conductors

In most of the developed world, overhead conductors are ubiquitous. Power lines, telephone cables, and TV cables are examples of conductors that are commonly found strung from the poles along most of our streets. Regardless of their intended use, these conductors interact with AM broadcast signals.

Typically, overhead conductors do not occur in isolated sections: they form vast interconnected networks. I did not attempt to simulate these vast networks nor did I attempt to account for their effects in my navigation software. Instead, I characterize the effects of the conductors by showing post-fit phase residuals from an experiment in which I purposely put the rover in close proximity to overhead conductors.

During this experiment, which was conducted on 28 January 2002, I investigated three cases of overhead conductors: a single isolated steel guy wire, and two different cases of the more typical situation in which multiple conductors are strung from the power poles. The base receiver in this experiment was located at a GPS-surveyed location (WGS-84: N42.45443 W71.26997) in an unobstructed parking lot on Hanscom Air Force Base in Bedford, MA. The rover was placed in the back of a pickup truck where it remained for
the duration of the experiment. Because the rover was in the back of the truck, the foot of the rover antenna was about two meters above ground level throughout the experiment. In all cases I slowly drove the rover in a straight line perpendicular to the projection of the conductors onto the plane of the ground.

Figure 4.21 shows a map of the base station and rover positions during the experiment.

![Map of receiver positions during an experiment in which the rover was placed in close proximity to overhead conductors. The experiment was conducted on 28 January 2002.](image)

Figure 4.21: Map of receiver positions during an experiment in which the rover was placed in close proximity to overhead conductors. The experiment was conducted on 28 January 2002.

The waypoint labeled “Base” in Figure 4.21 marks the location of the base receiver for the duration of the experiment. The waypoints labeled “Guy Wire,” “Multiple OHC 1,” and “Multiple OHC 2” mark the locations of the overhead conductors. The yellow-highlighted dashed line is the path of the rover as tracked by the GPS receiver. The location of the mapped area is about 25 kilometers NW of Boston. The locations of the AM transmitters are shown in Figure 4.22.
Chapter 4: AM Navigation Challenges

Figure 4.22: Wide-area map of receiver and transmitter positions during an experiment in which the rover was placed in close proximity to overhead conductors. The experiment was conducted on 28 January 2002.

The position of the base receiver is again labeled “Base” in Figure 4.22. The path of the rover is barely visible at this scale. The location of each AM transmitter in the Boston area is marked with an antenna symbol and labeled with the station’s nominal frequency in kHz. Some of the AM stations are not visible in Figure 4.22 because of close proximity to another station.

4.4.1 Isolated Overhead Steel Guy Wire

First, I show the effect of a single overhead steel guy wire. The wire connects the tops of two wooden power poles that are located on Hanscom Air Force Base. The location of the guy wire is marked “Guy Wire” in Figure 4.21. The wire is approximately 90 meters long and 12 meters off of the ground. It is insulated at both ends by the power poles and is oriented such that its long axis is aligned with 0 and 180 degrees.
I slowly drove the rover truck from about 15 meters (horizontally) from the wire to about 15 meters on the other side of the wire. The rover passed directly underneath the wire at about its midpoint. The post-fit carrier-phase residuals are shown in Figure 4.23.

![Figure 4.23: Post-fit phase residuals from a portion of an experiment in which the rover is near an isolated overhead guy wire.](image)

Between the elapsed times of 10.5 and 11.5 minutes the rover was stationary directly under the overhead guy wire. At an elapsed time of 9 and 13 minutes the rover was about 15 meters from the wire but on opposite sides of the wire.

Not surprisingly, the proximity of the guy wire caused the phase to be perturbed. It is also not surprising that the perturbation appears to be symmetric with respect to the wire.

Because AM transmitters launch vertically polarized waves, it may not be immediately clear why a horizontal conductor had an effect on the fields. The solution to this puzzle lies in the ground. The Poynting vector of a ground wave is not parallel to the radial propagation vector. Because the ground is lossy, the Poynting vector is angled somewhat toward the ground, which implies that the electric field tilts toward the ground. Therefore, a horizontal conductor can interact with the electric field component that lies in the horizontal plane. The amount of interaction of course depends on the relative orientation of the conductor in the horizontal plane. Horizontal conductors that lie on a line radiating from an AM transmitter will interact with the signal radiated from the transmitter more than a horizontal conductor that is oriented any other way.

When the rover approached the guy wire, the phase residuals for 850 and 890 changed more than the phase residuals for other stations. Therefore it is not surprising that Figure
4.22 shows that 850 is south of the guy wire and that 890 is south-southwest of the wire. A large phase residual is also apparent for 590 but it does not appear to be correlated with the guy wire.

Figure 4.24 shows the ratio between the amplitude observed by the rover and the amplitude observed by the base station for each AM transmitter.

![Figure 4.24: Ratio between the amplitudes observed at the rover and base station from a portion of an experiment in which the rover is near an isolated overhead guy wire.](image)

The amplitude ratio, expressed in dB, was generally less than zero because the base station was located near the top of Katahdin Hill and the rover was down in a valley. Again, because the guy wire runs north south, we expect the amplitude ratio change caused by the guy wire to be larger for stations to the north and south of the rover. Indeed, the amplitude ratio change caused by the guy wire was large for 650 (co-located with 1200), 800, 850, 890 and 900, which were all generally north or south of the rover as shown in Figure 4.22. However, 590 and 740 were a bit anomalous. Unlike the phase residual change for 590, the amplitude ratio change appears to be correlated with guy wire proximity. The amplitude ratio change was large for 590 even though the station was east southeast of the rover. The amplitude ratio change for 740 was also large even though the station was southeast of the rover. There was another force at work. As mentioned in a previous section, the component of a groundwave that actually travels in the ground dies out more rapidly for stations operating at higher frequencies. Because 590 and 740 are low frequency stations that were relatively close to the rover, the component of the groundwave that travels in the ground near the rover was quite large. Therefore, the tilt of the electric field was greater for these stations than the tilt for other stations. The greater tilt implies more electric field in the horizontal plane, which results
in more interaction with horizontal conductors. This increased interaction explains the large changes in the phase residuals and amplitude ratios that were caused by the guy wire.

### 4.4.2 Multiple Overhead Conductors

The isolated overhead guy wire discussed in the previous section is actually quite rare. In this section, I discuss the much more common case of multiple, extended, overhead conductors.

Figure 4.25 shows the multiple overhead conductors that are along Route 2a near the Brooks Historical Area (Minute Man National Historical Park) in Concord, MA. The location of these conductors is labeled “Multiple OHC 1” on the map in Figure 4.21.

![Figure 4.25: Multiple overhead conductors along Route 2a near the Brooks Historical Area (Minute Man National Historical Park) in Concord, MA.](image)

Starting at the top of Figure 4.25, the first conductor is one phase of overhead power distribution. The second wire, which is bit hard to distinguish from the first, is its ground return. The third is a steel guy wire. A couple meters below, the fourth conductor is cable TV. Finally, the last two conductors are for telephone service. The telephone conductors are about five meters off the ground.

I approached the conductors with the rover starting from about 15 meters away to the north, in the Brooks Historical Area parking lot. I slowly drove the rover directly toward the conductors and stopped underneath for one minute. I was not able to drive beyond the conductors because that would have brought me out into traffic on Route 2a.

The post-fit phase residuals for this portion of the experiment are shown in Figure 4.26.
Figure 4.26: Post-fit phase residuals from a portion of an experiment in which the rover is near multiple overhead conductors.

The post-fit phase residuals, shown in Figure 4.26, were much larger than the guy wire phase residuals, shown in Figure 4.23. That the phase residuals were larger is not surprising considering that the conductors the rover was close to in this case are connected to vast networks of conductors. The large jump in the phase residual for station 740 was just a $2\pi$ phase wrap.

Despite the fact that the rover was near conductors that are part of a vast interconnected network, some of the residuals in Figure 4.26 are predictable. For instance, the large phase residuals for stations 590, 740, and 1000 can be explained by local conductor geometry and the locations of the transmitters. The conductors that the rover was near are parallel to Route 2a, which generally runs east-southeast to west-northwest (see Figure 4.22). Since 590 and 740 were east-southeast of the rover and 1000 was west-northwest of the rover, the large residuals are not surprising.

The large phase residual for station 650, however, is difficult to explain. Station 650 is collocated with station 1200, and was south-southwest of the rover (see Figure 4.22). So, 650 was nearly in a perpendicular direction from the general directional trend of the conductors. Therefore, the conductors should not have interacted with the signal from 650 at all. But the fact remains that these conductors do not just run along Route 2a. There are service feeds that branch off to houses and even long runs down secondary roads. The phase residuals for station 650 indicate a more complex situation than can be explained merely by the general orientation of the conductors.
Figure 4.27 shows the ratios between the amplitudes observed at the rover and the amplitudes observed at the base station for each AM radio station.

Figure 4.27 shows that the signals from several stations were attenuated significantly as the rover approached the conductor. The signals from stations 740, 950, and 1360, in particular, experienced deep nulls. The bearings from the rover to all of these stations ranged from due east to east-northeast (see Figure 4.22) so the large drop in signal strength observed by the rover is not surprising. The deep null on station 1360 that occurred a little after 32 minutes explains the erratic phase residuals that occurred at the same time. An interfering signal probably became stronger than the signal from 1360 so the rover actually reported the phase of the interfering signal.

Up until this point I have implied that the changes in phase residuals and amplitude ratios are the result of interaction between the proximate conductors and the electric fields. One may achieve greater intuition by considering that the conductor network “receives” the signals from the radio stations and reradiates phase-shifted versions that interfere with the direct signals. With this interpretation, however, it is very important to understand that the dominant sources of the radiation are the AM transmitters. If this were not the case, the phase-residual and amplitude-ratio curves would have most likely not been smooth like the ones in Figure 4.26 and Figure 4.27.

Although interference caused by reradiated signals from overhead conductors is the most likely cause of the changes in phase residuals and amplitude ratios, it is important to rule out interference from other sources. It is useful to consider frequency estimates when
attempting to rule out interference from sources other than the AM transmitters. If the base station and rover are not moving, there is no Doppler shift to consider and the frequency of the carrier of a given transmitter will be the same at the rover and base station. The sample clocks in the two receivers do not run at the same rate, however, so the frequencies observed by the rover and base station will differ. Since the rover and base station epochs are nearly simultaneous and the transmitter frequencies do not drift much, the ratios between the frequencies observed at the rover and base station for all transmitter signals will equal the inverse of the ratio between the rover and base station sample clock rates.

At each epoch, I estimated the inverse of the ratio between the rover and base station sample clock rates as the median of the ratio between the observed frequencies at the rover and base station. I used the median because it is immune to the large outliers that occur due to interference. The clock-rate ratio can easily be used to predict what the difference between the observed frequencies at the rover and base station should have been in the absence of measurement error. This predicted frequency difference was subtracted from the measured frequency difference to estimate the measurement error. This measurement error was of course due to rover and base station error but since the base station was in a benign environment, most of the error can be attributed to the rover. I will refer to error estimated in this manner as frequency difference estimation error.

Since the rover was moving exceedingly slowly (less than 0.1 meters/second) during this portion of the experiment and the base station was motionless, the frequency analysis described above can be applied. Figure 4.28 shows the frequency difference estimation error for this portion of the experiment.
4.4 Overhead Conductors

For the most part, the frequency difference estimation error was less than 0.2 Hz. However, the estimation error for station 1360 jumped off the chart up over 11 Hz just after 32 minutes elapsed time. This jump implies that the rover tracked a signal that was offset in frequency by over 11 Hz from the desired signal. This is consistent with my earlier claim that the rover was tracking an interfering signal during this time.

If the large error for 1360 is ignored, the largest frequency difference estimation error occurs for station 1000 and was about 0.17 Hz. Since the resolution of the FFT was only 1.2 Hz, a 0.17 Hz frequency offset is not alarming and certainly does not suggest that the rover was tracking an erroneous signal. All of the other stations had considerably smaller frequency offsets so I conclude that interference from sources other than the AM transmitters themselves did not contribute significantly to the large phase residuals observed near overhead conductors.

Finally, I present results from a portion of the experiment in which the rover is close to another set of multiple overhead conductors. This set had the same total number of conductors but the lowest conductor in the set was closer to the ground than the previous case. The location of these conductors is labeled “Multiple OHC 2” on the map in Figure 4.21. The conductors are located along the Battle Road in Minute Man National Historical Park in Lexington, MA.
Starting from the top of Figure 4.29, the first conductor is one phase of overhead power distribution. The next two conductors are 220-volt distribution wires and the two below that are ground wires. The lowest cable is for telephone service and is about four meters off the ground.

I slowly moved the rover toward the conductors starting from about 15 meters to the southwest. I stopped the rover for about one minute when it was directly below the conductors. I then proceeded to slowly move the rover away from the conductors to a location 15 meters to the northeast. Finally, I moved the rover relatively quickly back the starting point stopping only for about 30 seconds under the conductors.

The post-fit phase residuals for this portion of the experiment are shown in Figure 4.30.
4.4 Overhead Conductors

The post-fit phase residuals for this portion of the experiment indicate a more serious problem than either previous case. Figure 4.30 shows that the conductors affected the signals from more stations than before. In fact, at about 50 minutes elapsed time, when the rover was nearly under the conductors, the ambiguity function method failed. The failure of the ambiguity function method is clear because of the large jump in the phase residuals for all stations. The ambiguity function search resulted in a position estimate that was far removed from the actual position of the rover. When the rover was stopped under the conductors between about 50.2 and 51.3 minutes elapsed time, the ambiguity function method resulted in a position estimate that was wrong, but fairly consistent throughout the interval. However, there were three very conspicuous times, at about 50.2, 50.5, and 50.8 minutes in which the ambiguity function method resulted in yet another wrong position estimate. This behavior is to be expected because when the ambiguity function method fails, there are many peaks that have very similar ambiguity function values. So, even though the rover was motionless, small changes in the noise resulted in the ambiguity function method jumping to another peak.

As the rover moved away from the conductors on the opposite side, the ambiguity function method again chose the correct ambiguity function peak. The phase residuals, shown in Figure 4.30, were quite similar on opposite sides of the conductors, which is not surprising given the symmetry. The quick pass back under the conductors is evident from the figure at about 53.8 minutes and the phase residuals were similar to those from the first pass.
There were likely many factors that resulted in the larger phase residuals for this case of multiple overhead conductors. Chief among them may have been the fact that the conductors were closer to the rover. The lowest “Multiple OHC 1” conductor was five meters off the ground whereas the lowest “Multiple OHC 2” conductor was only four meters off the ground. Since the top of the rover antenna was close to 3 meters off the ground, the spacing between the antenna and the conductor in the two cases was significantly different.

Clearly, nearby overhead conductors can result in significant phase errors. In fact, I have shown that the phase errors caused by directly overhead conductors can result in ambiguity resolution failure. Because overhead conductors most often form large interconnected networks, modeling the effects of overhead conductors is difficult. However, the results in this section show that it may be possible to improve positioning performance by throwing out measurements from stations that have suddenly anomalous amplitudes.

### 4.5 Underground Conductors

Conductors need not be overhead to affect the phase measurements of AM radio signals. Since AM signals travel via groundwave, underground conductors can also cause problems. Just like overhead conductors, underground conductors are ubiquitous in the developed world. Plumbing is common everywhere and underground electric, telephone, and cable TV is common in many places, especially cities. In this section, I investigate the effects of underground conductors on AM radio signals.

#### 4.5.1 Underground Water Pipe

I first conducted an experiment in which the rover was near a two-meter-diameter underground water pipe. The water pipe under test is the Hultman Aqueduct, which carries water from the Quabbin Reservoir, in Western Massachusetts, to Boston. This experiment was conducted on 25 January 2002. The base station was placed at a GPS surveyed position (WGS-84: N42.45443 W71.26997) in a large open parking lot at Hanscom Air Force Base, which is located in Bedford, MA. I slowly moved the rover back and forth along Flanagan Street in Framingham, MA. The Hultman Aqueduct crosses perpendicularly under Flanagan Street. The location of the base station and water pipe are shown on a map in Figure 4.31.
Figure 4.31: Map base station and water pipe positions during an experiment in which the rover is near an underground water pipe. The experiment was conducted on 25 January 2002.

The Hultman Aqueduct is made of concrete lined with steel and is approximately two meters in diameter. I estimate that the pipe is only about one meter below Flanagan Street. The post-fit phase residuals for this experiment are shown in Figure 4.32.
Chapter 4: AM Navigation Challenges

Figure 4.32: Post-fit phase residuals from an experiment in which the rover is near a two-meter-diameter steel-lined water pipe.

Surprisingly, the post-fit phase residuals were not correlated with the proximity of the pipe. The rover passed directly over the pipe at 41.42, 46.33, 47.83, and 49.25 minutes elapsed time. The rover was also stationary over the pipe between 49.5 and 50.5 minutes elapsed time. There was virtually no detectable change in the post-fit residuals for any station at any of these times. The somewhat erratic behavior of 590, 1510, and 1600 was not correlated with the pipe. The large jump in the phase residuals at around 47 minutes elapsed time occurred when the rover passed under some overhead conductors.

Since the aqueduct is over 100 kilometers long, I expected a dramatic change in the phase residuals when the rover was near the pipe. However, I did not know a key characteristic of the pipe. It turns out that the steel lining of the pipe is not welded between pipe sections so the pipe is not a continuous conductor. Therefore, the section of pipe that runs under Flanagan road is isolated from the rest of the pipe. I was not able to find out the length of this section of pipe but judging from the above results, it is likely quite short.

The amplitude and frequency ratios for this experiment were similarly unaffected by the pipe. I omit plots of these ratios and move on to describing another experiment.

4.5.2 Briggs Field Underground Conductors

I also conducted an experiment on Briggs Field, which is located on the MIT campus in Cambridge, MA. Briggs Field is nearly 15 acres of grass-covered open space. I obtained
drawings from the MIT Facilities Department that showed the locations of underground utilities in the field. I conducted the experiment on 12 February 2002. The base station was placed at a GPS-surveyed location (WGS-84: N42.35695 W71.09881) on Briggs Field well away from any conductors indicated on the drawings. Figure 4.33 shows a map of receiver positions during the experiment.

![Map of receiver positions during an experiment in which the rover is moved over underground conductors. The experiment was conducted 12 February 2002 on Briggs Field, which is located on the MIT campus in Cambridge, MA.](image)

The small round circles in Figure 4.33 indicate the position of the rover as estimated by a Garmin GPS III+ handheld GPS receiver. The small solid black squares indicate the AM navigation rover position estimates. The purple lines link corresponding GPS and AM position estimates. The position estimates occurred every five seconds. The location of the base station is marked “Base” in Figure 4.33. The long purple line bisecting the path of the rover indicates the location of underground steam pipes, electricity wires, and telephone wires as shown in the drawings I obtained from the MIT Facilities Department. The “Gxx-xx” labels mark the location of the rover, according to the GPS receiver, at the indicated elapsed time in minutes where the dash should be interpreted as a decimal point. The “Goal post” label marks the location of a football goal post that I drove the rover under during the experiment at about 28 minutes elapsed time.

As shown in Figure 4.33, I first drove the rover at about one meter/second to a position about 25 meters northeast of the underground conductors. I then proceeded slowly toward the conductors passing them at 5.5 minutes elapsed time and proceeded slowly beyond to 25 meters southwest. I turned around and repeated the pass in the opposite direction. Finally, I drove the rover in a large loop around the field at about one meter/second.
As shown in Figure 4.33, the GPS and AM position estimates differed by as much as 40 meters during the experiment. GPS position estimation error using the Garmin receiver was probably less than ten meters, so the error in the AM position estimates was quite large. Interestingly, the largest position estimation errors do not seem to be correlated with the rover being close to the known underground conductors. In fact, the position errors when the rover was near the conductors are among the smallest indicated in Figure 4.33. The largest errors occurred when the rover was over 40 meters northeast of the conductors. It is also interesting to compare the position estimation errors in this experiment with previous results for similar baselines but from other places. Figure 3.7 and Figure 3.10 show that position estimation errors during two previous experiments never exceeded five meters for baselines up to almost 300 meters. In this experiment, however, baselines less than 50 meters sometimes resulted in position estimation errors that exceeded 20 meters.

The ambiguity function value at each epoch is plotted in Figure 4.34.

![Figure 4.34: Ambiguity function value during an experiment in which the rover is moved over underground conductors. The experiment was conducted 12 February 2002 on Briggs Field, which is located on the MIT campus in Cambridge, MA.](image)

Figure 4.34 shows that the ambiguity function value during the experiment got almost as low as 0.75 during the experiment. Figure 4.33 shows that the known conductor crossings occurred at 5.5, 9.75, 23.17, and 28.83 minutes elapsed time. The known conductors clearly had an affect on the ambiguity function value. The ambiguity function value reached local minima at 5.5 and 9.75 minutes when the rover crossed these conductors. Also, when the rover crossed these conductors more quickly at 23.17 and 28.83 minutes the ambiguity function made brief but unmistakable downward excursions.
However, other features of the ambiguity function are more prominent. For example, at about two minutes elapsed time, when the rover moved away from the base station, the ambiguity function value dropped precipitously to under 0.9. The ambiguity function also dropped precipitously at about 25 minutes and did not rise again until about 30 minutes. The position of the rover when various prominent features of the ambiguity function occurred is indicated in Figure 4.33. Most of these features were not correlated with the known underground conductors.

Clearly there is something other than the underground conductors shown on the MIT Facilities Department drawings that are affecting the AM measurements. Figure 4.35 is a photograph of Briggs Field.

![Figure 4.35: Photograph of Briggs Field, which is located on the MIT campus in Cambridge, MA.](image)

Figure 4.35 shows that Briggs Field is about as open as a field can get, free from buildings and overhead conductors. Because the field is so open, conductors under the surface of the field must have been responsible for the AM signal measurement errors. However, it is clear from the results presented in Figure 4.33 and Figure 4.34 that many more underground conductors exist under Briggs Field than those indicated on MIT Facilities Department drawings. This is actually not too surprising. During World War II, many temporary buildings were built on Briggs Field to provide space for MIT’s war efforts. It would be interesting to do a raster scan of the field with an AM navigation receiver but such an exercise is beyond the scope of this thesis.

It is quite likely that an unmapped conductor lies under Briggs Field quite close to the base station position. Because of the short baselines during this experiment, the ambiguity function value should have been greater than 0.98 for all epochs. Figure 4.34 shows that the ambiguity function value was never that large except when the rover was quite near the base station. So, unless the rover was always near an underground conductor, which seems unlikely, the base station must have been near an underground conductor. It is, of course, possible to position the base station well away from conductors so smaller errors than those encountered in this experiment can be expected.
For completeness and to provide further evidence of additional conductors under Briggs Field, I include a plot of the post-fit carrier-phase residuals in Figure 4.36.

Figure 4.36: Post-fit phase residuals from an experiment in which the rover is moved over underground conductors. The experiment was conducted 12 February 2002 on Briggs Field, which is located on the MIT campus in Cambridge, MA.

The changes in the post-fit phase residuals for individual stations were even more pronounced than those for the ambiguity function value. There were clearly systematic errors indicated by Figure 4.36 that cannot be explained by the mapped underground conductors.

Finally, in Figure 4.37, I include a plot of the frequency difference estimation errors for this experiment to show that interference from sources other than the AM transmitters themselves did not play a significant role in the errors.
Figure 4.37: Frequency difference estimation error from an experiment in which the rover is moved over underground conductors. The experiment was conducted 12 February 2002 on Briggs Field, which is located on the MIT campus in Cambridge, MA.

Figure 4.37 shows that the frequency-difference estimation errors were noisier than those from a previous experiment, shown in Figure 4.28. However, the noisier frequency ratios for the few stations that contributed to the larger spread can be simply explained by the lower signal strengths that resulted from the increased distance between the receivers and the transmitters. Furthermore, even the largest indicated frequency estimation error, about 0.35 Hz for station 1000, was much less than the FFT resolution, 1.2 Hz. Therefore, it is very unlikely that interference from other signal sources played a significant role in the errors encountered in this experiment.

The Briggs Field experiment clearly shows that underground conductors can affect AM signal measurements. Position estimation errors as large as 40 meters and ambiguity function values as small as 0.75 resulted from underground conductor proximity. The measurement errors resulting from underground conductors were generally smaller than errors resulting from overhead conductors. Furthermore, the errors resulting from underground conductors were never observed to cause ambiguity resolution failure.

4.6 Skywave

At night, when the D-layer of the ionosphere recombines, signals in the AM broadcast band reflect off the ionosphere enabling so-called skywave propagation [8]. Skywave can greatly extend the useful range of AM signals. I once listened to a Boston Bruins
hockey game on Boston station WBZ 1030 while traveling in Washington, D.C., which is over 600 kilometers away. Most readers have probably had similar experiences. While this extended range is quite amazing, skywave presents challenges to the AM broadcasting industry in general and to AM navigation in particular.

Because of skywave, regulating AM stations is quite complicated. Skywave not only extends the range of the radio station you are trying to listen to but it also extends the range of other stations, including ones that have the same frequency as the station you are trying to listen to. Obviously, this can lead to unwanted interference, so the FCC regulates stations differently at night to account for the extended range caused by skywave. Most stations are required to reduce power and to use directional antenna arrays at night. The nighttime patterns of AM stations are often specifically designed with nulls in the directions of similar-frequency distant stations.

For the navigator, skywave and the resulting regulations have profound implications. On just a practical level, the navigator must maintain day and night antenna databases for all stations. More importantly, the reduced power allowed at night reduces the effective range of an AM navigation system. Regulations aside, skywave signals interfere with groundwave signals. Because of the uncertain nature of the reflection, it is difficult to predict how the amplitude and phase of a skywave signal vary with distance from the transmitter.

To investigate the effects of skywave on navigation performance, I conducted the same experiment at noon and midnight. The daytime experiment was conducted on 7 March 2002 and the nighttime experiment was conducted on 16 April 2002. In both experiments, the base station was placed at a GPS-surveyed location (WGS-84: N42.45455 W71.26997) in a large parking lot located on Hanscom AFB in Bedford, MA and the rover was placed at a GPS-surveyed location (WGS-84: N42.40858 W71.18848) in the driveway of a home in Belmont, MA. Figure 4.38 shows a map of the base station and rover positions during both experiments.
4.6 Skywave

Figure 4.38: Map of receiver locations during two experiments designed to examine the effects of skywave. The daytime experiment was conducted on 7 March 2002. The nighttime experiment was conducted on 16 April 2002.

As Figure 4.38 shows, the distance between the base station and the rover was about eight kilometers. First, I show the amplitude reported by the rover for both the daytime and nighttime experiments in Figure 4.39 and Figure 4.40, respectively.
Chapter 4: AM Navigation Challenges

Figure 4.39: Amplitude reported by the rover during a daytime experiment designed to examine the effects of skywave. The experiment was conducted on 7 March 2002.

Figure 4.40: Amplitude reported by the rover during a nighttime experiment designed to examine the effects of skywave. The experiment was conducted on 16 April 2002.
The amplitude plots reveal some significant differences between daytime and nighttime AM navigation. The most obvious observation is the vastly fewer number of radio stations. During the day, 29 stations were usable for navigation in the Boston area. At night, many stations went off the air and others drastically reduced power levels so that the number of possibly usable stations dropped to 18. This number was further reduced by skywave interference.

The weaker signals had rapidly fluctuating amplitudes, as shown in Figure 4.40. These fluctuations are evidence of skywave interference. As I shall soon show, the signals from these weak stations were useless for navigation. First, however, I provide further evidence that interference was the culprit. In Figure 4.41, I show the frequency difference estimation errors for the nighttime experiment. I do not include a similar plot for the daytime experiment because it is similar to Figure 4.37.

![Figure 4.41: Frequency difference estimation error during a nighttime experiment designed to examine the effects of skywave. The experiment was conducted on 16 April 2002.](image)

Figure 4.41 shows that the rover, base station, or both reported erratic frequency estimates for several stations: 610, 1280, 1370, 1430, 1460, and to a lesser extent, 1260. Not surprisingly, these stations were the same stations that had the rapidly fluctuating amplitudes as shown in Figure 4.40. Because the frequency difference estimation errors for these stations were larger than the 1.2 Hz resolution of the FFT, it is clear that the rover and base station sometimes tracked different signals.

There are a number of ways that skywave can cause the rover and base station to track different signals. First, when a station is moderately distant, as was the case for 610, 1280, 1370, and 1460, the skywave component of the received signal can be significant,
even dominant, at night. Both the amplitude and phase of a skywave signal are not stable over time. If the groundwave and skywave signals are similar in amplitude, phase variation of the skywave signal will lead to interference fading of the received signal. So, even when the skywave signal is not dominant, amplitude fluctuations can be expected in the received signal. These fluctuations can cause the amplitude of the desired signal to drop below the noise level, in which case the receiver will report the frequency of some spurious signal that happens to have greater amplitude than the desired signal.

Another way skywave can cause the rover and base station to track different signals is by enhancing the signals from very distant stations such that they are stronger than the signals from local stations. It is very likely that a combination of local station fading and distant station enhancement lead to the results shown in Figure 4.41.

Regardless of the mechanism by which skywave causes erratic amplitude and frequency estimates, the phase measurements of the affected signals are useless for navigation. Figure 4.42 shows the post-fit phase residuals for the nighttime experiment.

Figure 4.42: Post-fit phase residuals from a nighttime experiment designed to examine the effects of skywave. The experiment was conducted on 16 April 2002.

The post-fit phase residuals for stations 610, 1280, 1370, 1430, 1460, and 1260 appear to have been completely random, as shown in Figure 4.42. For obvious reasons, the phase measurements from these stations were not included in the calculation of the position estimates that were used to compute the post-fit phase residuals. In addition, the phase measurements from 680 were also not used because of the large but somewhat constant phase residuals for that station. The large phase residuals for 680 were most likely due to
the use of nighttime antenna array parameters from the FCC database that were not correct.

At night, skywave reduces the number of usable stations in the Boston area to only 12. Ambiguity resolution and instantaneous position determination are still possible with this number of stations but position estimation accuracy is reduced. Figure 4.43 and Figure 4.44 show scatter plots of rover position estimate for the daytime and nighttime experiments, respectively.

![Scatter plot of rover position estimates during a daytime experiment designed to examine the effects of skywave. The experiment was conducted on 7 March 2002.](image)
Figure 4.44: Scatter plot of rover position estimates during a nighttime experiment designed to examine the effects of skywave. The experiment was conducted on 16 April 2002.

The daytime position scatter, shown in Figure 4.43, was much smaller than the nighttime position scatter, shown in Figure 4.44. The large nighttime position scatter was not just due to the fewer number of stations. Most likely, more subtle skywave interference than that discussed above affected the phase measurements from the remaining stations.

In summary, skywave has profound effects on nighttime AM navigation. Because many stations reduce power or completely go off the air at night, the number of stations that can be used for navigation drops significantly at night. Many of the remaining stations are so corrupted by skywave that the phase measurements are useless for navigation. Even the signals that can still be used are more subtly affected by skywave, resulting in reduced positioning accuracy.

4.7 Concluding Remarks

In this chapter, I have explored several impediments to AM navigation. I showed that the transmitter antenna positions listed in the FCC database have significant errors, which can lead to position determination errors. I demonstrated that this problem is easily overcome by surveying the antennas. I showed that the directional antennas that many AM stations use can lead to significant errors if the phase variation caused by them is not accounted for. I demonstrated that using a simple directional-antenna model reduces these errors dramatically. Variations in propagation velocity associated with electrical properties of the ground can also lead to significant errors if not accounted for, especially
for low-frequency stations and stations that are close to either the base station or rover. I showed that propagation errors are reduced using a model developed from NEC-4 modeling results. Proximate overhead and underground conductors distort the received signals, sometimes to the extent that ambiguity resolution fails. Skywave limits the number of stations that are available for nighttime navigation and reduces positioning performance.
Chapter 5: Navigation Performance

In this chapter, I demonstrate the positioning performance of the AM navigation system in various environments. In the previous chapter, I highlighted some of the issues that arise when navigating using AM broadcasts. That discussion may have left the reader with the incorrect notion that AM navigation does not work well. To the contrary, navigation using AM signals works quite well, yielding meter-level position estimates for baselines up to 40 kilometers. In this chapter, I will present the positioning results of several experiments that highlight the positioning capabilities of AM navigation.

5.1 Zero-Baseline Experiment

The most basic test of any differential radiolocation system is to test the system’s ability to determine position when both receivers are fed the same signal. Such a test, which I will refer to as a zero-baseline experiment, is useful to confirm algorithms and to provide a basis for comparison with more significant positioning tasks. I conducted a zero-baseline experiment for the AM navigation system.

I used short clip leads to connect the antenna and chassis “ground” terminals of the two receivers. I conducted this experiment on 2 May 2001. The receivers were inside Annex 6 at MIT Lincoln Laboratory in Lexington, MA during the experiment.

A scatter plot of position estimates for this experiment is shown in Figure 5.1.
Figure 5.1: Scatter plot of position estimates for a zero-baseline experiment conducted on 2 May 2001.

Figure 5.1 shows that the position estimates were minimally scattered but they had a non-zero bias to the southwest. Since the antenna terminals of the receivers were connected, the position estimates should have been centered on the origin. The resulting bias was small, however, and certainly tolerable in the context of the positioning errors that occurred for longer baselines.

The most likely cause for the bias in the data of Figure 5.1 is the fact that both receivers were plugged into AC power via extension cords. The AC wiring in Annex 6, or almost any building for that matter, has large RF currents induced in it by the incident broadcast signals. Because both receivers were plugged into AC power, the RF current in the building wiring flows through the clip lead that was used to connect the antenna terminals and especially through the clip lead that connects the chassis grounds. Since the clip leads have nonzero impedance and electrical length, there is a voltage drop across the clip leads to which the receivers responded.

The post-fit phase residuals from this experiment are shown in Figure 5.2.
5.2 Outdoor Experiments

The post-fit phase residuals, shown in Figure 5.2, were generally small but provide further evidence of the effects of the AC power lines. The residuals for most stations fell between zero and -2 meters. The residuals for four stations, however, were clustered between 2 and 4 meters. These four stations, 1370, 1510, 1550, and 1600 were the highest frequency stations used in this experiment. That the largest phase residuals occurred for the stations with the highest frequencies is not surprising since the phase shift in the clip leads increases with frequency.

Despite the evidence of small signals entering the receivers through paths other than the antenna terminals, the zero-baseline positioning errors were quite small. The zero-baseline results show that the system is capable of meter-level positioning.

5.2 Outdoor Experiments

In this section, I demonstrate the performance of an AM navigation system during several outdoor experiments. In all of these outdoor experiments I compare the AM position estimates with those produced by a Garmin GPS III Plus handheld GPS receiver. The use of position estimates from this receiver trades positioning performance for tracking performance. The Garmin only produces position estimates that have an accuracy of between five and ten meters. Interestingly, using DGPS corrections with the Garmin does not improve positioning performance significantly. A geodetic DGPS setup, such as the pair of Ashtech Z-12s that I used to survey the AM transmitter antennas, would certainly be more accurate than the Garmin receiver. The Garmin receiver, however, is
far superior to the Ashtech receiver in tracking performance. While attempting to use an Ashtech Z-12 while driving, I discovered that it would almost always lose lock on satellite signals when I drove under an overpass. It also would not quickly reacquire the satellite signals when out from under the shadow of the overpass. The Garmin receiver, on the other hand, almost never loses lock. Given this tradeoff, I chose to use the Garmin because the Ashtech simply would not work during significant portions of the experiments described below.

5.2.1 Hanscom Air Force Base Experiment

I conducted an experiment on Hanscom Air Force Base in Bedford, MA on 13 July 2001. I placed the base receiver at a GPS-surveyed location (WGS-84: N42.45639 W71.26670) in the MIT Lincoln Laboratory Annex 6 parking lot, which is located on the base. I placed the rover in the back of a pickup truck and drove around the base, reaching a maximum distance from the base station of two kilometers. A map of base station and rover positions during the experiment is shown in Figure 5.3.

Figure 5.3: Map of rover positions during an experiment conducted on Hanscom Air Force Base on 13 July 2001.

The AM and GPS rover position estimates are indicated with solid black squares and black circles, respectively, in Figure 5.3. Corresponding AM and GPS position estimates
are connected with purple lines. In general, the position estimates from the two systems agreed so well that it is difficult to see the purple lines. However, near the base station and in the northwest part of the map, there were some significant errors. All of these errors can be attributed to nearby overhead conductors.

It is interesting to see how inaccurate and incomplete the background map is in Figure 5.3. The map is a screenshot from the Garmin Mapsource software (Version 4.09, 11 June 2002). Both the AM and GPS position estimates diverged significantly from the roads. Also, it is clear that many roads at Hanscom Air Force Base are not mapped.

Figure 5.4 shows the difference between the AM and GPS position estimates as a function of time.

![Figure 5.4: Difference between AM and GPS position estimates during an experiment conducted on Hanscom Air Force Base on 13 July 2001.](image)

Figure 5.4 shows that the difference between the AM and GPS position estimates was usually less than 20 meters but got as high as 140 meters. So the reader can better understand the statistics, I include Figure 5.5, which is a histogram of the position differences.
Figure 5.5: Histogram of the difference between AM and GPS position estimates during an experiment conducted on Hanscom Air Force Base on 13 July 2001.

The median position difference was 6.2 meters and the 95\textsuperscript{th} percentile was 20.8 meters. All of the position differences greater than 25 meters can be attributed to overhead conductors. A very useful side benefit of using the ambiguity function method is the correlation between the maximum ambiguity function value and position errors. Figure 5.6 shows the maximum ambiguity function value for this experiment.
Comparing Figure 5.4 and Figure 5.6 reveals that the ambiguity function value is an exceedingly good predictor of positioning performance. All of the large position differences, shown in Figure 5.4, had corresponding large dips in the ambiguity function value plot, shown in Figure 5.6. There were 337 epochs in this experiment, 87% of which have an ambiguity function value that exceeded 0.95. Among these epochs that had an ambiguity function value greater than 0.95, the median position difference was 5.5 meters and the 95th percentile was 14.6 meters. Considering that five to ten meters of error can be attributed to the Garmin GPS receiver, the AM navigation system did quite well when the ambiguity function value was greater than 0.95.

5.2.2 Route 2 Experiment

I conducted an experiment in which I drove the rover along Route 2 between Cambridge, MA and Harvard, MA on 22 January 2002. I placed the base station at a GPS-surveyed location (WGS-84: N42.45450 W71.27003) in a large open parking lot on Hanscom Air Force Base in Bedford, MA. I placed the rover in the back of a pickup truck and drove to Route 2 via I-95, then east on Route 2 to Cambridge, MA, then back west on Route 2 to Harvard, MA, and finally back east on Route 2 for about two kilometers. The longest distance between the rover and base during the experiment was 35 kilometers. A map of base station and rover positions during the experiment is shown in Figure 5.7.
Chapter 5: Navigation Performance

Figure 5.7: Map of base station and rover positions during an experiment conducted along Route 2 on 22 January 2002.

Again, the AM and GPS position estimates are indicated with solid black squares and black circles, respectively, but at the scale shown in Figure 5.7, individual markers are impossible to distinguish. Figure 5.8 shows the difference between the AM and GPS position estimates as a function of time.

Figure 5.8: Difference between AM and GPS position estimates during an experiment conducted along Route 2 on 22 January 2002.

Figure 5.8 shows that the position estimates from AM and GPS generally differed by less than 20 meters but there were differences as large as 160 meters. All of the large position differences can be attributed to overhead conductors: either utilities or overpasses. The differences did not increase as the baseline increased. The differences actually were
slightly smaller at 60 minutes, when the rover was 35 kilometers away from the base, than at 12 minutes, when the rover was only two kilometers away from the base. This shows that the ground model developed in the previous chapter is quite effective. A histogram of the position differences is shown in Figure 5.9.

The histogram in Figure 5.9 peaks at 15 meters, which is 10 meters greater than the value for which the histogram peaked for the previous experiment. The median position difference was 11.4 meters and the 95th percentile was 26.1 meters. So, both the median and the 95th percentile position difference increased by about 5 meters compared to the much shorter baseline experiment conducted on Hanscom Air Force Base. These statistics, combined with my previous observations about Figure 5.8, indicate that AM navigation errors increase as the baseline exceeds a few kilometers but then stabilizes as the baseline increases to 35 kilometers.

Again, low ambiguity function values were highly correlated with large position differences. The ambiguity function value for this experiment is shown in Figure 5.10.
A comparison of the drops in the ambiguity function value in Figure 5.10 with the large position differences in Figure 5.8, in general, shows high correlation. However, there are a few notable exceptions.

There is no rise in the position differences corresponding to the systematic dip in the ambiguity function value that occurs at about 21.5 minutes elapsed time. This dip in the ambiguity function value occurs when the rover reaches the bottom of Belmont Hill and enters the marshy area between Spy Pond and Little Pond. The resulting propagation uncertainty caused by the change in ground properties explains the dip in the ambiguity function. There is not a significant positioning error associated with this dip in the ambiguity function value because the signals from only a few stations (ones that are close to the rover and/or ones that have low frequencies) are significantly affected.

The position differences at 12.67, 23.5, and 54 minutes elapsed time were 53, 90, and 97 meters, respectively. The corresponding ambiguity function values, 0.97, 0.88, and 0.93, respectively, were extraordinarily high considering the large position differences. All of these epochs occurred when the rover was directly under overpasses (I-95 under Route 2, Route 2 under Route 60, and Route 2 under Littleton Road, respectively). One possible explanation is that the GPS receiver produced anomalous position estimates. This seems plausible since the GPS receiver obviously did not have a good view of the sky when under the overpasses. However, an inspection of a zoomed-in version of the map in Figure 5.7 reveals that the Garmin GPS III Plus position filters did quite well when dealing with the sky obstruction. The AM position estimates were clearly responsible for the vast majority of the position differences for these three epochs. This is quite
5.2 Outdoor Experiments

astonishing: the 26 stations that were used to form the ambiguity function in this experiment agreed quite well on the wrong position when the rover was directly under an overpass. In other words, the ambiguity resolution was quite robust but the position estimate was very wrong.

Fortunately, the navigator can observe amplitude to detect passage under an overpass. For the three epochs mentioned above, the amplitude reported by the rover dropped by 15, 20, and 25 dB, respectively, for all radio stations. This significant, simultaneous drop in the amplitude for all stations occurred almost exclusively when the rover passed under an overpass. Occasionally, overhead conductors caused a similar simultaneous drop in the amplitude, but these drops were almost always accompanied by a low ambiguity function value. Regardless of the cause, the navigator should be wary of large drops in observed amplitude.

The rover passed directly under overpasses seven times in addition to the three mentioned above. The rover passed under many other overpasses, but a sample did not occur while the rover was under these. For the additional seven, the significant, simultaneous drop in the observed amplitude for all stations occurred in all cases. Also, the ambiguity function for all seven cases remained high as before, but the large position errors did not occur. So, a simultaneous drop in the observed amplitude for all stations does not always imply a large position error, but such drops are correlated with large position errors enough that the prudent navigator should avoid using position estimates from such epochs.

Despite the specific cases mentioned above, in general, the ambiguity function value was highly correlated with position errors. In this experiment, 77% of the epochs had an ambiguity function value greater than 0.95. Among these epochs, the median was 11.1 meters and the 95th percentile was 21.9 meters. So, the 95th percentile dropped by a little over four meters when low-ambiguity-function-value epochs are ignored.

5.2.3 Route 128 Experiment

I conducted an experiment in which I drove the rover along Route 128 between Lexington, MA and Beverly, MA on 20 December 2001. The base station was placed at a GPS-surveyed location (WGS-84: N42.45455 W71.26995) in a large open parking lot on Hanscom Air Force Base in Bedford, MA. I placed the rover in the back of a pickup truck and drove off the base on Hartwell Avenue, then turned east on Route 4, and then north on Route 128 up to Beverly, MA. The maximum distance between the rover and base station during the experiment was 40 kilometers. Figure 5.11 shows a map of base station and rover positions during the experiment.
Chapter 5: Navigation Performance

Figure 5.11: Map of base station and rover positions during an experiment conducted along Route 128 on 20 December 2001.

As before, the AM and GPS position estimates are indicated with solid black squares and black circles, respectively. At the scale shown in Figure 5.11, individual markers are impossible to distinguish. Figure 5.12 shows the difference between the AM and GPS position estimates as a function of time.

Figure 5.12: Difference between AM and GPS position estimates during an experiment conducted along Route 128 on 20 December 2001.

Figure 5.12 shows that the position estimates generally differed by less than 20 meters for the first 30 minutes of the experiment. After 30 minutes, the position differences increased somewhat, but still were generally less than 40 meters. There were several
cases in which the position differences were quite large, reaching up to 190 meters. These large position differences can all be attributed to overhead conductors. Figure 5.13 shows a histogram of the position differences.

![Histogram of position differences](image)

**Figure 5.13:** Histogram of the difference between AM and GPS position estimates during an experiment conducted along Route 128 on 20 December 2001.

The histogram in Figure 5.13 is very similar in shape to the histogram for the previous experiment in Figure 5.9. The histogram for this experiment, however, indicates more epochs with large errors. The median position difference was 12.1 and the 95th percentile was 38.0 meters. The median was not substantially worse than the median from the previous experiment but the 95th percentile was larger by almost 12 meters. Figure 5.12 shows that larger position errors occurred later in the experiment when the rover was far from the base station. The errors, however, were not the result of an increase in propagation uncertainty associated with the increased baseline. If they were, the ambiguity function, plotted in Figure 5.14, would almost certainly have dropped in value as a result of the propagation uncertainty.
As shown in Figure 5.14, the ambiguity function value remained high throughout this experiment except for a few isolated epochs. If propagation uncertainty caused the increase in position differences late in the experiment, then the ambiguity function would have almost certainly reflected this. Since it clearly did not, there must be another explanation.

At 31 minutes elapsed time, about when the position differences started getting worse, the rover passed the intersection where Route 128 splits off from I-95 to the north of Boston. I suspect that the increase in position differences was due to a difference in the construction of the roadbed when the freeway changes from a federal interstate to a state highway. Perhaps on 128, north of I-95, the steel rebar in the roadbed is welded into larger sections, which interact with the AM signals more. Although less severe, the large phase differences accompanied by high ambiguity function values in this experiment was similar to the phenomenon observed in the previous experiment for overpasses.

There were three large position differences in Figure 5.12 that do not have corresponding low ambiguity function values. The rover was directly under overpasses during these three epochs. Like the previous experiment, the amplitude dropped for all stations when the rover was under the overpasses.

The ambiguity function value still predicted position differences reasonably well despite the above observations. In this experiment, 84% of the epochs had an ambiguity function value that exceeded 0.95. Among these epochs the median position difference was 11.9
meters and the 95th percentile was 33.9 meters. The median and the 95th percentile dropped 0.2 meters and 4.1 meters, respectively, from when all epochs were considered.

### 5.2.4 Urban Experiment

I conducted an experiment in which I drove the rover into downtown Boston to test the effectiveness of the AM navigation system in an urban environment. The system performed very poorly in the urban setting.

The urban experiment was conducted on 8 February 2002. I placed the base receiver at a GPS-surveyed location (WGS-84: N42.45644 W71.26676) in the MIT Lincoln Laboratory Annex 6 parking lot in Lexington, MA. I placed the rover in the back of a pickup truck and drove into Boston via Route 128 and Route 2 and then back via I-90 and Route 128. A map of base station and rover positions during the experiment is shown in Figure 5.15.

![Figure 5.15: Map of base station and rover positions during an experiment conducted in an urban area on 8 February 2002.](image)

As before, the AM and GPS position estimates are indicated with solid black squares and black circles, respectively, and corresponding position estimates are connected with purple lines. In general, the AM and GPS position estimates agreed well enough that the purple lines in Figure 5.15 are not visible. However, in downtown Boston the positioning...
errors were so large that the purple lines are visible. Figure 5.16 shows a close-in view of rover position estimates while the rover was in downtown Boston.

![Map of rover positions during an experiment conducted in an urban area on 8 February 2002.](image)

The map in Figure 5.16 shows that I drove the rover east on Storrow Drive, south on Clarendon Street, east on Boylston Street, south on Tremont, east on Kneeland Street through the Financial District, south on Hudson Street, and finally west on I-90. For much of this route through the city, tall buildings line both sides of the streets. While the rover moved through the city, ambiguity resolution failed for almost all epochs, resulting in large position errors. Only a few good AM position estimates occurred while the rover was along Clarendon Street and along Boylston Street near the Public Garden and Boston Common.

The Garmin GPS receiver also lost satellite-signal lock several times in the city. However, the filtering the receiver does is quite good so the GPS position estimates remained reasonable through the loss-of-lock outages.

The position differences for the experiment are plotted as a function of time in Figure 5.17.
5.2 Outdoor Experiments

![Graph showing position difference (meters) versus elapsed time (minutes).]

Figure 5.17: Difference between AM and GPS position estimates during an experiment conducted in an urban area on 8 February 2002.

The rover turned onto Clarendon Street at about 31 minutes elapsed time and emerged from the urban area back onto open highway at about 44 minutes elapsed time. The position differences during the time the rover was in Boston were exceedingly large as shown in Figure 5.17. Most of these large differences were the result of ambiguity function failure, so the errors could have been even larger if the ambiguity function search space was larger. A histogram of the position differences is shown in Figure 5.18.
Figure 5.18: Histogram of the difference between AM and GPS position estimates during an experiment conducted in an urban area on 8 February 2002.

As expected, the histogram in Figure 5.18 shows that a large number of epochs had very large position differences. The median position difference was 16.2 meters and the 95th percentile was a whopping 185 meters. The only somewhat good news is that the ambiguity function value, shown in Figure 5.19, predicted these dismal errors quite well.
All of the large position differences encountered while the rover was in the city were accompanied by low ambiguity function values. Only 31% of the epochs in this experiment had an ambiguity function value greater than 0.95. Among these epochs, the median position difference was 11.6 meters and the 95th percentile was 27.9 meters. Comparing these numbers with those above shows that the ambiguity function value predicted large position errors quite well.

5.3 Forest Experiment

I conducted an experiment in which I drove the rover through a forest on the Nobscot Boy Scout Reservation in Sudbury, MA. GPS receivers, including the Garmin GPS III Plus, do not track well under the cover of dense foliage. I conducted this experiment during the winter, when the leaves are not on the trees, with the hope that GPS position estimates could be obtained and compared to AM position estimates. The Garmin GPS receiver did, in fact, maintain signal lock throughout this experiment so a comparison between the AM and GPS position estimates was possible. However, it is clear from the results I will present that the GPS position estimates were not as good as the AM position estimates.

I conducted this experiment on 6 February 2002. I placed the base station at a GPS-surveyed location (WGS-84: N42.35058 W71.43678) in an open field located on the Nobscot Boy Scout Reservation in Sudbury, MA. I placed the rover in the back of a
Chapter 5: Navigation Performance

pickup truck and drove through the woods on the reservation. A map of the base station and rover positions is shown in Figure 5.20.

Figure 5.20: Map of rover positions during an experiment conducted in a forest on 6 February 2002.

In Figure 5.20, the solid black squares and the black circles mark the AM and GPS rover position estimates, respectively. Corresponding AM and GPS rover position estimates are connected by purple lines. The location of the base receiver is marked “Base” and the location of an overhead power line is marked “OHP”. The rover moved along each road twice: once to move toward a dead end and once to move back. The roads traversed are very narrow one-lane four-wheel-drive roads with trees on both sides. Because the roads are so narrow, the position estimates from both passes along the roads should have been inline. Figure 5.20 shows that the AM position estimates, in general, were more consistent than the GPS position estimates in this regard. This was especially true in the middle of the map where the road runs east to west. Here the GPS position estimates from both passes differed considerably while the AM position estimates from both passes were indistinguishable. The long purple lines near the “OHP” waypoint show that when the rover was near the overhead power line, the errors in the AM position estimates increased dramatically. The position differences for this experiment are shown in Figure 5.21.
Figure 5.21: Difference between AM and GPS position estimates during an experiment conducted in a forest on 6 February 2002.

Figure 5.21 shows that the position differences were generally smaller than 20 meters but they were unmistakably larger than the position differences from the Hansom Air Force Base experiment shown in Figure 5.4. The larger position differences in this experiment were due to an increase in the GPS positioning errors. The L-band GPS signals are blocked and reflected by trees, which leads to signal loss and multipath. The inconsistency of the GPS position estimates, shown in Figure 5.20, illustrates the degraded performance of GPS in the woods. In summer, when the leaves are on the trees, GPS performance would be even worse.

A histogram of position differences is shown in Figure 5.22.
The histogram in Figure 5.22 shows that there was a rather even distribution of errors up to 20 meters. These relatively large position differences for an experiment with such short baselines were caused by an increase in GPS positioning errors. The median position difference in this experiment was 12.1 meters and the 95th percentile was 25.8 meters. In the Hanscom Air Force Base experiment, which had baselines longer than those in this experiment, the median was only 6.2 meters and the 95th percentile was only 20.8 meters.

The ambiguity function value for this experiment is shown in Figure 5.23.
5.3 Forest Experiment

The ambiguity function value for this experiment, shown in Figure 5.23, was extraordinarily high. The only two drops in the ambiguity function value correspond to the rover passing under the overhead power line. Over 98% of the epochs had an ambiguity function value greater than 0.95. Among these epochs, the median position difference was 12.0 meters and the 95th percentile was 24.8 meters. Because so few epochs had ambiguity function values less than 0.95, the median and 95th percentile were virtually unchanged. The generally high ambiguity function values in this experiment provide further evidence that GPS position errors were larger than the AM position errors.

The results from this experiment show that AM navigation is virtually unaffected by foliage. In fact, the extraordinarily high ambiguity function value in this experiment suggests that AM navigation may improve in the woods. The improvement is easily explained by the fact that forests have fewer conductors in them than developed areas. GPS positioning performance, on the other hand, is much worse in the woods. Even when there are no leaves on the trees, as was the case for this experiment, GPS positioning performance was noticeably worse in the woods than in the open. In the woods, Garmin GPS receivers, like the one used in this experiment, track satellite signals better than most other receivers. However, while hiking in the woods in the summer with my personal Garmin GPS III Plus receiver, I often become frustrated with the performance of the receiver. It often will not track enough satellites to provide position estimates. An AM navigation system would solve these problems.
5.4 Indoor Experiments

I tested the AM navigation system inside two suburban wood-frame houses. Unfortunately, the results from both experiments show that indoor AM positioning is inaccurate and sometimes results in ambiguity resolution failure. In sections 4.4 and 4.5, I showed that proximate overhead and underground conductors, especially ones that are part of vast interconnected grids, perturb the phases and amplitudes of AM signals. Since the wiring and metal plumbing inside a house is electrically connected to vast external conductor grids, it is not surprising that similar phase and amplitude perturbations occur inside houses and buildings. Fortunately, the results from both experiments show that phase perturbations for a given indoor location are constant over time. Since the phase perturbations are constant, calibration of an indoor environment may be possible.

5.4.1 Belmont, MA Wood-Frame House Experiment

I conducted the first indoor experiment in a wood-frame house in Belmont, MA on 9 May 2001. The base station was placed at a GPS-surveyed location (WGS-84: N42.4086 W71.18851) in the driveway of the house. I slaved the sample clock of the rover to the sample clock of the base station during this experiment. I slowly moved the rover along a straight line from the driveway into the middle of the open garage of the house and then back out along the same line. The largest baseline in this experiment was about 15 meters. Because GPS does not work indoors, I used a tape measure to independently determine the position of the rover. Since the tape measurements are very accurate, the AM positioning error can be determined by computing the distance between the tape measure positions and the AM position estimates. The AM positioning errors for this experiment are shown in Figure 5.24.
The AM position errors, shown in Figure 5.24, were very large for such a short baseline experiment. Between when the rover entered the garage at 6 minutes elapsed time and when the rover left the garage at 12 minutes elapsed time, the position error was over 50 meters. Even when the rover was a little over one meter outside the garage at 4 and 14 minutes elapsed time, the position error was about 15 meters. Both on the way in and out, when the rover was very near the garage door opening the position error jumped up to over 300 meters. This happened because ambiguity resolution failed. As a result, the position estimates were quite arbitrary and the errors could have actually been much larger if the ambiguity function search space was expanded. The ambiguity function value for this experiment is shown in Figure 5.25.
The drop in the ambiguity function value, shown in Figure 5.25, was highly correlated with the large rise in the position errors, shown in Figure 5.24. The ambiguity function values were very low when the rover was inside the garage. Therefore, it is not surprising that ambiguity resolution failed when the rover was near the garage door opening.

The true position of the rover was known because it was measured with a tape measure. The receiver clock offset was also known because the rover sample clock was slaved to the base station sample clock. Since the base station was in the open on the driveway, the base station phase measurements were much more accurate than the rover phase measurements. Since all three searched-over variables (two position, one clock) were known and the base station phase measurements were quite good, the rover phase measurement error for each radio station can be computed. The rover phase measurement error during this experiment is shown in Figure 5.26.

Figure 5.25: Ambiguity function value during an experiment conducted in the garage of a wood-frame house in Belmont, MA on 9 May 2001.
5.4 Indoor Experiments

Figure 5.26: Rover phase error during an experiment conducted in the garage of a wood-frame house in Belmont, MA on 9 May 2001.

Figure 5.26 shows that the rover phase errors for a large number of radio stations were very large when the rover was inside the garage. Clearly the conductors inside even a wood-frame house conspire to produce very large phase measurement errors. Interestingly, the phase errors, though large, were constant for a given location during the short time interval between the rover pass into and out of the garage. The motion of the rover is exactly symmetric over time: the rover moved out of the garage at the same rate that the rover moved into the garage. Since the phase errors, shown in Figure 5.26, were also symmetric over time, the phase errors for a given location were constant over the short time interval between the trip into and out of the garage. Constant phase errors over time suggest that an indoor environment could be calibrated: the phase perturbations could be determined prior to when navigation services are required. The perturbation information could then be used to reduce errors during subsequent navigation.

5.4.2 Sudbury, MA Wood-Frame House Experiment

I also conducted an experiment inside a wood-frame house in Sudbury, MA on 7 June 2001. The description of the Belmont, MA indoor experiment above applies equally well to this experiment except the base station was, of course, placed at a different GPS-surveyed location (WGS-84: N42.38470 W71.44387). The AM positioning errors for this experiment are shown in Figure 5.27.
As with the Belmont experiment, the AM position errors were very large when the rover was inside the garage between 4.5 and 10.25 minutes elapsed time. In this experiment, however, the position errors suggest that ambiguity resolution failed much more often than before. The position errors when the rover was inside the garage were bimodal: they jumped back and forth between about 100 meters and about 220 meters. This suggests that there were two peaks of the ambiguity function in the search space that had very similar values. The ambiguity function value for this experiment is shown in Figure 5.28.
The ambiguity function values, shown in Figure 5.28, were surprisingly high when the rover was inside the garage considering that ambiguity resolution failure occurred so frequently during this experiment. Ambiguity resolution failure only occurred during the Belmont experiment when the ambiguity function value dropped below 0.65. Ambiguity resolution failure for such high values of the ambiguity function occurs because of poor geometry. Sudbury is west of almost all of the radio stations in the database of radio stations I use for navigation. Belmont, on the other hand, has many stations in all directions. Therefore, the ambiguity function method undoubtedly has more geometric leverage on the ambiguity resolution problem in Belmont than it does in Sudbury.

The rover phase errors during this experiment are shown in Figure 5.29.
The rover phase errors were smaller during this experiment than the phase errors that occurred during the Belmont experiment. This is not surprising since the ambiguity function value was larger during this experiment. As before, the symmetry of the phase errors suggests that phase errors for a given indoor location are constant over time.

5.5 Concluding Remarks

Outside in the open, AM positioning errors are less than 10 meters 90% of the time. In the woods, AM positioning performance is no different than it is in the open. Large positioning errors sometimes occur when the rover is near overhead power lines or under overpasses. When these errors occur near overhead conductors, the ambiguity function value is usually very low so poor positioning performance can be predicted. When these errors occur while the rover is under an overpass, the ambiguity function value often remains high, but the observed amplitude drops considerably for all stations so poor positioning performance can still be predicted. Near tall buildings and underground conductors in urban areas and inside wood-frame houses, positioning performance is poor, often resulting in ambiguity-resolution failure.
Chapter 6: Conclusions and Future Work

6.1 Conclusions

The most significant conclusion from this work is that instantaneous positioning is possible using observations of AM carrier-phases. Previous carrier-phase signal-of-opportunity systems only tracked the motion of a user from a known starting point. Using an ambiguity function method, I showed that the phase ambiguities can be resolved using measurements from a single epoch.

I also demonstrated that a navigation receiver can be implemented as a software radio. The navigation receivers that I designed are unique in that they digitize the entire AM band without downconverting. This design allows for simultaneous reception of all available AM signals and eliminates the problem of interchannel bias that a multichannel receiver has. The software radio design is also quite flexible because all radio functions are implemented in software.

AM radio stations that use multiple-tower directional antenna systems can still be used in an AM radiolocation system despite their large azimuthal phase variation. I showed that a simple far-field approximation can be used effectively to model the phase even when the local ground properties are not known.

A groundwave model that treats the ground as homogeneous can be used to effectively reduce measurement errors over a large geographical area. I developed a groundwave model using NEC-4 simulation results and experimental results. I showed that the use of this model reduces positioning errors significantly.

Overhead and underground conductors pose significant challenges to AM navigation. Since most overhead and underground conductors form vast interconnected networks, they distort the AM fields around them more so than otherwise might be expected. In some cases, overhead conductors were even shown to cause ambiguity-resolution failure.

Nighttime AM radiolocation is possible but is less accurate and ambiguity resolution is less robust because of skywave propagation. An indirect consequence of skywave is that fewer radio station signals are useful for nighttime navigation because of FCC regulations. A direct consequence of skywave is that reflected signals interfere with the desired direct groundwave signals and cause unpredictable phase variation.

The positioning performance of the AM navigation system depends heavily on the electrical environment of the receivers. Outside in the open, positioning errors are less than seven meters for several-kilometer baselines 95% of the time. Positioning errors are
less than 15 meters for baselines up to 35 kilometers 95% of the time. Operation in wooded areas does not affect positioning performance. AM positioning indoors and near tall buildings and underground conductors in urban environments is subject to large errors and frequent ambiguity resolution failure. However, indoor and urban errors at a given location are constant over time so calibration may be possible.

6.2 Future Work

The radiolocation system described in this thesis was designed to be a proof-of-concept system. To make the system broadly useful, many enhancements are needed. The system should be more portable and a radio data link is needed to provide real-time navigation services. In addition to these engineering tasks, there are number of research opportunities that are worth pursuing. I discuss a few of these opportunities below.

6.2.1 Underwater Navigation

The skin depth of AM signals in fresh water is about 10 meters so navigation may be possible down to 30 meters or so. The skin depth in sea water is only about 20 centimeters. This may at first seem too shallow to be useful, but in some military applications, navigating with an antenna slightly below the surface may be far more attractive than navigating with an antenna slightly above the surface.

6.2.2 Underground Navigation

Navigation in mines and caves may also be possible. Navigation in mines may be difficult due to the wiring that is often installed in them. In undeveloped caves, however, no such wiring exists and navigation performance could be quite good. If navigation performance in caves is in fact good, spelunkers would undoubtedly find it very useful.

6.2.3 Indoor Calibration

The results presented in section 5.4 show that the AM phase perturbations inside houses, although large, are constant over time. Therefore, it should be possible to calibrate an indoor environment for future navigation. Further investigation of the errors inside houses is needed to determine if calibration is practical and to determine what positioning accuracy can be achieved with such calibration.

6.2.4 Outdoor Calibration

Calibration may also be useful outdoors. The results presented in section 5.2.4 show that positioning performance in urban environments is quite poor. However, since buildings
and underground conductors do not change much, it may be possible to calibrate an urban environment to improve positioning performance. Also, calibration may be useful for minimizing the effects of propagation uncertainty.

6.2.5 Transmitter Positioning

The system described in this thesis is designed to determine the position of a roving receiver. However, it is also possible to reverse the problem and determine the positions, and even the patterns, of the transmitting antennas. This exercise would require the rover to be positioned independently, probably using DGPS. Successful transmitter positioning would have profound effects on the practicality of an AM navigation system. If the transmitter antenna positions of radios stations in a large geographic area could be determined accurately by just observing the AM signals, radiolocation service could be established without the laborious task of surveying the antennas. If the base station and rover were in aircraft, the transmitter antenna patterns and positions could be determined in a matter of minutes.

The ability to locate transmitters would also be useful for regulatory and military purposes. The FCC could locate illegal transmitters using this technology. Also, the military could locate transmitters to aid surveillance activities.

6.2.6 Sensing

The results in this thesis show that AM signals are sensitive to many things including ground properties and conductor proximity. Where practical I sought to minimize these effects in order to maximize positioning performance. It is not difficult, however, to imagine applications that could benefit from this sensitivity. For instance, AM observations may be useful for geological sensing or aid in locating underground conductors. Also, the FCC could use the technology to determine if directional antennas are performing as licensed. Undoubtedly, many other applications exist. One man’s noise is another man’s signal.

6.2.7 H-field Antenna

The one-meter vertical whip I used in the system described in this thesis is an omnidirectional E-field antenna. Unfortunately, the performance of an E-field antenna in the AM band is as dependent on the ground system, i.e., on its counterpoise, as it is on the antenna. This is usually not a problem in a car where a decent ground is available. However, in a small handheld device, an E-field antenna is less practical. An attractive alternative is an H-field antenna. It is no accident that most portable AM radios use H-field antennas: they are cheap and compact. Unlike the ferrite magnetic dipoles that are typically found in portable AM radios, an H-field antenna for navigation should be
omnidirectional. This may require three magnetic dipoles mounted orthogonally. Because of the directionality of the individual magnetic dipoles, this antenna topology could also yield antenna orientation. It would also be interesting to investigate the use of both E-field and H-field antennas. Measuring both the electric and magnetic fields might be useful in sensing applications.
Appendix A: Boston-Area AM Radio Stations

In this appendix, I provide Boston-area radio station data.

A.1 Radio-Station Power

Table A.1 shows the number of antenna towers that each station has, and the radiated power for daytime, critical hours, and nighttime operation. Critical hours are the two hours before local sunset and the two hours after local sunrise. The data are from the FCC database. In the table, I indicate that some stations that have very low nighttime power have zero nighttime power because the signals from these stations are too weak to use at night.
### Appendix A: Boston-Area AM Radio Stations

<table>
<thead>
<tr>
<th>Frequency (kHz)</th>
<th>Number of towers</th>
<th>Daytime Watts</th>
<th>Critical hours Watts</th>
<th>Nighttime Watts</th>
</tr>
</thead>
<tbody>
<tr>
<td>590</td>
<td>2</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
</tr>
<tr>
<td>610</td>
<td>4</td>
<td>5000</td>
<td>5000</td>
<td>1000</td>
</tr>
<tr>
<td>650</td>
<td>2</td>
<td>250</td>
<td>250</td>
<td>0</td>
</tr>
<tr>
<td>680</td>
<td>3</td>
<td>50000</td>
<td>50000</td>
<td>50000</td>
</tr>
<tr>
<td>740</td>
<td>1</td>
<td>250</td>
<td>250</td>
<td>0</td>
</tr>
<tr>
<td>800</td>
<td>1</td>
<td>1000</td>
<td>1000</td>
<td>0</td>
</tr>
<tr>
<td>850</td>
<td>3</td>
<td>50000</td>
<td>50000</td>
<td>50000</td>
</tr>
<tr>
<td>890</td>
<td>5</td>
<td>25000</td>
<td>25000</td>
<td>3400</td>
</tr>
<tr>
<td>900</td>
<td>1</td>
<td>910</td>
<td>910</td>
<td>0</td>
</tr>
<tr>
<td>950</td>
<td>1</td>
<td>5000</td>
<td>5000</td>
<td>0</td>
</tr>
<tr>
<td>980</td>
<td>4</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>1000</td>
<td>1000</td>
<td>0</td>
</tr>
<tr>
<td>1030</td>
<td>2</td>
<td>50000</td>
<td>50000</td>
<td>50000</td>
</tr>
<tr>
<td>1060</td>
<td>2</td>
<td>42000</td>
<td>22000</td>
<td>0</td>
</tr>
<tr>
<td>1090</td>
<td>1</td>
<td>5000</td>
<td>1000</td>
<td>0</td>
</tr>
<tr>
<td>1120</td>
<td>4</td>
<td>5000</td>
<td>5000</td>
<td>1000</td>
</tr>
<tr>
<td>1150</td>
<td>3</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
</tr>
<tr>
<td>1200</td>
<td>2</td>
<td>10000</td>
<td>10000</td>
<td>1000</td>
</tr>
<tr>
<td>1260</td>
<td>3</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
</tr>
<tr>
<td>1280</td>
<td>3</td>
<td>5000</td>
<td>5000</td>
<td>1000</td>
</tr>
<tr>
<td>1300</td>
<td>1</td>
<td>1000</td>
<td>1000</td>
<td>0</td>
</tr>
<tr>
<td>1330</td>
<td>2</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
</tr>
<tr>
<td>1360</td>
<td>1</td>
<td>700</td>
<td>700</td>
<td>0</td>
</tr>
<tr>
<td>1370</td>
<td>2</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
</tr>
<tr>
<td>1430</td>
<td>2</td>
<td>5000</td>
<td>5000</td>
<td>1000</td>
</tr>
<tr>
<td>1460</td>
<td>2</td>
<td>5000</td>
<td>5000</td>
<td>1000</td>
</tr>
<tr>
<td>1510</td>
<td>4</td>
<td>50000</td>
<td>50000</td>
<td>50000</td>
</tr>
<tr>
<td>1550</td>
<td>1</td>
<td>10000</td>
<td>10000</td>
<td>0</td>
</tr>
<tr>
<td>1600</td>
<td>2</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
</tr>
</tbody>
</table>

**Table A.1:** Daytime, critical hours, and nighttime power levels for Boston-area radio stations.

### A.2 Antenna Coordinates

In Table A.2 and Table A.3, I show tower heights and WGS-84 coordinates for Boston-area radio stations. The tower heights are from the FCC database [10]. If the FCC database does not list the tower height for a particular radio tower, the field is left blank in the tables. As described in section 4.1.2, I conducted geodetic surveys of the radio towers to produce the coordinates listed in the tables. The antenna for the DGPS base station used during the surveys was located on top of the easternmost staircase of the South Lab building at Lincoln Laboratory. On top of this staircase, there are seven antenna posts mounted to a horizontally mounted 6x6 board. The DGPS base station antenna was located on the northernmost of these antenna posts. The assumed WGS-84 coordinates for this post were: N42.4594972, W71.2651564. The antenna was connected to the receiver, which was located in South Lab room number S2-522, via a semi-rigid low-loss coax cable.
### A.2 Antenna Coordinates

<table>
<thead>
<tr>
<th>Frequency (kHz)</th>
<th>Tower number</th>
<th>Tower height (degrees)</th>
<th>Latitude (degrees)</th>
<th>Longitude (degrees)</th>
</tr>
</thead>
<tbody>
<tr>
<td>590</td>
<td>1</td>
<td>75.6</td>
<td>42.4064097</td>
<td>-71.0880324</td>
</tr>
<tr>
<td>590</td>
<td>2</td>
<td>75.6</td>
<td>42.4064103</td>
<td>-71.0852920</td>
</tr>
<tr>
<td>610</td>
<td>1</td>
<td>75.6</td>
<td>43.0166931</td>
<td>-71.4800023</td>
</tr>
<tr>
<td>610</td>
<td>2</td>
<td></td>
<td>43.0161548</td>
<td>-71.4798343</td>
</tr>
<tr>
<td>610</td>
<td>3</td>
<td></td>
<td>43.0156148</td>
<td>-71.4796647</td>
</tr>
<tr>
<td>610</td>
<td>4</td>
<td></td>
<td>43.0150754</td>
<td>-71.4794966</td>
</tr>
<tr>
<td>650</td>
<td>1</td>
<td>115.8</td>
<td>42.2874487</td>
<td>-71.4314535</td>
</tr>
<tr>
<td>650</td>
<td>2</td>
<td>115.8</td>
<td>42.2878535</td>
<td>-71.4310714</td>
</tr>
<tr>
<td>680</td>
<td>1</td>
<td>108</td>
<td>42.4899178</td>
<td>-71.2200032</td>
</tr>
<tr>
<td>680</td>
<td>2</td>
<td>97</td>
<td>42.4903588</td>
<td>-71.2177714</td>
</tr>
<tr>
<td>680</td>
<td>3</td>
<td>108</td>
<td>42.4908024</td>
<td>-71.2155409</td>
</tr>
<tr>
<td>740</td>
<td>1</td>
<td>75.8</td>
<td>42.3867504</td>
<td>-71.1387838</td>
</tr>
<tr>
<td>800</td>
<td>1</td>
<td>117.1</td>
<td>42.6741118</td>
<td>-71.1902725</td>
</tr>
<tr>
<td>850</td>
<td>1</td>
<td>207</td>
<td>42.2777037</td>
<td>-71.2679258</td>
</tr>
<tr>
<td>850</td>
<td>2</td>
<td>207</td>
<td>42.2778737</td>
<td>-71.2665272</td>
</tr>
<tr>
<td>850</td>
<td>3</td>
<td>207</td>
<td>42.2780420</td>
<td>-71.2651214</td>
</tr>
<tr>
<td>890</td>
<td>1</td>
<td>173.6</td>
<td>42.2471508</td>
<td>-71.4260712</td>
</tr>
<tr>
<td>890</td>
<td>2</td>
<td>173.6</td>
<td>42.2471465</td>
<td>-71.4251197</td>
</tr>
<tr>
<td>890</td>
<td>3</td>
<td>173.6</td>
<td>42.2471488</td>
<td>-71.4241608</td>
</tr>
<tr>
<td>890</td>
<td>4</td>
<td>173.6</td>
<td>42.2471495</td>
<td>-71.4232045</td>
</tr>
<tr>
<td>890</td>
<td>5</td>
<td>173.6</td>
<td>42.2471495</td>
<td>-71.4222538</td>
</tr>
<tr>
<td>900</td>
<td>1</td>
<td>62.7</td>
<td>42.7594827</td>
<td>-71.4765780</td>
</tr>
<tr>
<td>950</td>
<td>1</td>
<td>94.2</td>
<td>42.4357372</td>
<td>-70.9923422</td>
</tr>
<tr>
<td>980</td>
<td>1</td>
<td>90</td>
<td>42.6546905</td>
<td>-71.3630862</td>
</tr>
<tr>
<td>980</td>
<td>2</td>
<td>90</td>
<td>42.6543652</td>
<td>-71.3622615</td>
</tr>
<tr>
<td>980</td>
<td>3</td>
<td>90</td>
<td>42.6547279</td>
<td>-71.3615593</td>
</tr>
<tr>
<td>980</td>
<td>4</td>
<td>90</td>
<td>42.6540439</td>
<td>-71.3614447</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>90</td>
<td>42.5236866</td>
<td>-71.7344450</td>
</tr>
<tr>
<td>1030</td>
<td>1</td>
<td>188.5</td>
<td>42.2789580</td>
<td>-70.8761792</td>
</tr>
<tr>
<td>1030</td>
<td>2</td>
<td>188.5</td>
<td>42.2789632</td>
<td>-70.8752583</td>
</tr>
<tr>
<td>1060</td>
<td>1</td>
<td>122</td>
<td>42.2874487</td>
<td>-71.4314535</td>
</tr>
<tr>
<td>1060</td>
<td>2</td>
<td>122</td>
<td>42.2878535</td>
<td>-71.4310714</td>
</tr>
<tr>
<td>1090</td>
<td>1</td>
<td>101.3</td>
<td>42.4105810</td>
<td>-71.0751774</td>
</tr>
</tbody>
</table>

Table A.2: FCC-database antenna tower heights and surveyed WGS-84 coordinates for Boston-area radio stations 590 through 1090.
## Appendix A: Boston-Area AM Radio Stations

<table>
<thead>
<tr>
<th>Frequency (kHz)</th>
<th>Tower number</th>
<th>Tower height (degrees)</th>
<th>WGS-84 coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>1120</td>
<td>1</td>
<td>80.4</td>
<td>42.4481709</td>
</tr>
<tr>
<td>1120</td>
<td>2</td>
<td>80.4</td>
<td>42.4481723</td>
</tr>
<tr>
<td>1120</td>
<td>3</td>
<td>80.4</td>
<td>42.4481740</td>
</tr>
<tr>
<td>1120</td>
<td>4</td>
<td>80.4</td>
<td>42.491431</td>
</tr>
<tr>
<td>1150</td>
<td>1</td>
<td>114</td>
<td>42.4138629</td>
</tr>
<tr>
<td>1150</td>
<td>2</td>
<td>114</td>
<td>42.4134388</td>
</tr>
<tr>
<td>1150</td>
<td>3</td>
<td>114</td>
<td>42.4130217</td>
</tr>
<tr>
<td>1200</td>
<td>1</td>
<td></td>
<td>42.2874487</td>
</tr>
<tr>
<td>1200</td>
<td>2</td>
<td></td>
<td>42.2878535</td>
</tr>
<tr>
<td>1260</td>
<td>1</td>
<td>92.2</td>
<td>42.2743261</td>
</tr>
<tr>
<td>1260</td>
<td>2</td>
<td>92.2</td>
<td>42.2736152</td>
</tr>
<tr>
<td>1260</td>
<td>3</td>
<td>92.2</td>
<td>42.2750367</td>
</tr>
<tr>
<td>1280</td>
<td>1</td>
<td>93.5</td>
<td>42.5944866</td>
</tr>
<tr>
<td>1280</td>
<td>2</td>
<td>93.5</td>
<td>42.5943098</td>
</tr>
<tr>
<td>1300</td>
<td>1</td>
<td>96.1</td>
<td>42.2598229</td>
</tr>
<tr>
<td>1330</td>
<td>1</td>
<td></td>
<td>42.3543447</td>
</tr>
<tr>
<td>1330</td>
<td>2</td>
<td></td>
<td>42.3546222</td>
</tr>
<tr>
<td>1360</td>
<td>1</td>
<td>189.1</td>
<td>42.4529879</td>
</tr>
<tr>
<td>1370</td>
<td>1</td>
<td>99.7</td>
<td>42.9074278</td>
</tr>
<tr>
<td>1370</td>
<td>2</td>
<td>175.4</td>
<td>42.9074972</td>
</tr>
<tr>
<td>1430</td>
<td>1</td>
<td>102.5</td>
<td>42.4030294</td>
</tr>
<tr>
<td>1430</td>
<td>2</td>
<td>102.5</td>
<td>42.4032609</td>
</tr>
<tr>
<td>1460</td>
<td>1</td>
<td>122</td>
<td>42.0482976</td>
</tr>
<tr>
<td>1460</td>
<td>2</td>
<td>156</td>
<td>42.0482974</td>
</tr>
<tr>
<td>1510</td>
<td>1</td>
<td>199</td>
<td>42.3862357</td>
</tr>
<tr>
<td>1510</td>
<td>2</td>
<td>199</td>
<td>42.3859981</td>
</tr>
<tr>
<td>1510</td>
<td>3</td>
<td>199</td>
<td>42.3864516</td>
</tr>
<tr>
<td>1510</td>
<td>4</td>
<td>199</td>
<td>42.3862154</td>
</tr>
<tr>
<td>1550</td>
<td>1</td>
<td></td>
<td>42.3575484</td>
</tr>
<tr>
<td>1600</td>
<td>1</td>
<td>205</td>
<td>42.2886949</td>
</tr>
<tr>
<td>1600</td>
<td>2</td>
<td>205</td>
<td>42.2888635</td>
</tr>
</tbody>
</table>

Table A.3: FCC-database antenna tower heights and surveyed WGS-84 coordinates for Boston-area radio stations 1120 through 1600.

### A.3 Antenna Field and Phase

Table A.4, Table A.5, and Table A.6 show the daytime, critical hours, and nighttime, respectively, field and phase data for Boston-area radio stations. Only stations that have different daytime and critical hours patterns are listed in Table A.5. The data are from the FCC database [10]. Sometimes the FCC database is ambiguous about which field and phase go with which tower. In these cases, I could usually solve the ambiguity problem experimentally. So despite some ambiguity in the FCC database, the field and phase data should be listed for the correct towers in the following tables.
A.3 Antenna Field and Phase

<table>
<thead>
<tr>
<th>Frequency (kHz)</th>
<th>Tower number</th>
<th>Field</th>
<th>Phase (degrees)</th>
<th>Frequency (kHz)</th>
<th>Tower number</th>
<th>Field</th>
<th>Phase (degrees)</th>
</tr>
</thead>
<tbody>
<tr>
<td>590</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1120</td>
<td>1</td>
<td>0.4</td>
<td>-176.2</td>
</tr>
<tr>
<td>590</td>
<td>2</td>
<td>1.449</td>
<td>-30</td>
<td>1120</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>610</td>
<td>1</td>
<td>0.28</td>
<td>65</td>
<td>1120</td>
<td>3</td>
<td>0.47</td>
<td>-115.1</td>
</tr>
<tr>
<td>610</td>
<td>2</td>
<td>0.55</td>
<td>200</td>
<td>1120</td>
<td>4</td>
<td>0.73</td>
<td>-5.8</td>
</tr>
<tr>
<td>610</td>
<td>3</td>
<td>0.55</td>
<td>220</td>
<td>1150</td>
<td>1</td>
<td>0.515</td>
<td>166</td>
</tr>
<tr>
<td>610</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1150</td>
<td>2</td>
<td>1.03</td>
<td>14</td>
</tr>
<tr>
<td>650</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1150</td>
<td>3</td>
<td>0.515</td>
<td>-166</td>
</tr>
<tr>
<td>650</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1200</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>680</td>
<td>1</td>
<td>0.5</td>
<td>104</td>
<td>1200</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>680</td>
<td>2</td>
<td>0.9</td>
<td>40</td>
<td>1260</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>680</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1260</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>740</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1260</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>800</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1280</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>850</td>
<td>1</td>
<td>1.27</td>
<td>88.9</td>
<td>1280</td>
<td>2</td>
<td>0.504</td>
<td>96</td>
</tr>
<tr>
<td>850</td>
<td>2</td>
<td>1.95</td>
<td>0</td>
<td>1280</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>850</td>
<td>3</td>
<td>1</td>
<td>-41.1</td>
<td>1300</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>890</td>
<td>1</td>
<td>1</td>
<td>316</td>
<td>1330</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>890</td>
<td>2</td>
<td>3.067</td>
<td>156.7</td>
<td>1330</td>
<td>2</td>
<td>0.8</td>
<td>-135</td>
</tr>
<tr>
<td>890</td>
<td>3</td>
<td>4.135</td>
<td>0</td>
<td>1360</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>890</td>
<td>4</td>
<td>2.968</td>
<td>-154.9</td>
<td>1370</td>
<td>1</td>
<td>0.65</td>
<td>-10</td>
</tr>
<tr>
<td>890</td>
<td>5</td>
<td>0.986</td>
<td>-310</td>
<td>1370</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>900</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1430</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>950</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1430</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>980</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1460</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>980</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1460</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>980</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1510</td>
<td>1</td>
<td>1.16</td>
<td>0</td>
</tr>
<tr>
<td>980</td>
<td>4</td>
<td>0.28</td>
<td>114</td>
<td>1510</td>
<td>2</td>
<td>0</td>
<td>92.1</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1510</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1030</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1510</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1030</td>
<td>2</td>
<td>1</td>
<td>86.4</td>
<td>1550</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1060</td>
<td>1</td>
<td>0.874</td>
<td>111</td>
<td>1600</td>
<td>1</td>
<td>1.11</td>
<td>90</td>
</tr>
<tr>
<td>1060</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1600</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1090</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A.4: Daytime antenna field and phase for Boston-area radio stations.

<table>
<thead>
<tr>
<th>Frequency (kHz)</th>
<th>Tower number</th>
<th>Field</th>
<th>Phase (degrees)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1060</td>
<td>1</td>
<td>0.96</td>
<td>123</td>
</tr>
<tr>
<td>1060</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1090</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table A.5: Critical-hours antenna field and phase for Boston-area radio stations.
## Appendix A: Boston-Area AM Radio Stations

<table>
<thead>
<tr>
<th>Frequency (kHz)</th>
<th>Tower number</th>
<th>Field</th>
<th>Phase (degrees)</th>
<th>Frequency (kHz)</th>
<th>Tower number</th>
<th>Field</th>
<th>Phase (degrees)</th>
</tr>
</thead>
<tbody>
<tr>
<td>590</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1150</td>
<td>1</td>
<td>0.568</td>
<td>149</td>
</tr>
<tr>
<td>590</td>
<td>2</td>
<td>1.449</td>
<td>-30</td>
<td>1150</td>
<td>2</td>
<td>1</td>
<td>2.5</td>
</tr>
<tr>
<td>610</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1150</td>
<td>3</td>
<td>0.568</td>
<td>-149</td>
</tr>
<tr>
<td>610</td>
<td>2</td>
<td>0.839</td>
<td>191.5</td>
<td>1200</td>
<td>1</td>
<td>0.8</td>
<td>115</td>
</tr>
<tr>
<td>610</td>
<td>3</td>
<td>0.839</td>
<td>-191.5</td>
<td>1200</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>680</td>
<td>1</td>
<td>1</td>
<td>104</td>
<td>1260</td>
<td>1</td>
<td>1</td>
<td>-11.5</td>
</tr>
<tr>
<td>680</td>
<td>2</td>
<td>1.904</td>
<td>55.8</td>
<td>1260</td>
<td>2</td>
<td>0.574</td>
<td>135.5</td>
</tr>
<tr>
<td>680</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1280</td>
<td>1</td>
<td>1</td>
<td>128.5</td>
</tr>
<tr>
<td>850</td>
<td>1</td>
<td>1</td>
<td>75</td>
<td>1280</td>
<td>2</td>
<td>1.89</td>
<td>0</td>
</tr>
<tr>
<td>850</td>
<td>2</td>
<td>1.97</td>
<td>0</td>
<td>1280</td>
<td>3</td>
<td>1</td>
<td>-128.5</td>
</tr>
<tr>
<td>850</td>
<td>3</td>
<td>1</td>
<td>-75</td>
<td>1330</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>890</td>
<td>1</td>
<td>0.25</td>
<td>-100.2</td>
<td>1330</td>
<td>2</td>
<td>0.95</td>
<td>-110.5</td>
</tr>
<tr>
<td>890</td>
<td>2</td>
<td>0.712</td>
<td>130.5</td>
<td>1370</td>
<td>1</td>
<td>0.885</td>
<td>-3.8</td>
</tr>
<tr>
<td>890</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1370</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>890</td>
<td>4</td>
<td>0.724</td>
<td>-128.6</td>
<td>1430</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>890</td>
<td>5</td>
<td>0.237</td>
<td>101.5</td>
<td>1430</td>
<td>2</td>
<td>0.85</td>
<td>-113.8</td>
</tr>
<tr>
<td>980</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1460</td>
<td>1</td>
<td>0.88</td>
<td>110</td>
</tr>
<tr>
<td>980</td>
<td>2</td>
<td>1.51</td>
<td>-125.1</td>
<td>1460</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>980</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1510</td>
<td>1</td>
<td>1.56</td>
<td>-137.1</td>
</tr>
<tr>
<td>980</td>
<td>4</td>
<td>0.65</td>
<td>112.2</td>
<td>1510</td>
<td>2</td>
<td>0.99</td>
<td>100.9</td>
</tr>
<tr>
<td>1030</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1510</td>
<td>3</td>
<td>1.57</td>
<td>125.1</td>
</tr>
<tr>
<td>1030</td>
<td>2</td>
<td>1</td>
<td>86.4</td>
<td>1510</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1120</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1600</td>
<td>1</td>
<td>1.11</td>
<td>90</td>
</tr>
<tr>
<td>1120</td>
<td>2</td>
<td>1.62</td>
<td>-108.7</td>
<td>1600</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1120</td>
<td>3</td>
<td>0.7</td>
<td>141.6</td>
<td>1600</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1120</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A.6: Nighttime antenna field and phase for Boston-area radio stations.
Appendix B: Source Code

This appendix contains the C++ source code for the software radio program and the navigation algorithm program. Both programs were written using Microsoft Visual C++ and are intended to run on the Windows NT operating system.

B.1 Software Radio Program

The software radio program is the program that runs on the receiver computers. It implements the software radio and produces a data file that contains frequency, phase, and amplitude data for the AM radio station signals.

B.1.1 Header Files

////////////////////////////////////////////////////////////////////////
// board.h
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// Datel PCI-416 A/D board header file
////////////////////////////////////////////////////////////////////////

ifndef _BOARD_H
#define _BOARD_H

#include "windows.h"
#include "pci416df.h"

class Board {
public:
    Board(const WORD bn=0, const double sfsppm=50) : BoardNum(bn), SampleFreqStab(sfsppm){}

    DWORD mBoardNum() const {return BoardNum;}
    DWORD mDmaBufferSizeBytes() const {return DmaBufferSizeBytes;}
    int mGetADMMax() const;
    int mGetADMMin() const;

#endif
double mSampleFreqStab() const {return SampleFreqStab;}
double mScaleFactor() const {return ScaleFactor;}

DWORD mInitialize();

private:
    const WORD BoardNum;
    const double SampleFreqStab; // Stability of sample clock
    expressed as fraction of nominal frequency

    DWORD FifoSizeSamples;
    DWORD DmaBufferSizeBytes;
    DWORD ModelNumber;
    DWORD IirqEnabled; // 1=IRQ enabled, 0=IRQ disabled
    WORD ResolutionBits;
    WORD Channels;
    WORD ShortCycle;   // 1 = short cycle the A/D channels possible
    REALTYPE MaxSingleSampleRate;
    REALTYPE MaxScanSampleRate;
    double ScaleFactor;

    DWORD ErrorCode;
};

#endif

/////////////////////////////////////////////////
// comport.h
/////////////////////////////////////////////////
/////////////////////////////////////////////////
// Serial port header file
/////////////////////////////////////////////////

#ifdef __COMPORT_H
#define __COMPORT_H

#include "winbase.h"

class ComPort {
public:
    ComPort(){}
    ~ComPort(){
        int mPulseDTR(const int & pwms);

private:
    HANDLE HCom;
    BOOL PortReady;
    DCB ComDcb;
};

164
#ifndef _DC_H
#define _DC_H

#include <windows.h>
#include "sys\types.h"
#include "sys\timeb.h"

#include "board.h"
#include "comport.h"

class DataCollection {
public:
    DataCollection(const Board& b,
        const double tls = 10,
        const double tr = 0.2,
        const DWORD wls = 4194304, const double sr = 5.0e6,
        const WORD cs = 0, const WORD ts = 0, const WORD tm = 2,
        const WORD c = 0) : TheBoard(b),
        TotalLengthSec(tls), TriggerRate(tr), WindowLengthSamples(wls),
        SampleRate(sr), ClockSource(cs), TriggerSource(ts),
        TriggerMode(tm), Channel(c), BufNum(0), WindowCount(0),
        WindowNum(0),
        StartTime(0){}

    const double & mGetClockOffsetNeg() const {return
        ClockOffsetNeg;}
    const double & mGetClockOffsetPos() const {return
        ClockOffsetPos;}
    int mGetStartTime() const {return StartTime;}
    double mGetTriggerPeriod() const {return TriggerPeriod;}
    int mGetWindowNum() const {return WindowNum;}
    double mSampleRateMHz() const {return SampleRate/1.0e6;}
    DWORD mWindowLengthSamples() const {return WindowLengthSamples;}

    int mClockOffset(ComPort & cp, short *&pData, const int & pwms=50);
    int mExitLoudly();
    int mFixSignExt(short*& pData);
    int mInitialize();
    DWORD mNext(int& done);
    DWORD mStart();
Appendix B: Source Code

```cpp
DWORD mStop();
DWORD mWait(short*& pData);

private:
const Board& TheBoard;
double TotalLengthSec;
double TriggerRate; //Hz
double TriggerPeriod; //Seconds
DWORD WindowLengthSamples;
const double SampleRate;
WORD ClockSource;  // 0=internal
WORD TriggerSource; // 0=internal trigger source
const WORD TriggerMode;  // 2=multi-internal trigger
const WORD Channel;

DWORD BufNum;
double WindowLengthSec;
DWORD TotalLengthWindows;
DWORD WindowCount;
DWORD DoubleBufferSizeBytes[2];
DWORD DoubleBufferHandle[2];
DWORD TransferCountRem;

// Time stamp parameters
    time_t StartTime;
    int WindowNum;

    double ClockOffsetPos; // Offset in seconds to positive going pulse
    double ClockOffsetNeg; // Offset in seconds to negative going pulse

DWORD ErrorCode;

};
#endif
```

```cpp
/////////////////////////////////////////////////
// fe.h
/////////////////////////////////////////////////

#ifndef _FE_H
#define _FE_H

class FrontEnd {
public:
```
FrontEnd(const int pcd=249, const double pcb=5.0, const double hpc=0.01,
    const double lpc=1.705, const double pcs=1.0e-6) :
PhaseCalDivisor(pcd), PhaseCalBase(pcb),
    HighPassCutoff(hpc), LowPassCutoff(lpc), PhaseCalStab(pcs) {}

    const double mGetHighPassCutoff() const {return HighPassCutoff;}
    const double mGetLowPassCutoff() const {return LowPassCutoff;}
    const double mGetPhaseCalBase() const {return PhaseCalBase;}
    const int mGetPhaseCalDivisor() const {return PhaseCalDivisor;}
    const double mGetPhaseCalStab() const {return PhaseCalStab;}

private:
    const int PhaseCalDivisor;
    const double PhaseCalBase; // MHz
    const double HighPassCutoff; // MHz
    const double LowPassCutoff; // MHz
    const double PhaseCalStab; // PPM
};

#endif

// fftw.h

#include <vector>
#include <complex>
#include "rfftw_threads.h"

class Fftw {
public:
    Fftw() : NumSamp(1048576) {pFftTemp = new double[NumSamp];}
    Fftw(int ns) : NumSamp(ns) {pFftTemp = new double[NumSamp];}
    ~Fftw();

    int mFft(double *pData, std::vector<std::complex<double>> &fft);
    int mInitialize();
    int mMakeWisdom();

private:
    int NumSamp;
    double *pFftTemp;
#ifdef FFTW_PLAN
fftw_plan FftPlan;
#endif

#include <vector>

class HarmonicData {
public:
    HarmonicData(const FrontEnd& fe, const double scsppm=50.0);

    const std::vector<MeasSingle>& mGetData() const {return Data;}
    int mGetNumHarms() const {return NumHarms;}
    double mGetPCF() const {return PhaseCalFundamental;}

    int mCalc(NavData& nd);

private:
    int NumHarms;
    double PhaseCalFundamental;
    std::vector<MeasSingle> Data;
};

#include <vector>

#ifndef _HD_H
#define _HD_H

#include "fe.h"
#include "meassingle.h"
#include "navdata.h"

class HarmonicData {
public:
    HarmonicData(const FrontEnd& fe, const double scsppm=50.0);

    const std::vector<MeasSingle>& mGetData() const {return Data;}
    int mGetNumHarms() const {return NumHarms;}
    double mGetPCF() const {return PhaseCalFundamental;}

    int mCalc(NavData& nd);

private:
    int NumHarms;
    double PhaseCalFundamental;
    std::vector<MeasSingle> Data;
};

#endif

#ifndef _MATLABENG_H
#define _MATLABENG_H

#include <vector>

#endif
/include <complex>
#include <string>
#include "engine.h"

class MatlabEng {
public:
   MatlabEng();
~MatlabEng() {engClose(pEng);}

   int mHold();
   int mPlot(const std::vector<std::complex<double> > & y);
   int mPlot(const double * const pData, int NumElem);
   int mPlot(const std::vector<double>& x, const std::vector<double>& y, const int num=0);
   int mPlotFft(const std::vector<std::complex<double> > & data,
                const double srm=5.0,
                const std::string morp="Mag");

private:
   Engine *pEng;
};

#endif

/////////////////////////////////////////////////
// meassingle.h
/////////////////////////////////////////////////
/////////////////////////////////////////////////
// AM measurements header file
/////////////////////////////////////////////////
#ifndef _MEASSINGLE_H
#define _MEASSINGLE_H

class MeasSingle {
public:
   MeasSingle() : Id(0), NomFreq(0.0), Stab(0), Freq(0.0),
                  Phase(0.0), Mag(0.0) {}
   ~MeasSingle() {};

   double mGetFreq() const {return Freq;}
   int mGetId() const {return Id;}
   double mGetMag() const {return Mag;}
   double mGetNomFreq() const {return NomFreq;}
   double mGetPhase() const {return Phase;}
   double mGetStab() const {return Stab;}

   bool operator<(MeasSingle& rhs) {return (Id < rhs.Id);}

   int mInit(int id, double nf, double st);}
Appendix B: Source Code

```c
int mPrint();
int mSetFPM(double f, double p, double m);

private:
  int Id;
  double NomFreq;
  double Stab;
  double Freq;
  double Phase;
  double Mag;
);

#endif

/////////////////////////////////////////////////
// navdata.h
/////////////////////////////////////////////////

/////////////////////////////////////////////////
// Navigation data header file
/////////////////////////////////////////////////

#ifndef _NAVDATA_H
#define _NAVDATA_H

#include <vector>
#include <complex>
#include "board.h"
#include "dc.h"
#include "fftw.h"
#include "meassingle.h"

class MatlabEng;

class NavData {
public:
  NavData(const DataCollection& dc,const int hrm = 100);
  ~NavData();

  const double * const mGetDataPtr() const { return pData; } 
  const std::vector<std::complex<double> >& mGetFft() const { 
    return Fft; }
  const std::vector<std::complex<double> >& mGetHR() const { 
    return WindowHRFT; }
  const std::vector<std::complex<double> >& mGetXcorr() const { 
    return Xcorr; }
  int mMHztoI(const double freqmhz) const { return 
    mRound(freqmhz*MHztoI); }
  int mNumSamp() const { return NumSamp; }
  int mRound(const double doublein) const { return ((int)
    (doublein + 0.5)); }
```
double mSampleRateMHz() const { return SampleRateMHz; }
int mApplyWindow();
inline std::complex<double> mBwft(int nElem, double f, double fs);
int mCalcWindow();
int mGetFPM(MeasSingle& ms);
double mInterpolate(const std::vector<std::complex<double> >::iterator i);
int mLoadData(short *pRawData, const Board& ADBoard);
inline std::complex<double> mRwft(int nElem, double f, double fs);
int mTransform(Fftw & fftw, MatlabEng * me = NULL);
int mXcorr(const int maxindex);

private:
const int NumSamp;
const double SampleRateMHz;
const int HRMult;
const double MHztoI;
const double ItoMHz;
double *pData;
double *pWindow;
double MainlobeWidthHz;
double WindowHRFTGain;
int NumSampHRFT;
int WinMult;

std::vector<std::complex<double> > Fft;
std::vector<std::complex<double> > WindowHRFT;
std::vector<std::complex<double> > Xcorr;

#endif

/****** ndf.h ******/
/****** Navigation data file header file ******/

#ifndef _NDF_H
#define _NDF_H

#include <iostream>
#include <fstream>
#include <string>

#endif

#include <iostream>
#include <fstream>
#include <string>
#include "meassingle.h"
#include "dc.h"
#include "ssd.h"
#include "hd.h"

class NavDataFile {
   public:
      NavDataFile(const std::string fn)
         : FileName(fn),
          File(FileName.c_str(),std::ios::out|std::ios::app) {}

      int mWriteClockOffset(const DataCollection & dc);
      int mWriteData(const DataCollection& dc, const SignalSourceData& ssd, const HarmonicData& hd);
      int mWriteFreqs(const SignalSourceData& ssd);
      int mWriteHarm(const MeasSingle& ms);
      int mWriteMeas(const MeasSingle& ms);
      int mWriteSource(const MeasSingle& ms);
   
   private:
      std::string FileName;
      std::ofstream File;
};

#endif

/////////////////////////////////////////////////
// ssd.h
/////////////////////////////////////////////////

/////////////////////////////////////////////////
// Signal source data header file
/////////////////////////////////////////////////

#ifndef _SSD_H
#define _SSD_H

#include <vector>
#include "navdata.h"
#include "meassingle.h"

class SignalSourceData {
   public:
      SignalSourceData(const double sfspmm=20.0, const double scspmm=50.0);

      const std::vector<MeasSingle>& mGetData() const {return Data;}
      int mGetNumStations() const {return NumStations;}

      int mCalc(NavData& nd);

};

#include <vector>
#include "navdata.h"
#include "meassingle.h"

class SignalSourceData {
   public:
      SignalSourceData(const double sfspmm=20.0, const double scspmm=50.0);

      const std::vector<MeasSingle>& mGetData() const {return Data;}
      int mGetNumStations() const {return NumStations;}

      int mCalc(NavData& nd);
int mPrint();

private:
    const int NumStations;
    std::vector<MeasSingle> Data;
};

#endif

/////////////////////////////////////////////////
// util.h
/////////////////////////////////////////////////

/////////////////////////////////////////////////
// Utility inline function header file
/////////////////////////////////////////////////

#ifndef _UTIL_H
#define _UTIL_H

const double TPI=6.283185307179586476925286766590057683943388;
const double PI=TPI/2.0;

#include <complex>
#include <cmath>
#include "meassingle.h"

namespace std {
    template <class T>
    bool operator< (const std::complex<T>& c1, const std::complex<T>& c2) {
        return std::norm(c1)<std::norm(c2);}

    bool ltsqr(const double& c1,const double& c2){
        return ((::pow(c1,2.0)) < (::pow(c2,2.0)));
    }
}

#endif

B.1.2 Source Files

////////////////////////////////////////////////////////////////////////
// main.cpp
////////////////////////////////////////////////////////////////////////

#include <iostream>

#include "windows.h"
#include "winbase.h"
#include "board.h"
#include "comport.h"
#include "dc.h"
#include "fe.h"
#include "fftw.h"
#include "hd.h"
#include "matlabeng.h"
#include "meassingle.h"
#include "navdata.h"
#include "ndf.h"
#include "ssd.h"

using namespace std;

int main(int argc, char *argv[]) {
    //Performance measuring variables
    LARGE_INTEGER SystemFreq;
    LARGE_INTEGER Count1;
    LARGE_INTEGER Count2;
    double DTime;
    QueryPerformanceFrequency(&SystemFreq);

    //Open Com1 for sync pulse generation
    ComPort Com1;

    //Define parameters of front end electronics
    FrontEnd fe;

    //Initialize the Datel board
    Board b;
    b.mInitialize();

    //Define data collection parameters (from file dcparams.txt)
    DataCollection dc(b);
    dc.mInitialize();

    //Define characteristics of the phase cal harmonics
    HarmonicData hd(fe, b.mSampleFreqStab());

    //Define characteristics of the AM Broadcast stations
    SignalSourceData ssd(20.0, b.mSampleFreqStab());

    //Initialize data processing/reduction algorithms
    NavData nd(dc, 10);

    //Initialize FFT algorithm
    Fftw fftw(dc.mWindowLengthSamples());
    //fftw.mMakeWisdom();
    fftw.mInitialize();
// Data file
NavDataFile ndf("output.txt");

// Pulse the com port a few times
for(int i=0;i<5;i++)
{
    Com1.mPulseDTR(50);
    Sleep(1000);
}

DWORD ErrorCode;
int done;
short *pData;

if ((ErrorCode=dc.mStart()) != NOERROR)
{
    cout << "Error mStart: " << ErrorCode << endl;
    return (ErrorCode);
}

// Measure receiver clock offset to a common pulse
dc.mClockOffset(Com1,pData,50);
ndf.mWriteClockOffset(dc);

do{
    if ((ErrorCode=dc.mWait(pData) ) != NOERROR)
    {
        cout << "Error mWait: " << ErrorCode << endl;
        return (ErrorCode);
    }

    // Sound a beep after every data window
    std::cout << "\a" << endl;
    QueryPerformanceCounter(&Count1);

    if ((ErrorCode=dc.mNext(done) ) != NOERROR)
    {
        cout << "Error mNext: " << ErrorCode << endl;
        return (ErrorCode);
    }

    nd.mLoadData(pData,b);
    nd.mApplyWindow();
    nd.mTransform(fftw,NULL);
    ssd.mCalc(nd);
    hd.mCalc(nd);
    ndf.mWriteData(dc,ssd,hd);

    QueryPerformanceCounter(&Count2);
Appendix B: Source Code

\[
DTime = (((\text{double})\text{Count2.QuadPart} - \text{double}\text{Count1.QuadPart})/((\text{double})\text{SystemFreq.QuadPart}));
\]

\[
\text{cout << "Calculation took " << DTime << " seconds." << endl;}
\]

\}

\}

if (((\text{ErrorCode}=\text{dc.mStop})()) != \text{NOERROR})
{
\text{cout << "Error mStop: " << ErrorCode << endl;}
\text{return (ErrorCode);} 
\}

\text{return EXIT_SUCCESS;}
\}

/////// // board.cpp 

// Datel PCI-416 A/D board source file

#include <cmath>
#include <iostream>

#include "416dlllex.h"

#include "board.h"

using namespace std;

int Board::mGetADMax() const
{
\text{return ( (int) (pow(2.0,(\text{double}) (\text{ResolutionBits}-1) )) ) - 1;}
\}

int Board::mGetADMin() const
{
\text{return -( (int) (pow(2.0,(\text{double}) (\text{ResolutionBits}-1) )) );}
\}

\text{DWORD Board::mInitialize()}
{
\text{DWORD BoardCount = BoardNum;}

\text{// initialize board -> loads the device driver}
\text{if (((\text{ErrorCode}=\text{pci416_init(&BoardCount)}) !\text{= NOERROR})}
{
\text{cout << "Error pci416_init: " << ErrorCode << endl;}
\text{return (ErrorCode);} 
\}
if (!BoardCount)
{
    cout << "No PCI416 boards found." << endl;
    return (ERROR_DEV_NOT_EXIST);
}

DWORD caps[5];
if ((ErrorCode=pci416_getcaps(BoardNum, sizeof(caps), caps)) != NOERROR)
{
    cout << "Error pci416_getcaps: " << ErrorCode << endl;
    return (ErrorCode);
}

FifoSizeSamples = caps[0];
// Does the function return bytes or samples ????
DmaBufferSizeBytes = caps[1];
ModelNumber = caps[2];
IrqEnabled = caps[4];

if (IrqEnabled)
{
    cout << "IRQ Enabled!!!  Disable IRQ in control panel applet." << endl;
    return -1;
}

WORD Model;
if ((ErrorCode=get_adm_inf(BoardNum, &Model, &ResolutionBits, &Channels, &ShortCycle, &MaxSingleSampleRate, &MaxScanSampleRate)) != NOERROR)
{
    cout << "Error get_adm_inf: " << ErrorCode << endl;
    return (ErrorCode);
}

ScaleFactor = 10.0/(pow(2.0,((double)(ResolutionBits)))-1);

cout << "Scale Factor: " << ScaleFactor << endl;

return NOERROR;
}
/*******/
#include <iostream>
#include "windows.h"
#include "comport.h"

int ComPort::mPulseDTR(const int & pwms = 50) {
    HCom = CreateFile("Com1",GENERIC_WRITE | GENERIC_READ,0,NULL,OPEN_EXISTING,0,NULL);
    if(HCom == INVALID_HANDLE_VALUE ) {
        std::cout << "Error: ComPort::mPulseDTR: Could not open COM port! Error code: ";
        std::cout << GetLastError() << std::endl;
        exit(EXIT_FAILURE);
    }
    PortReady = SetupComm(HCom,1024,1024);
    PortReady = GetCommState(HCom,&ComDcb);
    ComDcb.BaudRate = 9600;
    ComDcb.ByteSize = 8;
    ComDcb.StopBits = ONESTOPBIT;
    ComDcb.Parity = NOPARITY;
    ComDcb.fDtrControl = DTR_CONTROL_DISABLE;
    PortReady = SetCommState(HCom,&ComDcb);
    ComDcb.fDtrControl = DTR_CONTROL_ENABLE;
    PortReady = SetCommState(HCom,&ComDcb);
    Sleep(pwms);
    ComDcb.fDtrControl = DTR_CONTROL_DISABLE;
    PortReady = SetCommState(HCom,&ComDcb);
    CloseHandle(HCom);
    return EXIT_SUCCESS;
}

// data collection source file

178
```cpp
#include <string>
#include <iostream>
#include <fstream>
#include <iomanip>
#include <time.h>
#include "timer.h"
#include "416dlllex.h"
#include "dc.h"

using namespace std;

int DataCollection::mClockOffset(ComPort & cp, short *&pData,
const int & pwms)
{
    //Maximum number of preceding points to search for half extrema
    //Also maximum number of preceding points to search for local
    //extrema
    const int imax = 20;

    //Maximum fraction of absolute extrema allowed for preceding
    local extrema
    const double maxf = 0.1;

    DWORD WindowStart;
    short Data;
    short *pMax;
    short *pMin;
    short Max = 0;
    short Min = 0;

    int done;

    // Wait for first data window to complete
    do{
        Sleep(10);
        WindowStart = read_pt_acquire_status( TheBoard.mBoardNum() );
    }while (!WindowStart);

    //Generate pulse in the middle of data window
    Sleep( WindowLengthSec*500 - pwms/2 - 20);
    cp.mPulseDTR(pwms);

    //Wait for data window to complete
    if ((ErrorCode=mWait(pData) ) != NOERROR)
    {
        cout << "Error mWait: " << ErrorCode << endl;
        return (ErrorCode);
    }
```
// Wake user up and prompt to remove cables
std::cout << "\a";
Sleep(20);
std::cout << "\a";

std::cout << "Please remove clock sync cables and adjust gain quickly!" << endl;

// Find absolute maximum and minimum of data sample window
for(int i=0;i<WindowLengthSamples;i++)
{
  Data = *(pData+i);
  if(Data > Max)
  {
    Max = Data;
    pMax = pData+i;
  }
  if(Data < Min)
  {
    Min = Data;
    pMin = pData+i;
  }
}

// Check to make sure search ranges below are valid
if( ((pMax - imax - 1) < pData) || ((pMin - imax - 1) < pData) )
{
  std::cout << "Error: DataCollection::mSyncClock: ";
  std::cout << "Pulse is too close to edge of data window!" << endl;
  mExitLoudly();
}

// Make sure absolute extrema are not preceded by significant local extrema. Ensures that hardware is configured correctly.
short *pTest;
for(i=1;i < imax;i++)
{
  pTest = pMax - i;
  if( ( (*pTest) > (*(pTest-1)) )
      && ( (*pTest) > (*(pTest+1)) )
      && ( ((double)(*pTest)) > (maxf*((double)(*pMax))) ) )
  {
    std::cout << "Error: DataCollection::mSyncClock: Significant local max found!" << endl;
    std::cout << "Check sync cable connections and retry." << endl;
  }
mExitLoudly();
}

for(i=1; i < imax; i++)
{
pTest = pMin - i;

if( (*pTest) < (*pTest-1) )
&& ( (*pTest) < (*pTest+1) )
&& ( ((double)(*pTest)) < (maxf*((double)(*pMin))) )
{
    std::cout << "Error: DataCollection::mSyncClock: Significant local min found!" << endl;
    std::cout << "Check sync cable connections and retry." << endl;
    mExitLoudly();
}
}

// Back off absolute extrema to find first points
// that are less/greater than half extrema value
short *pHalfMax;
short *pHalfMin;
bool HalfMaxFound = false;
bool HalfMinFound = false;

for(i=0; i < imax; i++)
{
pTest = pMax - i;

if( (*pTest) <= (Max/2) )
{
    pHalfMax = pTest;
    HalfMaxFound = true;
    break;
}
}

for(i=0; i < imax; i++)
{
pTest = pMin - i;

if( (*pTest) >= (Min/2) )
{
    pHalfMin = pTest;
    HalfMinFound = true;
    break;
}
}
//Make sure half points were found
if( (!HalfMaxFound) || (!HalfMinFound) )
{
    std::cout << "Error: DataCollection::mSyncClock: Half points
                not found!" << endl;
    mExitLoudly();
}

//Interpolate to find clock offset to pulse
double MaxOffset;
double MinOffset;
MaxOffset = ( (double) (Max/2 - *pHalfMax) ) / ( (double)
(*pHalfMax+1) - *pHalfMax )
MinOffset = ( (double) (Min/2 - *pHalfMin) ) / ( (double)
(*pHalfMin+1) - *pHalfMin )

ClockOffsetPos = ( ((double) (pHalfMax - pData)) + MaxOffset
)/SampleRate;
ClockOffsetNeg = ( ((double) (pHalfMin - pData)) + MinOffset
)/SampleRate;

//Make sure pulse width is reasonable and that max occurs before
//min
double pw = ClockOffsetNeg - ClockOffsetPos;
double apw = ((double) pwms)/1000.0;
if( (pw > (apw + apw/2)) || (pw < (apw - apw/2)) )
{
    std::cout << "Error: DataCollection::mSyncClock: Pulse width
              abnormal!" << endl;
    mExitLoudly();
}

std::cout << setw(18) << fixed << setprecision(15);
std::cout << "Clock offset: " << ClockOffsetPos << " " <<
ClockOffsetNeg << endl;

mNext(done);

return EXIT_SUCCESS;

int DataCollection::mExitLoudly()
{
    std::cout << "Press Cntl-C to quit." << endl;
    while(true)
    {
        std::cout << "\a";
        Sleep(50);
    }
    return EXIT_SUCCESS;
}
//This function should not be necessary. Suggested as possible
fix by Datel.
int DataCollection::mFixSignExt(short*& pData)
{
    const unsigned short SetMask = 61440;
    const unsigned short ClearMask = 4095;
    const unsigned short TestMask = 2048;

    short * pd;

    for(int i=0; i< WindowLengthSamples; i++)
    {
        pd = pData+i;

        if( (*pd) & TestMask )
            (*pd) = (*pd) | SetMask;
        else
            (*pd) = (*pd) & ClearMask;
    }

    return EXIT_SUCCESS;
}

int DataCollection::mInitialize()
{
    string FileName("dcparams.txt");
    ifstream File(FileName.c_str());
    string s;
    int StartHour;
    int StartMin;
    int StartSec;

    if(File)
    {
        std::cout << "Initializing data collection parameters from " << FileName << "." << endl;
        int cs;
        while(File >> s)
        {
            if(s=="StartTime"){
                File >> StartHour;
                File >> StartMin;
                File >> StartSec;
                continue;
            }

            if(s=="WindowLengthSamples"){
                File >> WindowLengthSamples;
            }

            // Other code...
        }
    }
}
std::cout << "Number of samples in data window: " << WindowLengthSamples << endl;
    continue;
}
if(s=="TriggerRate"){
    File >> TriggerRate;
    TriggerPeriod = 1.0/TriggerRate;
    std::cout << "Trigger rate (Hz): " << TriggerRate << endl;
    continue;
}
if(s=="TotalLengthSec"){
    File >> TotalLengthSec;
    std::cout << "Total length of data collection (seconds): " << TotalLengthSec << endl;
    continue;
}
if(s=="ExtClockSource"){
    File >> cs;
    if(cs==0)
    {
        ClockSource = 0;
        TriggerSource = 0;
        std::cout << "Using internal clock." << endl;
    }
    if(cs==1)
    {
        ClockSource = 1;
        TriggerSource = 1;
        std::cout << "Using external clock." << endl;
    }
    continue;
}
else
{
    std::cout << FileName << " not found. Using default parameters." << endl;
}

// Calculate collection parameters
WindowLengthSec = ((double)WindowLengthSamples)/SampleRate;
    cout << "Length of data window in seconds: " << WindowLengthSec << endl;
    TotalLengthWindows = (DWORD)(TriggerRate*TotalLengthSec);

    // Set start time
    struct tm *StartTm;
    time_t CurrentTt;

    time( &CurrentTt );
    std::cout << "The current time is : " << ctime( &CurrentTt );
if( StartHour < 23)
{
    StartTm = localtime( &CurrentTt );
    StartTm->tm_hour = StartHour;
    StartTm->tm_min = StartMin;
    StartTm->tm_sec = StartSec;
    StartTime = mktime( StartTm );

    if ( difftime( StartTime, CurrentTt ) < 0.0)
    {
        std::cout << "The requested start time has already past!" << endl;
        exit(EXIT_FAILURE);
    }
    else
    {
        std::cout << "The requested start time is: " << ctime( &StartTime );
    }
    else
    {
        StartTime = 0;
        std::cout << "The requested start time is: Now!" << endl;
    }

    return 0;
}

DWORD DataCollection::mNext(int& done)
{
    WindowCount++;
    if (WindowCount >= TotalLengthWindows)
        done = 1;
    else{
        done = 0;
        if (BufNum==0)
            BufNum=1;
        else
            BufNum=0;
        if ((ErrorCode=pci416_reload_dma(TheBoard.mBoardNum(), BufNum,
            &DoubleBufferSizeBytes[BufNum], &DoubleBufferHandle[BufNum]))
            != NOERROR)
            return ErrorCode;
        return 0;
    }

DWORD DataCollection::mStart()
{
    //Check to see if boot time allocation is large enough to support
    //double buffering
if (TheBoard.mDmaBufferSizeBytes() >= WindowLengthSamples << 2)
// Set the size of both dma buffers to hold one window of data
    DoubleBufferSizeBytes[0] = DoubleBufferSizeBytes[1] = WindowLengthSamples << 1;
else
{
    cout << "Error: Boot-time DMA buffer allocation: " << (TheBoard.mDmaBufferSizeBytes() >> 1) << " samples." << endl;
    cout << "Required size: " << (WindowLengthSamples << 1) << " samples." << endl;
    cout << "Use control panel applet to increase size and reboot machine." << endl;
    return -1;
}

// Set up the PCI-416 with the desired data collection parameters
if ((ErrorCode = set_modes(TheBoard.mBoardNum(), ClockSource,
        SampleRate, WindowLengthSamples,
        TriggerSource, Channel, 0, 0, 0)) != NOERROR)
{
    cout << "Error set_modes : " << ErrorCode << endl;
    return ErrorCode;
}

if ((ErrorCode=pci416_setup_dma(TheBoard.mBoardNum(),
        DMA_DOUBLE,
        DoubleBufferSizeBytes, &DoubleBufferHandle[BufNum])) != NOERROR)
{
    cout << "Error pci416_setup_dma: " << ErrorCode << endl;
    return ErrorCode;
}

// Wait until requested start time
struct _timeb TimeBuffer;
int WaitMs;

if( StartTime )
{
    cout << "Waiting until: " << ctime( &StartTime );
    _ftime( &TimeBuffer );
    WaitMs = (StartTime - TimeBuffer.time)*1000 -
        TimeBuffer.millitm;
    Sleep(WaitMs-5);
}
else
time( &StartTime );

// The following code is a rewrite of start_daq that
// does a better job of setting the trigger rate
set_pt_convert_enable(TheBoard.mBoardNum(), 1);
// The following is a very simple algorithm for factoring
// the trigger period count into the two counter registers.
// It only works for a limited range of trigger rates, but it
// will work well for the trigger rates that we are likely to
// choose.
if( (TriggerRate > 250) || (TriggerRate < 0.007) )
{
    cout << "TriggerRate is out of bounds." " << endl;
    exit(EXIT_FAILURE);
}

UINT hword,lword;
hword = 10000;
lword = (int) (5.0e6/(TriggerRate*hword) + 0.5);
lword = lword * 2;

pci416_set_timer(TheBoard.mBoardNum(), TM_CONT_TRIGGER, hword, lword);

TriggerRate = 10.0e6/(hword*lword);

TriggerPeriod = 1.0/TriggerRate;

cout << "Trigger rate: " << TriggerRate << " Hz." << endl;
return NOERROR;

DWORD DataCollection::mStop()
{
    if ((ErrorCode=pci416_stop_dma(TheBoard.mBoardNum(), TransferCountRem)) == NOERROR)
        cout << "DMA stopped: Transfer count remaining: " << TransferCountRem << endl;
    else
        {cout << "Error pci416_stop_dma: " << ErrorCode << endl;
         return ErrorCode;
        }

    if ((ErrorCode=stop_daq(TheBoard.mBoardNum())) == NOERROR)
        cout << "DAQ stopped." << endl;
    else
        {cout << "Error pci416_stop_daq: " << ErrorCode << endl;
         return ErrorCode;
        }
    return NOERROR;
}

DWORD DataCollection::mWait(short*& pData)
DWORD WindowComplete=0;

cout << "Waiting";

pData = (short *) DoubleBufferHandle[BufNum];
do{
    std::cout << ".";
    if ( ((ErrorCode=pci416_dma_status(TheBoard.mBoardNum(), &WindowComplete)) != NOERROR))
        return ErrorCode;
    Sleep(10);
}while (!WindowComplete);

cout << "done!" << endl;

if(pt_fifo_full(TheBoard.mBoardNum()))
    cout << "Fifo full!" << endl;

WindowNum = WindowCount;

return 0;

}
int Fftw::mFft(double *pData, std::vector< std::complex<double> >& fft)
{
    int i = 0;
    int imax = 0;

    //rfftw_one(FftPlan, pData, pFftTemp);
    rfftw_threads_one(2, FftPlan, pData, pFftTemp);

    imax = NumSamp/2;  //Integer division (rounds down)

    //First point of DFT of real sequence is purely real
    fft[0] = std::complex<double>(*pFftTemp,0.0);

    //Copy data to complex vector, except last point
    //The FFTW package only stores the first half of the spectrum of a real
    //sequence because the other half is the complex conjugate of the first
    //This ends up being a (N/2)+1 length complex sequence
    //We are not storing the (N/2) complex number because is not of interest
    //and this leaves us with a nice N/2 length sequence and eliminates some logic
    for(i=1;i<imax;i++)
    {
        fft[i] =
        std::complex<double>(*pFftTemp+i),*(pFftTemp+(NumSamp-i)));
    }

    return EXIT_SUCCESS;
}

int Fftw::mInitialize()
{
    FILE *WisdomFile;
    bool HaveWisdom;

    fftw_threads_init();

    if((WisdomFile = fopen("wisdom.dat", "r")) == NULL)
    {
        cout << "Wisdom file not found!" << endl;
        HaveWisdom = false;
    }
    else
    {
Appendix B: Source Code

```cpp
if (FFTW_FAILURE == fftw_import_wisdom_from_file(WisdomFile))
{
    cout << "Error reading wisdom!" << endl;
    HaveWisdom = false;
} else {
    cout << "Wisdom file loaded!" << endl;
    HaveWisdom = true;
}
fclose(WisdomFile);

if(HaveWisdom)
    FftPlan = rfftw_create_plan(NumSamp, FFTW_REAL_TO_COMPLEX,
    FFTW_MEASURE | FFTW_USE_WISDOM);
else
    FftPlan = rfftw_create_plan(NumSamp, FFTW_REAL_TO_COMPLEX,
    FFTW_ESTIMATE);
return EXIT_SUCCESS;

int Fftw::mMakeWisdom()
{
    FILE *WisdomFile;

    WisdomFile = fopen("wisdom.dat", "w");
    FftPlan = rfftw_create_plan(NumSamp, FFTW_REAL_TO_COMPLEX,
    FFTW_MEASURE | FFTW_USE_WISDOM);
    fftw_export_wisdom_to_file(WisdomFile);
    fclose(WisdomFile);

    return EXIT_SUCCESS;
}

#include <iostream>
#include "hd.h"
using namespace std;

HarmonicData::HarmonicData(const FrontEnd& fe, const double scsppm)
```
{  
double hfsppm = fe.mGetPhaseCalStab();

    PhaseCalFundamental = fe.mGetPhaseCalBase()/(double)
fe.mGetPhaseCalDivisor());
    cout << "Phase Cal Fundamental (MHz) = " << PhaseCalFundamental
<< endl;

    int LowestHarmonic = (int)
ceil(fe.mGetHighPassCutoff()/PhaseCalFundamental);
    cout << "Lowest harmonic = " << LowestHarmonic << endl;

    int HighestHarmonic = (int)
floor(fe.mGetLowPassCutoff()/PhaseCalFundamental);
    cout << "Highest harmonic = " << HighestHarmonic << endl;

    NumHarms = HighestHarmonic - LowestHarmonic + 1;
    cout << "Number of harmonics = " << NumHarms << endl;

    Data.resize(NumHarms);

    int Harm;
    double NomFreq;
    double Stab;

    for(int i=0;i<NumHarms;i++)
    {
        Harm = LowestHarmonic + i;
        NomFreq = PhaseCalFundamental*((double) Harm);
        Stab = (hfsppm+scsppm)*(1.0e-6)*NomFreq;
        Data[i].mInit(Harm,NomFreq,Stab);
    }
}

int HarmonicData::mCalc(NAVData& nd)
{
    int istop = Data.size();
    for(int i=0;i<istop;i++)
        nd.mGetFPM(Data[i]);

    return EXIT_SUCCESS;
}

________________________________________________________
// matlabeng.cpp
________________________________________________________

// Matlab engine source file
________________________________________________________
# include <iostream>

# include "matlabeng.h"

using namespace std;

MatlabEng::MatlabEng() 
{
    cout << "Opening Matlab engine..." << endl;
    pEng = engOpen("\0");
    cout << "Matlab engine opened successfully!" << endl;
}

int MatlabEng::mHold() 
{
    engEvalString(pEng,"hold on");
    return EXIT_SUCCESS;
}

int MatlabEng::mPlot(const std::vector<std::complex<double> >& y) 
{
    mxArray *pMdata;
    double *pMdatar;
    double *pMdatai;
    int i;
    const int imax=y.size();

    pMdata = mxCreateDoubleMatrix(1,imax,mxCOMPLEX);
    pMdatar = mxGetPr(pMdata);
    pMdatai = mxGetPi(pMdata);
    mxSetName(pMdata, "data");
    for(i=0;i<imax;i++)
    {
        *(pMdatar+i)=real(y[i]);
        *(pMdatai+i)=imag(y[i]);
    }
    engPutArray(pEng,pMdata);
    engEvalString(pEng, "plot(abs(data));");

    mxDestroyArray(pMdata);

    return EXIT_SUCCESS;
}

int MatlabEng::mPlot(const double * const pData, int NumElem) 
{
    mxArray *pMdata;

    pMdata = mxCreateDoubleMatrix(1,NumElem,mxREAL);
B.1 Software Radio Program

```c
mxSetName(pMdata, "data");
memcpy((void *) mxGetPr(pMdata), (void *) pData, sizeof(double)*NumElem);
engPutArray(pEng,pMdata);

engEvalString(pEng, "plot(data);");

mxDestroyArray(pMdata);

return EXIT_SUCCESS;
}

int MatlabEng::mPlot(const std::vector<double>& x, const std::vector<double>& y, const int num)
{
    mxArray *pX;
    mxArray *pY;
    double *pXr;
    double *pYr;
    int i;
    int imax = x.size();

    pX = mxCreateDoubleMatrix(1,imax,mxREAL);
    pY = mxCreateDoubleMatrix(1,imax,mxREAL);
    pXr = mxGetPr(pX);
    pYr = mxGetPr(pY);

    switch(num)
    {
    case 1 :
        mxSetName(pX, "x1");
        mxSetName(pY, "y1");
        break;
    case 2 :
        mxSetName(pX, "x2");
        mxSetName(pY, "y2");
        break;
    case 3 :
        mxSetName(pX, "x3");
        mxSetName(pY, "y3");
        break;
    default :
        mxSetName(pX, "x0");
        mxSetName(pY, "y0");
    }

    for(i=0;i<imax;i++)
    {
        *(pXr+i)=x[i];
        *(pYr+i)=y[i];
    }
}
```
Appendix B: Source Code

```cpp
engPutArray(pEng, pX);
engPutArray(pEng, pY);

switch(num)
{
    case 1:
        engEvalString(pEng, "plot(x1,y1,'mo');");
        break;
    case 2:
        engEvalString(pEng, "plot(x2,y2,'g*');");
        break;
    case 3:
        engEvalString(pEng, "plot(x3,y3,'r+');");
        break;
    default:
        engEvalString(pEng, "plot(x0,y0,'bd');");
}

mxDestroyArray(pX);
mxDestroyArray(pY);

return EXIT_SUCCESS;
}

int MatlabEng::mPlotFft(const vector<complex<double> >& data,
const double srm,
const std::string morp)
{
    mxArray *pMx;
    mxArray *pMy;
    double *pMxr;
    double *pMyr;
    int i;

    const int NumElem = data.size();
    const double FundMHz = (srm/2.0)/((double) NumElem);

    //Create frequency value vector and copy to Matlab workspace
    pMx = mxCreateDoubleMatrix(1,NumElem,mxREAL);
    pMxr = mxGetPr(pMx);
    mxSetName(pMx, "X");
    for(i=0;i<NumElem;i++)
        (*(pMxr+i)) = FundMHz*((double) i);
    engPutArray(pEng, pMx);

    //Calculate phase or db magnitude depending on value of morp
    pMy = mxCreateDoubleMatrix(1,NumElem,mxREAL);
    pMyr = mxGetPr(pMy);
    mxSetName(pMy, "Y");
    if(morp == "Mag")
        for(i=0;i<NumElem;i++)
            (*(pMyr+i)) = ...
```

194
B.1 Software Radio Program

```c
{  for(i=0;i<NumElem;i++)  
    (*)(pMyr+i)) = 10*log10(norm(data[i]));  
    engPutArray(pEng,pMy);  
    engEvalString(pEng, "plot(X,Y)");  
    engEvalString(pEng, "ylabel('Magnitude of DFT (db)');");  
  }
if(morp == "Phase")  
{  
    for(i=0;i<NumElem;i++)  
      (*)(pMyr+i)) = arg(data[i]);  
    engPutArray(pEng,pMy);  
    engEvalString(pEng, "plot(X,Y)");  
    engEvalString(pEng, "ylabel('Phase (radians)');");  
  }
  engEvalString(pEng, "title('Discrete Fourier Transform');");  
  engEvalString(pEng, "xlabel('Frequency (MHz)');");

  mxDestroyArray(pMx);
  mxDestroyArray(pMy);

  return EXIT_SUCCESS;
}

////////////////////////////////////////////////////
// meassingle.cpp
////////////////////////////////////////////////////

////////////////////////////////////////////////////
// AM measurements source file
////////////////////////////////////////////////////

#include <iostream>
#include "meassingle.h"

using namespace std;

int MeasSingle::mInit(int id, double nf, double st)
{
  Id = id;
  NomFreq=nf;
  Stab=st;
  Freq=0.0;
  Phase=0.0;
  Mag=0.0;
  return EXIT_SUCCESS;
}

int MeasSingle::mPrint()
```
Appendix B: Source Code

```cpp
{
    cout << Id << " " << NomFreq << " " << Stab << " " << Freq << " " << Phase << " " << Mag << endl;
    return EXIT_SUCCESS;
}

int MeasSingle::mSetFPM(double f, double p, double m)
{
    Freq = f;
    Phase = p;
    Mag = m;
    return EXIT_SUCCESS;
}

////////////////////////////////////////////////////
// navdata.cpp
////////////////////////////////////////////////////

////////////////////////////////////////////////////
// Navigation data source file
////////////////////////////////////////////////////

#include <algorithm>
#include <iostream>
#include <numeric>
#include <vector>
#include "time.h"
#include "dc.h"
#include "matlabeng.h"
#include "meassingle.h"
#include "navdata.h"
#include "util.h"

using namespace std;

NavData::NavData(const DataCollection& dc, const int hrm):
    NumSamp(dc.mWindowLengthSamples()),
    SampleRateMHz(dc.mSampleRateMHz()),
    HRMult(hrm),
    MHztoI((double) NumSamp/SampleRateMHz),
    ItoMHz(1.0/MHztoI)
{
    pData = new double[NumSamp];
    pWindow = new double[NumSamp];
    Fft.resize(NumSamp/2);
    Xcorr.resize(HRMult*2+1);
    mCalcWindow();
}

NavData::~NavData()
```
B.1  Software Radio Program

```c
{
  if(pData)
    delete pData;
  if(pWindow)
    delete pWindow;
}

int NavData::mApplyWindow()
{
  for(int i=0;i<NumSamp;i++)
    *(pData+i) = (*(pWindow+i))(*(pData+i));
  return EXIT_SUCCESS;
}

// Computes the Fourier transform of a Blackman window
inline std::complex<double> NavData::mBwft(int nElem, double f, double fs)
{
  // Optimized for accuracy
  double fr = fs/((double) (nElem-1));

  std::vector<std::complex<double> > results(5);
  std::vector<double> realresults(5);
  std::vector<double> imagresults(5);
  double realresult;
  double imagresult;

  results[0] = 0.42*mRwft(nElem,f,fs);
  results[1] = -0.25*mRwft(nElem,f-fr,fs);
  results[2] = -0.25*mRwft(nElem,f+fr,fs);
  results[3] = 0.04*mRwft(nElem,f-2.0*fr,fs);
  results[4] = 0.04*mRwft(nElem,f+2.0*fr,fs);

  for(int i=0;i < 5;i++)
  {
    realresults[i] = real(results[i]);
    imagresults[i] = imag(results[i]);
  }

  sort(realresults.begin(),realresults.end(),ltsqr);
  sort(imagresults.begin(),imagresults.end(),ltsqr);

  realresult =
  accumulate(realresults.begin(),realresults.end(),0.0);
  imagresult =
  accumulate(imagresults.begin(),imagresults.end(),0.0);

  return std::complex<double>(realresult,imagresult);
}

int NavData::mCalcWindow()
```
Appendix B: Source Code

```c++
{
    int i;
    double id;
    const double Coef1 = 2.0*PI/((double)(NumSamp-1));
    const double Coef2 = 2.0*Coef1;
    const double mlb = ((SampleRateMHz*1.0e6)/((double) (NumSamp-1)));
    const double hrfund = SampleRateMHz/(((double) NumSamp)*((double) HRMult));

    for(i=0;i<NumSamp;i++)
    {
        id = (double) i;
        *(pWindow+i)=0.42-0.5*cos(Coef1*id)+0.08*cos(Coef2*id);
    }

    WinMult = 6;

    MainlobeWidthHz = mlb*WinMult;
    cout << "Mainlobe width (Hz): " " << MainlobeWidthHz << endl;

    NumSampHRFT = HRMult*WinMult+1;
    WindowHRFT.resize(NumSampHRFT);
    const int midp = (NumSampHRFT-1)/2;

    //Calculate high resolution FT of window - mainlobe only
    cout << "Calculating Hi res..." " << endl;
    for(i=0;i<=midp;i++)
    {
        WindowHRFT[midp+i] = mBwft(NumSamp,hrfund*i,SampleRateMHz);
        WindowHRFT[midp-i] = conj(WindowHRFT[midp+i]);
    }

    cout << "Done!" " << endl;

    //Calculate RMS "gain" of the window fourier transform in db
    WindowHRFTGain = norm(WindowHRFT[midp]);
    for(i=1;i<WinMult/2;i++)
    {
        WindowHRFTGain += 2.0*norm(WindowHRFT[midp+i*HRMult]);
        WindowHRFTGain = 10.0*log10(WindowHRFTGain);
    }

    cout << "Window gain: " " << WindowHRFTGain << " db" " << endl;

    return EXIT_SUCCESS;
}

int NavData::mGetFPM(MeasSingle& ms)
{
    double Freq;
    double Phase;
    double Mag;

    int maxi;
    int swpts;
```
int corrcor;
double freqi;

std::vector<std::complex<double> >::iterator iter1;

maxi = mMHztol(ms.mGetNomFreq());
swpts = mMHztol(ms.mGetStab());

iter1 = Fft.begin() + maxi;
maxi += ( max_element(iter1-swpts,iter1+swpts) - iter1 );

mXcorr(maxi);

iter1 = max_element(Xcorr.begin(),Xcorr.end());
corrcor = iter1 - Xcorr.begin() - (Xcorr.size()/2);
double icorr = mInterpolate(iter1);

//Check if index is odd or even and adjust appropriate phases by pi
int mask = 1;
if( maxi & mask )
    Phase = arg( Fft[maxi-1] - Fft[maxi] + Fft[maxi+1] );
else
    Phase = arg( -Fft[maxi-1] + Fft[maxi] - Fft[maxi+1] );

//Fit (least squares) magnitude to a parabola and evaluate at interpolated maximum
const std::complex<double> ym1 = *(iter1-1);
const std::complex<double> y0 = *iter1;
const std::complex<double> y1 = *(iter1+1);
Mag = ((0.5)*(abs(y1)+abs(ym1))-abs(y0))*icorr*icorr;
Mag += (0.5)*(abs(y1)-abs(ym1))*icorr;
Mag += abs(y0);

freqi = -((double) icorr)/((double) HRMult);
freqi -= ((double) corrcor)/((double) HRMult);
freqi += (double) maxi;
Freq = freqi * ItoMHz;

if(Mag > 1)
    Mag = 20.0*log10(Mag) - WindowHRFTGain;
else
{
    Mag = 0.0;
    Phase = 0.0;
    Freq = ms.mGetNomFreq();
}
ms.mSetFPM(Freq,Phase,Mag);

return EXIT_SUCCESS;
double NavData::mInterpolate(const std::vector<complex<double>> ::iterator i)
{
    double a,b,c;
    
a = abs(*(i-1));
b = abs(*i);
c = abs(*(i+1));

    return ((1.0/2.0)*(c-a)/(2*b-c-a));
}

int NavData::mLoadData(short *pRawData, const Board& ADBoard)
{
    double ScaleFactor = ADBoard.mScaleFactor();
    bool Sat = false;
    short RawData;
    short MaxRaw = 0;
    short MinRaw = 0;

    for(int i=0;i<NumSamp;i++)
    {
        RawData = *(pRawData+i);

        //Check for A/D saturation and issue a warning if suspected
        if( (RawData == 2047) || (RawData == -2048))
            Sat = true;

        if(RawData > MaxRaw)
            MaxRaw = RawData;
        if(RawData < MinRaw)
            MinRaw = RawData;

        *(pData+i)=( ((double)(RawData))+0.5 )*ScaleFactor;
    }

    if(Sat)
        std::cerr << "Warning:  Possible A/D saturation." " << endl;
    cout << "Max: " << MaxRaw << "  Min: " << MinRaw << endl;

    return EXIT_SUCCESS;
}

// Computes the Fourier transform of a rectangular window
inline std::complex<double> NavData::mRwft(int nElem, double f, double fs)
{
    const double fzero = PI*f/fs;
    double mag = 0.0;

    if(f < 1.0e-25 && f > -1.0e-25)
mag = (double) nElem;
else
    mag = sin(fzero*nElem)/sin(fzero);
const double phase = -fzero*(nElem-1);
return std::complex<double>(std::polar(mag,phase));
}

int NavData::mTransform(Fftw& fftw, MatlabEng * me)
{
    fftw.mFft(pData, Fft);
    if(me)
    {
        me->mPlotFft(Fft);
        fgetc(stdin);
    }
    return EXIT_SUCCESS;
}

int NavData::mXcorr(const int maxindex)
{
    //index into FFT array
    int i;
    const int imax = (WinMult-2)/2;
    const int imin = -imax;

    //index into HR Window DFT array
    int j;
    const int jmax = WindowHRFT.size()-1;
    const int joff = ((HRMult*WinMult)/2)-((Xcorr.size()-1)/2);

    //index into result
    int k;
    const int kmax = Xcorr.size()-1;

    for(k=0;k<=kmax;k++)
    {
        Xcorr[k] = std::complex<double>(0.0,0.0);
        for(i=imin;i<=imax;i++)
        {
            j=i*HRMult+k+joff;
            if(j>=0 && j<=jmax)
                Xcorr[k]+= (Fft[i+maxindex])*(conj(WindowHRFT[j]));
        }
    }
    return EXIT_SUCCESS;
}
# Appendix B: Source Code

```cpp
#include <iostream>
#include <iomanip>
#include "ndf.h"

using namespace std;

int NavDataFile::mWriteClockOffset(const DataCollection & dc)
{
  if(File)
  {
    File << "T ";
    File << dc.mGetStartTime() << " ";
    File << dc.mGetWindowNum() << " ";
    File << setw(20) << fixed << setprecision(15) <<
    dc.mGetTriggerPeriod();
    File << endl;

    File << "S ";
    File << setw(19) << fixed << setprecision(15) <<
    dc.mGetClockOffsetPos() << " ";
    File << setw(19) << fixed << setprecision(15) <<
    dc.mGetClockOffsetNeg();
    File << endl;
  }
  return EXIT_SUCCESS;
}

int NavDataFile::mWriteData(const DataCollection& dc, const
SignalSourceData& ssd, const HarmonicData& hd)
{
  int i;
  int istop;

  const std::vector<MeasSingle> SData(ssd.mGetData());
  const std::vector<MeasSingle> HData(hd.mGetData());

  if(File)
  {
    File << "T ";
    File << dc.mGetStartTime() << " ";
    File << dc.mGetWindowNum() << " ";
    File << setw(20) << fixed << setprecision(15) <<
    dc.mGetTriggerPeriod();
    File << endl;
```
...
Appendix B: Source Code

```cpp
int NavDataFile::mWriteHarm(const MeasSingle& ms)
{
    File << "C ";
    mWriteMeas(ms);
    return EXIT_SUCCESS;
}

int NavDataFile::mWriteMeas(const MeasSingle& ms)
{
    File << setw(5) << ms.mGetId() << " ";
    File << setw(18) << fixed << setprecision(15) << ms.mGetFreq() << " ";
    File << setw(19) << fixed << setprecision(15) << ms.mGetPhase() << " ";
    File << setw(20) << fixed << setprecision(15) << ms.mGetMag();
    File << endl;
    return EXIT_SUCCESS;
}

int NavDataFile::mWriteSource(const MeasSingle& ms)
{
    File << "R ";
    mWriteMeas(ms);
    return EXIT_SUCCESS;
}

#include <algorithm>
#include <functional>
#include "ssd.h"

using namespace std;

// Signal source data source file

// ssd.h

SignalSourceData::SignalSourceData(const double sfSppm, const double scSppm): NumStations(170-53+1)
{
    Data.resize(NumStations);

    int Sta;
    double NomFreq;
    double Stab;

    for(int i=0;i<NumStations;i++)
    {
```
B.2 Navigation Algorithm Program

The navigation algorithm program processes the software radio program output files from the base station and the rover. It implements the ambiguity function method to produce position estimates.

B.2.1 Header Files

///////////////////////////////////////////////////////////////////////////////
// af.h
///////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// Ambiguity function header file
////////////////////////////////////////////////////////////////////////////////

#ifndef _AF_H
#define _AF_H

#include <vector>
#define MSWIND //Tells Matlab libraries we are using Windows
#define MSVC    //Tells Matlab libraries we are using Visual C++
#include "matlab.hpp"

#endif _AF_H

#include <vector>

#define MSWIND //Tells Matlab libraries we are using Windows
#define MSVC    //Tells Matlab libraries we are using Visual C++
#include "matlab.hpp"

205
#include "meas.h"

class StationSet;
class AFElem;

class AmbiguityFunc {
public:

AmbiguityFunc(const StationSet & ss, const MeasSetFD & ms,
const double & baseeast, const double & basenorth);

const int & mGetNumUsed() const {return NumUsed;}
int mGetResCnt(const double & ThresholdM) const;
void mPrint(std::ostream& OutStream = std::cout) const;
int mPrintIds(std::ostream& OutStream = std::cout) const;
int mPrintPhaseResMeters(std::ostream& OutStream = std::cout) const;
int mPrintResiduals(std::ostream& OutStream = std::cout) const;
int mPrintWeights(std::ostream& OutStream = std::cout) const;
int mDontUseOddMan(const int & num=1);
void mInit();
void mNeverUse(const std::vector<int> & ids);
void mNeverReport(const std::vector<int> & ids);
int mNormalizeWeight(const std::string & method = "equal");
void mSetBasePos(const double & baseeast, const double &
basenorth)
    { BaseEast=baseeast; BaseNorth=basenorth; }
void mSetFBFExp(const int & fbfe) { FBFExp = fbfe; }
void mSetResiduals(const mwArray & sv);
double operator()( const mwArray & sv );
double operator()( const mwArray & sv, mwArray & gradf );

private:
int FBFExp;
int NumUsed;

double BaseEast;
double BaseNorth;

std::vector<AFElem> Elements;
};

class Station;
class MeasElemFD;

class AFElem {
public:

AFElem(const Station & sta, const MeasElemFD & meas);
const int & mGetId() const {return Meas.mGetId();}
const double & mGetMagMin() const {return Meas.mGetMagMin();}
const double & mGetMagRoam() const {return Meas.mGetMagRoam();}
double mGetResidualMeters() const { return (acos(Residual)/Wavenumber ); }
bool mGetUse() const {return Use;}
const double & mGetWeight() const {return Weight;}
int mPrint(std::ostream& OutStream = std::cout) const;
int mPrintId(std::ostream& OutStream = std::cout) const;
int mPrintResidual(std::ostream& OutStream = std::cout) const;
int mPrintPhaseResMeters(std::ostream& OutStream = std::cout) const;
int mPrintWeights(std::ostream& OutStream = std::cout) const;
bool operator<(AFElem & rhs) const;
void mDontUse() {Use = false;}
// mInit should only be called after MeasElemFD had been updated
// with latest epoch of data and error checking had been
// performed
// on that data!!!!
bool mInit(const double & baseeast, const double & basenorth);
void mNeverUse() {NeverUse = true;}
void mNeverReport() {NeverReport = true;}
void mSetResidual(const mwArray & sv);
void mSetWeight(const double & wt) {Weight = wt;}
AFElem& operator=(const AFElem & rhs);
double operator()(const mwArray & sv, const int & fbfexp);
double operator()(const mwArray & sv, const int & fbfexp,
mwArray & gradf);

private:

const MeasElemFD & Meas;
const Station & Sta;

bool Use;
bool NeverUse;
bool NeverReport;

double BaseThPhase;
double Wavenumber;
double Residual;
double PhaseRes;
double Weight;

};

#endif

////////////////////////////////////////////////////
// afmax.h

207
Appendix B: Source Code

#ifndef _AFMAX_H
#define _AFMAX_H

#define MSWIND // Tells Matlab libraries we are using Windows
#define MSVC // Tells Matlab libraries we are using Visual C++
#include "matlab.hpp"
#include "af.h"
#include "bfgs.h"

class MatlabEng;

class AFMax : public BFGSMin {
public:

AFMax(AmbiguityFunc & af, const double sf = -1.0e5);

int mPrintBestMins(std::ostream& OutStream = std::cout) const;
int mPrintLocHeader(std::ostream& OutStream = std::cout) const;
int mPrintMins(std::ostream& OutStream = std::cout, int num=10) const;
int mPrintMinsNormTable(std::ostream& OutStream = std::cout) const;
bool mTooCloseToMin(const mwArray & point, const double & threshold) const;
bool mAddMin(const mwArray & point);

double mCompute(const mwArray & x) {return ScaleFactor*AF(x);}
double mComputeGrad(const mwArray & x, mwArray & dx);
int mFindAll(const mwArray & Center, const mwArray & Range, const double StepPercent);
double mFindMaxExh(mwArray & Center, const mwArray & Range, const double StepPercent, MatlabEng * me=NULL);
int mOddManBestMins();
int mPickBest(mwArray & sv);
void mPickBestMins() { mSortMinsAF(); BestMins = Mins(colon(),colon(1,NumBestMin)); }
int mPickBOTB(mwArray & Best);
void mSortBestMinsNumUsed() { BestMins = transpose(sortrows(transpose(BestMins),5)); }
void mSortMinsAF() { Mins(colon(),colon(1,NumMin)) = -transpose(sortrows(transpose(-Mins(colon(),colon(1,NumMin))),4)); }
};
private:

AmbiguityFunc & AF;

const int MaxNumMin;
const int NumBestMin;
int NumMin;

const double MaxFuncValToCalc;
const double MinSpToCalcM;
const double ScaleFactor;

mwArray BestMins;
mwArray Mins;

};

#endif

/////////////////////////////////////////////////////////////////////
// bfgs.h
/////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////
// BFGS minimization header file
/////////////////////////////////////////////////////////////////////

#ifndef _BFGS_H
#define _BFGS_H

#define MSWIND //Tells Matlab libraries we are using Windows
#define MSVC //Tells Matlab libraries we are using Visual C++
#include "matlab.hpp"

class BFGSMin {
public:
  BFGSMin( const int & numrows, const double & gtol, const double & tol,
            const double & alf, const double & sm,
            const int & im, const double & rt);
  virtual double mCompute(const mwArray & x) = 0;
  virtual double mComputeGrad(const mwArray & x, mwArray & dx) = 0;
  virtual double mFindMin(mwArray &x);
  int mLineSearch(const mwArray &x0, const mwArray &g, mwArray &p,
                   const double &f0,
                   mwArray &x, double &f);
protected:
  const int NumRows;
private:
  const double GTol;
  const double Tol;

```
Appendix B: Source Code

```c
const double Alf;
const double StepMax;
const int IterMax;
const double ResThr;
const double Eps;

mwArray g;
mwArray dg;
mwArray hdg;
mwArray H;
mwArray xnew;
mwArray p;
```

```c
#endif

///////////////////////////////////////////////// // constants.h // Constants header file
 //-----------------------------------------------------

#ifndef _CONSTANTS_H
#define _CONSTANTS_H

const double TPI=6.2831853071795864769252867665590057683943388;
const double PI=TPI/2.0;
const double C=299792458;
const double Deg2Rad=(TPI/360.0);
const double MHz2Hz=1.0e6;
const double Hz2RPS = TPI;
const double MHz2RPS = MHz2Hz*Hz2RPS;

#endif

///////////////////////////////////////////////// // coordsys.h // Coordinate system transformation header file
//-----------------------------------------------------

#ifndef _COORDSYS_H
#define _COORDSYS_H

#define MSWIND  //Tells Matlab libraries we are using Windows
#define MSVC    //Tells Matlab libraries we are using Visual C++
#include "matlab.hpp"

```

210
class CoordSys {
public:
    CoordSys(const double & originlat, const double & originlon,
             const double & originheight=0.0);
    ~CoordSys();
    int mGetENU(double & east, double & north, double & up,
                const double & lat, const double & lon, const double & height);
    int mGetGeo(double & lat, double & lon, double & height,
                const double & east, const double & north, const double & up);
    void mGetGeo(const mwArray & sv, mwArray & latlon);
private:
    double mGetRoc(double lat);
    int mGetECEF(mwArray & ecef, double lat, double lon, double h=0.0);
    const double OriginLat;
    const double OriginLon;
    const double OriginHeight;
    const double SMajorAxis;
    const double eSq;
    mwArray OriginECEF;
    mwArray Rotate2Loc;
};

#ifndef _DYNAMICS_H
#define _DYNAMICS_H

#include <vector>
#include <iostream>
#define MSWIND  //Tells Matlab libraries we are using Windows
#define MSVC    //Tells Matlab libraries we are using Visual C++
#include "matlab.hpp"
#include "kalman.h"

class MeasSetFD;

class Dynamics {
public:
    Dynamics(const mwArray & SVInit, const mwArray & SVRateInit);
    void mPrint(std::ostream & OutStream = std::cout) const;
    void mPredict(mwArray & PredSV, mwArray & PredSVSigma);
void mUpdate(const mwArray & SVIn, const double & AFVal, const int & NumUsed);
private:
    std::vector<Kalman> KFilts;

    mwArray AllPredSV;
    mwArray AllPredSVSigma;

    mwArray AllFiltSV;
    mwArray AllFiltSVSigma;
};
#endif

//////////////////////////////////////////////////
// garmin.h
//////////////////////////////////////////////////

//////////////////////////////////////////////////
// Garmin Mapsource header file
//////////////////////////////////////////////////

#ifndef _GARMIN_H
#define _GARMIN_H

#include <string>
#include <iostream>

class mwArray;
class MeasSetBase;

class Garmin
{
public:
    Garmin(const std::string fn);
    ~Garmin() {delete pFile;}  
mPrint(const mwArray & latlon, const MeasSetBase & msb);
    mPrint(const double & latd, const double & lond,
           const std::string & WName, const std::string & 
CommentString="");
    mGetDegMin(int & degs, double & mins, const double & ddegs);
private:
    std::ofstream * const pFile;
};
#endif

//////////////////////////////////////////////////
// kalman.h
//////////////////////////////////////////////////
// Kalman filter header file

#ifndef _KALMAN_H
#define _KALMAN_H

#define MSWIND    // Tells Matlab libraries we are using Windows
#define MSVC      // Tells Matlab libraries we are using Visual C++
#include "matlab.hpp"

class Kalman {
public:
    Kalman(const double & beta, const double & sigma, const double & dt,
            const double & pos, const double & vel, const double & acc,
            const double & possig, const double & velsig, const double & accsig );
    void Test(std::ostream & OutStream = std::cout);
    void mFilter(const double & Meas, const double & Sigma, mwArray & sv, mwArray & sigmaout);
    void mPredict(mwArray & sv, mwArray & sigma);
    Kalman & operator=(const Kalman & rhs);
private:
    const double Beta;
    const double Sigma;
    const double DeltaT;
    mwArray SDA;
    mwArray SDQ;
    mwArray SV;
    mwArray Obs;
    mwArray ObsT;
    mwArray ECov;
    mwArray KGain;
};

#endif

#ifndef _MATLABENG_H
#define _MATLABENG_H
#include <vector>
#define MSWIND    // Tells Matlab libraries we are using Windows
#endif

#include <vector>
#define MSWIND    // Tells Matlab libraries we are using Windows
#define MSVC    //Tells Matlab libraries we are using Visual C++
#include "engine.h"

class MatlabEng {
public:
    MatlabEng();
    ~MatlabEng();
    int mPlot(const std::vector<double> & x, const std::vector<double> & y, const int num=0);
    int mPut(const double * const pData, const int num=0);
    int mPut(const double * const pXin, const double * const pYin, const double * const pTin,
              const double * const pZin, const int m, const int n, const int p);
private:
    Engine *pEng;
};

#endif

/////////////////////////////////////////////////////////////////////
// meas.h
/////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////
// AM signal measurements header file
/////////////////////////////////////////////////////////////////////

#ifndef _MEAS_H
#define _MEAS_H

#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <vector>
#include <time.h>
#include ../include/constants.h

const int StationNum = 118;      // Total number of stations to
expect in input files
const int StationIdMin = 530;    // Minimum station id (nominal
frequency in kHz
const int StationIdSpacing = 10; // Station spacing
const double StationId2NomFreq = 0.001*MHz2Hz*Hz2RPS;  // Station
id to nominal frequency in radians/second
const int HarmonicNum = 84;      // Total number of harmonics to
expect in input files
const double HarmonicId2NomFreq = 0.020080321285141*MHz2Hz*Hz2RPS; // Harmonic id to nominal
frequency in radians/second

214
class MeasSetSS;

class MeasFile {
public:
  MeasFile(const std::string fn);
  ~MeasFile() { delete pFile; }
  bool mGetPos(double & lat, double & lon);
  bool mReadEpoch(MeasSetSS & mss);
  bool mCheckLineType(char lt) const;
  const double & mGetPulseTime() const { return PulseTime; }
  const double & mGetPulseTimeStamp() const { return PulseTimeStamp; }
private:
  std::ifstream * const pFile;
  bool HavePos;
  double Lat;
  double Lon;
  double PulseTime;
  double PulseTimeStamp;
};

class MeasSetBase {
public:
  MeasSetBase() : StartTime(0), EpochNum(0), TimeStamp(0.0) {} 
  int mPrintDateTime(std::ostream & OutStream = std::cout) const;
  const int & mGetStartTime() const { return StartTime; }
  const int & mGetEpochNum() const { return EpochNum; }
  const double & mGetTimeStamp() const { return TimeStamp; }
  time_t mGetTimeStampT() const { return ( (time_t) (TimeStamp + 0.5)); }
  double mGetElapsedTimeMin() const { return ( TimeStamp - ((double) StartTime) )/60.0; }
protected:
  int StartTime;
  int EpochNum;
  double TimeStamp;
private:
};

class MeasElemSS;

class MeasSetSS : public MeasSetBase {
public:
  MeasSetSS();
  int mLoadData(std::ifstream * pFile, const MeasFile & mf);
  int mPrint(std::ostream & OutStream = std::cout) const;
  int mPrintIds(std::ostream & OutStream) const;
  int mPrintFreqs(std::ostream & OutStream = std::cout) const;
  int mPrintMags(std::ostream & OutStream = std::cout) const;
  const MeasElemSS & mGetMeasSta(const int i) const { return StationData[i]; }
};
Appendix B: Source Code

```cpp
int mSetUse(const std::vector<int> & ids);
int mDontUse(const std::vector<int> & ids);
void mNeverReport(const std::vector<int> & ids);
protected:
    std::vector<MeasElemSS> StationData;
    std::vector<MeasElemSS> HarmData;
private:
};
class MeasElemFD;
class StationSet;

class MeasSetFD : public MeasSetBase {
public:
    MeasSetFD();
    int mLoadData(const MeasSetSS & lhs, const MeasSetSS & rhs);
    int mPrint(std::ostream & OutStream = std::cout) const;
    int mEstimateFreqRatio();
    int mInvalidateFR(const double & MaxFreqDevPpm);
    const MeasElemFD & mGetStaRef(const int & id) const;
    const MeasElemFD * mGetStaPtr(const int & id) const;
    double mGetClockUpdate() { return ClockOffsetRate*EpochPeriod; }  
    const double & mGetEpochPeriod() const {return EpochPeriod;}
    void mSetTimeStamp(const double & ts) { TimeStamp = ts; }
    void mDontUse(const std::vector<int> & ids);
    void mDontUseWeak(const double & ThreshDB);
protected:
    std::vector<MeasElemFD> Data;
    double EpochPeriod;  // In seconds
    double EstFreqRatio;
    double ClockOffsetRate; // In meters/second
private:
};
class MeasElemBase {
public:
    MeasElemBase(const int id, const double nf) : Id(id),
    NomFreq(nf), Use(false) {}
    MeasElemBase& operator=(const MeasElemBase & rhs);
    int mPrint(std::ostream & OutStream = std::cout) const;
    bool mGetUse() const {return Use;}
    const int & mGetId() const {return Id;}
    void mUse() { Use = true; }
    void mDontUse() { Use = false; }
protected:
    const int Id;
    const double NomFreq;  // Nominal frequency in radians/sec
    bool Use;
private:
};
class MeasElemSS : public MeasElemBase {
```
public:
    MeasElemSS(const int id, const double nf) : MeasElemBase(id, nf), Freq(0.0), Phase(0.0), Mag(0.0), Report(true) {
        MeasElemSS& operator=(const MeasElemSS & rhs);
        int mLoadData(std::ifstream * pFile);
        int mPrint(std::ostream & OutStream = std::cout) const;
        void mPrintId(std::ostream & OutStream = std::cout) const
            {OutStream << Id << ",";}
        void mPrintFreq(std::ostream & OutStream = std::cout) const
            {OutStream << std::setw(18) << std::fixed <<
                    std::setprecision(15) << Freq/TPI << ",";}
        void mPrintMag(std::ostream & OutStream = std::cout) const
            {OutStream << std::setw(18) << std::fixed <<
                    std::setprecision(15) << Mag << ",";}
        const double & mGetFreq() const {return Freq;}
        const double & mGetPhase() const {return Phase;}
        const double & mGetMag() const {return Mag;}
        bool mGetReport() const {return Report;}
        void mNeverReport() {Report = false;}
    protected:
        double Freq;  // Frequency in radians/sec
        double Phase; // Phase in radians
        double Mag;   // Magnitude in db
    private:
    Report;
};

class Station;

class MeasElemFD : public MeasElemBase {
public:
    MeasElemFD(const int id, const double nf) : MeasElemBase(id, nf),
        FreqBase(0.0), PhaseFD(0.0), MagRoam(0.0), MagMin(0.0),
        FreqRatio(0.0) {}
    MeasElemFD& operator=(const MeasElemFD & rhs);
    int mLoadData(const MeasElemSS & lhs, const MeasElemSS & rhs);
    int mPrint(std::ostream & OutStream = std::cout) const;
    const double & mGetFreqBase() const { return FreqBase; }
    const double & mGetFreqFD() const { return FreqRatio; }
    const double & mGetPhaseFD() const { return PhaseFD; }
    const double & mGetMagRoam() const { return MagRoam; }
    const double & mGetMagMin() const { return MagMin; }
    void mDontUse() { Use = false; }
    protected:
        double FreqBase; // Frequency reported by "base" receiver in radians/second
        double PhaseFD; // First difference phase in radians
        double MagRoam; // Magnitude reported by "roam" receiver in db
        double MagMin; // Minimum magnitude reported by either receiver in db
};
Appendix B: Source Code

double FreqRatio; // Frequency reported by "roam"
receiver/Frequency reported by "base" receiver
private:
);
#endif

/////////////////////////////////////////////////
// param.h
/////////////////////////////////////////////////

/////////////////////////////////////////////////
// Data processing parameters header file
/////////////////////////////////////////////////

 ifndef _PARAM_H
#define _PARAM_H

#include <string>
#include <fstream>
#include <iostream>
#include <vector>
class Param{
public:
 Param(const std::string fn);
 ~Param(){ delete pFile;}
 const int & mNumToSkip() const {return NumToSkip;}
 const int & mMaxNumEpoch() const {return MaxNumEpoch;}
 bool mUseMatlab() const {return UseMatlab;}
 const double & mMaxFreqDevPPM() const {return MaxFreqDevPPM;}
 int mElimBadMeas() const {return ElimBadMeas;}
 const double & mMinAF() const {return MinAF;}
 bool mDoSearch() const {return DoSearch;}
 const double & mSearchRangeMeters() const {return SearchRangeMeters;}
 const double & mClockSearchRangeMeters() const { return ClockSearchRangeMeters;}
 const double & mStepSizePercent() const {return StepSizePercent;}
 const std::vector<int> & mStationsNotUsed() const {return StationsNotUsed;}
 const std::vector<int> & mStationsNotReported() const {return StationsNotReported;}
private:
 std::ifstream * const pFile;
 int NumToSkip;
 int MaxNumEpoch;
 bool UseMatlab;
 double MaxFreqDevPPM;
 int ElimBadMeas;
 double MinAF;
bool DoSearch;
double SearchRangeMeters;
double ClockSearchRangeMeters;
double StepSizePercent;
std::vector<int> StationsNotUsed;
std::vector<int> StationsNotReported;
};

#endif

/////////////////////////////////////////////////////////////////////
// rateest.h
/////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////
// Rate estimation header file
/////////////////////////////////////////////////////////////////////

#ifndef _RATEEST_H
#define _RATEEST_H

#include <vector>
#define MSWIND    // Tells Matlab libraries we are using Windows
#define MSVC      // Tells Matlab libraries we are using Visual C++
#include "matlab.hpp"

class StationSet;
class MeasSetFD;
class Station;
class MeasElemFD;

class RateEst {
public:
    RateEst(const StationSet & ss, const MeasSetFD & msf);
    void mEstimateRates(mwArray & svin, mwArray & svdot);
    void mGetInvalidIds(std::vector<int> & ids, const double threshppm) const;
private:
    mwArray Obs;
    mwArray MeasVec;
    mwArray Weight;
    mwArray Residuals;
    std::vector<const Station *> Stas;
    std::vector<const MeasElemFD *> Meas;
};

#endif

/////////////////////////////////////////////////////////////////////
// station.h
/////////////////////////////////////////////////////////////////////
// Radio station data header file

#ifndef _TOWER_H
#define _TOWER_H

#include <string>
#include <iostream>
#include <vector>
#include <cmath>

const int MaxStations = 118;
const int MaxHarms = 84;
const int MaxNumTowers = 5;

class CoordSys;
class Station;

class StationSet {
public:
    StationSet(const std::string sfn,const std::string tfn,const std::string efn);
    int mSetENU(CoordSys & LCS);
    int mPrint(std::ostream& OutStream = std::cout) const;
    int mPrintNomFreqs(std::ostream& OutStream = std::cout) const;
    int mGetIds(std::vector<int> & ids) const;
    const Station & mGetStaRef(const int & id) const;
    const Station * mGetStaPtr(const int & id) const;
    void mCalcCenter();
private:
    std::vector<Station> Data;
};

class AntEx;
class TowerPos;

class Station {
public:
    Station(std::ifstream & File);
    int mGetId() const {return Id;}
    bool mGetUse() const {return Use;}
    int mLoadTPData(std::ifstream & File);
    int mLoadExData(std::ifstream & File);
    int mSetENU(CoordSys & LCS);
    int mPrint(std::ostream& OutStream = std::cout) const;
    void mPrintNomFreq(std::ostream& OutStream = std::cout) const
    {OutStream << Id << ",";}
    double mCalcThPhase(const double & east, const double & north,
                        const double & Wavenumber) const;

}
double mCalcThPhase(const double & east, const double & north, const double & Wavenumber, double & dwrteast, double & dwrtnorth) const;
void mCalcCenter();
void mCalcUnit(const double & east, const double & north, double & eastunit, double & northunit) const;
double mCalcDistToCenter(const double & east, const double & north) const
    {return sqrt( (CenterEast-east)*(CenterEast-east) + (CenterNorth-north)*(CenterNorth-north) );}
private:
    bool Use;
    int Id;
    int CurrentEx;
    int NumTowers;
    int NumEx;
    int Tod2Ex[3];
    double Power[3];
    double CenterEast;
    double CenterNorth;
    double GndFuncB;
    double GndFuncM;
    double GndFuncE;
    std::vector<AntEx> ExData;
    std::vector<TowerPos> TPData;
};

class TowerPos {
public:
    TowerPos(std::ifstream & File);
    int mSetENU(CoordSys & lcs);
    int mSetGeo(CoordSys & lcs);
    int mPrint(std::ostream& OutStream = std::cout) const;
    double mCalcDist(const double & ea, const double & no) const
        { return sqrt( (ea-East)*(ea-East) + (no-North)*(no-North) ); } // double mCalcDist(const double & ea, const double & no,
    double mCalcDist(const double & ea, const double & no, double & deltaeast, double & deltanorth) const;
    const double & mGetEast() const {return East;}
    const double & mGetNorth() const {return North;}
private:
    int Id;
    int TowerId;
    double Lat;
    double Lon;
    double Height;
    double East;
    double North;
    double Up;
};

class TowerEx;
class AntEx {
public:
    AntEx(std::ifstream & File, const int nt);
    int mPrint(std::ostream& OutStream = std::cout) const;
    const double & AntEx::mGetField(const int & id) const;
    const double & AntEx::mGetPhase(const int & id) const;
private:
    double Power;
    int NumTowers;
    std::vector<TowerEx> Data;
};

class TowerEx {
public:
    TowerEx(std::ifstream & File);
    int mPrint(std::ostream& OutStream = std::cout) const;
    const double & mGetField() const { return Field; }
    const double & mGetPhase() const { return Phase; }
private:
    int Id;
    int TowerId;
    int ExId;
    double Field;
    double Phase;
    double TowerHeight;
};

B.2.2 Source Files

#include <vector>
#define MSWIND //Tells Matlab libraries we are using Windows
#define MSVC    //Tells Matlab libraries we are using Visual C++
#include "matlab.hpp"
#include "windows.h"
#include "winbase.h"
#include "../include/constants.h"
#include "../include/coordsys.h"
#include "../include/garmin.h"
#include "../include/matlabeng.h"
#include "station.h"
#include "af.h"
#include "meas.h"
#include "afmax.h"
#include "param.h"
#include "dynamics.h"
#include "rateest.h"

using namespace std;

int main(int argc, char *argv[]) {

    // Input file names
    const string StationName("input/station.csv");
    const string TowerName("input/tower.csv");
    const string ExName("input/ex.csv");
    const string BaseFileName("input/base.txt");
    const string RoamFileName("input/roam.txt");
    const string ParamName("input/par.txt");
    const string PosTruthName("input/postruth.csv");

    // Input file streams
    std::ifstream PosTruth( PosTruthName.c_str() );

    // Output file names
    const string PosFileName("output/pos.csv");
    const string AFFileName("output/af.csv");
    const string WeightFileName("output/weight.csv");
    const string BaseFreqName("output/basefreq.csv");
    const string RoamFreqName("output/roamfreq.csv");
    const string BaseMagName("output/basemag.csv");
    const string RoamMagName("output/roammag.csv");
    const string PhaseResName("output/phaseres.csv");
    const string GarminFileName("output/garmin.wpt");

    // Output file streams
    std::ofstream PosFile(PosFileName.c_str(),std::ios::out|std::ios::trunc);
    std::ofstream AFFile(AFFileName.c_str(),std::ios::out|std::ios::trunc);
    std::ofstream WeightFile(WeightFileName.c_str(),std::ios::out|std::ios::trunc);
    std::ofstream BaseFreqFile(BaseFreqName.c_str(),std::ios::out|std::ios::trunc);
    std::ofstream RoamFreqFile(RoamFreqName.c_str(),std::ios::out|std::ios::trunc);
    std::ofstream BaseMagFile(BaseMagName.c_str(),std::ios::out|std::ios::trunc);
    std::ofstream RoamMagFile(RoamMagName.c_str(),std::ios::out|std::ios::trunc);
    std::ofstream PhaseResFile(PhaseResName.c_str(),std::ios::out|std::ios::trunc);
}

223
StationSet StaSet(StationName,TowerName,ExName);
MeasFile BaseFile(BaseFileName);
MeasFile RoamFile(RoamFileName);
Param RunParam(ParamName);
Garmin GFile(GarminFileName);

// Define a coordinate system centered on the stationary base
station position
// Express tower positions in this coordinate system
double latb,lonb;
BaseFile.mGetPos(latb,lonb);
CoordSys BaseENU(latb,lonb,54.72);
StaSet.mSetENU(BaseENU);
StaSet.mCalcCenter();

// Set up containers to hold raw measurement data
// Only use measurements from stations that we have info on
MeasSetSS Base;
MeasSetSS Roam;
std::vector<int> IdList;
StaSet.mGetIds(IdList);
Base.mSetUse(IdList);
Roam.mSetUse(IdList);
Base.mNeverReport(RunParam.mStationsNotReported());
Roam.mNeverReport(RunParam.mStationsNotReported());

// Container for first difference measurement data
MeasSetFD FDMeas;

// Class that uses observations of frequency to estimate SV
rates
RateEst Rates(StaSet,FDMeas);

// State variables
// RoamSV is the state of the roam receiver
// RoamSV: 1-east(meters) 2-north(meters) 3-clock
offset(meters)
// RoamSVDotInit: Used to initialize state vector rates based
on doppler
// PredRoamSV is the predicted state of RoamSV - used to
initialize search
// PredRoamSVSigma is the estimation error standard deviation
for PredRoamSV
mwArray RoamSV;
mwArray RoamSVDot;
mwArray PredRoamSV;
mwArray PredRoamSVSigma;
mwArray LatLon;
RoamSV = zeros(3,1);
RoamSVDot = zeros(3,1);
PredRoamSV = zeros(3,1);
PredRoamSVSigma = zeros(3,1);
LatLon = zeros(2,1);

RoamSV(3,1) = - (RoamFile.mGetPulseTime() -
BaseFile.mGetPulseTime() );
FDMeas.mSetTimeStamp( BaseFile.mGetPulseTimeStamp() );
AmbiguityFunc AF(StaSet,FDMeas,0.0,0.0);
AF.mNeverUse(RunParam.mStationsNotUsed());
AF.mNeverReport(RunParam.mStationsNotReported());
AFMax AFM(AF);

if( RunParam.mElimBadMeas() > 1 )
AF.mSetFBFExp( RunParam.mElimBadMeas() );

for(int i=0;i<RunParam.mNumToSkip();i++)
{
BaseFile.mReadEpoch(Base);
RoamFile.mReadEpoch(Roam);
PosTruth.ignore(1000,'\n');
}

// Use most recent "skipped" data to initialize roam receiver sv rates
FDMeas.mLoadData(Roam,Base);
Rates.mEstimateRates(RoamSV,RoamSVDot);

// Keep only the estimated clock rate - force position rates to zero
RoamSVDot(1,1) = 0.0;
RoamSVDot(2,1) = 0.0;

// Advance the clock estimate using the estimated clock rate
RoamSV(3,1) = RoamSV(3,1) + RoamSVDot(3,1) *
FDMeas.mGetEpochPeriod();

// The dynamics class is the true guardian of the roam receiver state vector
// RoamSV and RoamSVDot initialize the internal class states
// Statistics for these initializations are handled in the class
Dynamics Dyn(RoamSV,RoamSVDot);

double afval=0.0;

mwArray SrchRng;
SrchRng = zeros(3,1);

//Column headers for output files
Appendix B: Source Code

```cpp
AFFile << "Date, Time, Epoch Number, Elapsed Time (minutes),";
WeightFile << "Date, Time, Epoch Number, Elapsed Time (minutes),";
BaseFreqFile << "Date, Time, Epoch Number, Elapsed Time (minutes),";
RoamFreqFile << "Date, Time, Epoch Number, Elapsed Time (minutes),";
BaseMagFile << "Date, Time, Epoch Number, Elapsed Time (minutes),";
RoamMagFile << "Date, Time, Epoch Number, Elapsed Time (minutes),";
PhaseResFile << "Date, Time, Epoch Number, Elapsed Time (minutes),";
PosFile << "Date, Time, Epoch Number, Elapsed Time (minutes), East (meters), North (meters),";
PosFile << "Receiver Clock Offset (meters), Ambiguity Function Value, AF (meters),";
PosFile << "Number of signals used,"
PosFile << "Predicted East (meters), Predicted North (meters), Predicted Clock (meters),"
PosFile << "Predicted East Velocity (m/s), Predicted North Velocity (m/s), Predicted Clock Velocity (m/s),"
PosFile << "Predicted East Acceleration (m/s^2), Predicted North Acceleration (m/s^2), Predicted Clock Acceleration (m/s^2),"
PosFile << "Predicted Error StDev East (meters), Predicted Error StDev North (meters), Predicted Error StDev Clock (meters), Predicted Error StDev East Velocity (m/s), Predicted Error StDev North Velocity (m/s), Predicted Error StDev Clock Velocity (m/s),"
PosFile << "Filtered East (meters), Filtered North (m/s), Filtered Clock (m/s), Filtered Clock Velocity (m/s),"
PosFile << "Filtered East Velocity (m/s), Filtered North Velocity (m/s), Filtered Clock Velocity (m/s),"
PosFile << "Filtered East Acceleration (m/s^2), Filtered North Acceleration (m/s^2), Filtered Clock Acceleration (m/s^2),"
PosFile << "Filtered Error StDev East (meters), Filtered Error StDev North (meters), Filtered Error StDev Clock (meters), Filtered Error StDev East Velocity (m/s), Filtered Error StDev North Velocity (m/s), Filtered Error StDev Clock Velocity (m/s),"
PosFile << "Filtered Error StDev East Acceleration (m/s^2), Filtered Error StDev North Acceleration (m/s^2), Filtered Error StDev Clock Acceleration (m/s^2),"
PosFile << std::endl;
Base.mPrintIds(BaseFreqFile);
Roam.mPrintIds(RoamFreqFile);
Base.mPrintIds(BaseMagFile);
Roam.mPrintIds(RoamMagFile);
AF.mPrintIds(AFFile);
AF.mPrintIds(PhaseResFile);
```

226
// Matlab engine
MatlabEng * pme;
if( RunParam.mUseMatlab() )
{
pme = new MatlabEng;
}
else
pme = NULL;

int time;
double east;
double north;
double clock;
int j;

for(i=0;i<RunParam.mMaxNumEpoch();i++)
{
if( !( BaseFile.mReadEpoch(Base) && RoamFile.mReadEpoch(Roam) ) )
break;

Base.mPrintDateTime(BaseFreqFile);
Base.mPrintDateTime(RoamFreqFile);
Base.mPrintDateTime(BaseMagFile);
Base.mPrintDateTime(RoamMagFile);

Base.mPrintFreqs(BaseFreqFile);
Roam.mPrintFreqs(RoamFreqFile);
Base.mPrintMags(BaseMagFile);
Roam.mPrintMags(RoamMagFile);

FDMeas.mLoadData(Roam,Base);
Dyn.mPredict(PredRoamSV,PredRoamSVSigma);

Rates.mEstimateRates(PredRoamSV,RoamSVDot);
Rates.mGetInvalidIds(IdList,RunParam.mMaxFreqDevPPM());

FDMeas.mDontUseWeak(45.0);
AF.mInit();

afval = -1.0;

if(PosTruth)
{
PosTruth >> time;
PosTruth.ignore();
PosTruth >> east;
PosTruth.ignore();
PosTruth >> north;
PredRoamSV(1) = east;
PredRoamSV(2) = north;
PredRoamSVSigma(1) = 100.0;
PredRoamSVSigma(2) = 100.0;

if(PosTruth.peek() == ',')
{
    PosTruth.ignore();
    if(PosTruth.peek() != '\n')
    {
        PosTruth >> clock;
        PredRoamSV(3) = clock;
        PredRoamSVSigma(3) = 200.0;
    }
}

while( afval < RunParam.mMinAF() )
{
    RoamSV = PredRoamSV;

    AF.mNormalizeWeight("equal");
    if( RunParam.mDoSearch() )
    {
        if(pme)
        {
            SrchRng(1) = RunParam.mSearchRangeMeters();
            SrchRng(2) = SrchRng(1);
            SrchRng(3) = RunParam.mClockSearchRangeMeters();
        }
        afval = AF.mFindMin(RoamSV);
        AF.mFindMaxExh(RoamSV,SrchRng,RunParam.mStepSizePercent(),pme);
        afval = AF.mFindMin(RoamSV);
        if(afval < 0.8)
        {
            std::cout << afval << " " << RoamSV(1) << " " << RoamSV(2) << " " << RoamSV(3) << " " << AF.mGetNumUsed() << std::endl;
            SrchRng(1) = RunParam.mSearchRangeMeters();
            SrchRng(2) = SrchRng(1);
            SrchRng(3) = RunParam.mClockSearchRangeMeters();
            RoamSV = PredRoamSV;
            for(j=0;j<3;j++)
            {
                if( PredRoamSVSigma(j+1)*2.0 > SrchRng(j+1) )
                {
                    SrchRng(j+1) = PredRoamSVSigma(j+1)*2.0;
                }
            }
        }
    }
}
AFM.mFindAll(RoamSV, SrchRng, RunParam.mStepSizePercent());
AFM.mPickBest(RoamSV);
afval = AF(RoamSV);
}
}

afval = AF(RoamSV);

Base.mPrintDateTime(PosFile);
Base.mPrintDateTime(AFFile);
Base.mPrintDateTime(PhaseResFile);
Base.mPrintDateTime(WeightFile);

PosFile << setw(6) << fixed << setprecision(2) << RoamSV(1) << ",";
PosFile << setw(6) << fixed << setprecision(2) << RoamSV(2) << ",";
PosFile << setw(16) << fixed << setprecision(2) << RoamSV(3) << ",";
PosFile << setw(7) << fixed << setprecision(3) << afval << ",";
PosFile << setw(7) << fixed << setprecision(3) << 47.7*acos(afval) << ",";
PosFile << setw(4)<< AF.mGetNumUsed() << ",";

AF.mSetResiduals(RoamSV);
AF.mPrintResiduals(AFFile);
AF.mPrintPhaseResMeters(PhaseResFile);
AF.mPrintWeights(WeightFile);

std::cout << afval << " " << RoamSV(1) << " " << RoamSV(2) << 
" " << RoamSV(3) << " " << 
AF.mGetNumUsed() << std::endl;

if(RunParam.mElimBadMeas() == 1)
   AF.mDontUseOddMan();
else
   break;
}

BaseENU.mGetGeo(RoamSV, LatLon);
GFile.mPrint(LatLon, FDMeas);

Dyn.mUpdate(RoamSV, afval, AF.mGetNumUsed());
Dyn.mPrint(PosFile);
PosFile << std::endl;
PosTruth.ignore(1000, '\n');
}
if(pme)
    delete pme;

return EXIT_SUCCESS;
}

////////////////////////////////////////////////
// af.cpp
////////////////////////////////////////////////

////////////////////////////////////////////////
// Ambiguity function source file
////////////////////////////////////////////////

#include <vector>
#include <iostream>
#include <iomanip>
#include <algorithm>
#define MSWIND  //Tells Matlab libraries we are using Windows
#define MSVC    //Tells Matlab libraries we are using Visual C++
#include "matlab.hpp"
#include "..\include\constants.h"
#include "station.h"
#include "meas.h"
#include "af.h"

using namespace std;

AmbiguityFunc::AmbiguityFunc(const StationSet & ss, const
  MeasSetFD & ms,
  const double & baseeast, const double & basenorth)
  : FBFExp(0), NumUsed(0), BaseEast(baseeast),
    BaseNorth(basenorth)
{
    std::vector<int> ids;

    ss.mGetIds(ids);

    Elements.reserve( ids.size() );
    Elements.clear();

    for(int i=0;i<ids.size();i++)
    {
        Elements.push_back(
            AFElem(ss.mGetStaRef(ids[i]),ms.mGetStaRef(ids[i]) ) );
    }
}

int AmbiguityFunc::mGetResCnt(const double & ThresholdMeters) const
{ int count = 0;
    for(int i=0;i<Elements.size();i++)
    {
        if( Elements[i].mGetResidualMeters() < ThresholdMeters )
        { count++;
        }
        return count;
    }
}

int AmbiguityFunc::mPrint(std::ostream& OutStream) const
{
    for(int i=0;i<Elements.size();i++)
    {
        Elements[i].mPrint(OutStream);
    }
    return EXIT_SUCCESS;
}

void AmbiguityFunc::mPrintIds(std::ostream& OutStream) const
{
    for(int i=0;i<Elements.size();i++)
    {
        Elements[i].mPrintId(OutStream);
    }
    OutStream << endl;
}

int AmbiguityFunc::mPrintPhaseResMeters(std::ostream& OutStream) const
{
    for(int i=0;i<Elements.size();i++)
    {
        Elements[i].mPrintPhaseResMeters(OutStream);
    }
    OutStream << endl;
    return EXIT_SUCCESS;
}

int AmbiguityFunc::mPrintResiduals(std::ostream& OutStream) const
{
    for(int i=0;i<Elements.size();i++)
    {
        Elements[i].mPrintResidual(OutStream);
    }
    OutStream << endl;
    return EXIT_SUCCESS;
}

int AmbiguityFunc::mPrintWeights(std::ostream& OutStream) const
{
Appendix B: Source Code

```cpp
for(int i=0;i<Elements.size();i++)
{
    Elements[i].mPrintWeights(OutStream);
} 
OutStream << endl;
return EXIT_SUCCESS;
}

int AmbiguityFunc::mDontUseOddMan(const int & num)
{
    for(int i=0;i<num;i++)
    {
        std::min_element(Elements.begin(),Elements.end())->mDontUse();
    } 
    NumUsed -= num;
    return EXIT_SUCCESS;
}

void AmbiguityFunc::mInit()
{
    NumUsed = 0;
    for(int i=0;i<Elements.size();i++)
    {
        if( Elements[i].mInit(BaseEast,BaseNorth) )
            NumUsed++;
    }
}

void AmbiguityFunc::mNeverUse(const std::vector<int> & ids)
{
    int i=0;
    int j=0;

    for(j=0;j<ids.size();j++)
    {
        for(i=0;i<Elements.size();i++)
        {
            if( Elements[i].mGetId() == ids[j] )
            {
                Elements[i].mNeverUse();
                break;
            }
        }
    }
}

void AmbiguityFunc::mNeverReport(const std::vector<int> & ids)
{
    int i=0;
    int j=0;

    for(j=0;j<ids.size();j++)
    {
        for(i=0;i<Elements.size();i++)
        {
            if( Elements[i].mGetId() == ids[j] )
            {
                Elements[i].mNeverReport();
                break;
            }
        }
    }
}
```

232
for(j=0;j<ids.size();j++)
{
    for(i=0;i<Elements.size();i++)
    {
        if( Elements[i].mGetId() == ids[j] )
        {
            Elements[i].mNeverReport();
            break;
        }
    }
}

int AmbiguityFunc::mNormalizeWeight(const std::string & method)
{
    int count=0;

    if(method=="equal")
    {
        for(int i=0;i<Elements.size();i++)
        {
            if( Elements[i].mGetUse() )
            {
                count ++;
            }
        }
        for(i=0;i<Elements.size();i++)
        {
            if( Elements[i].mGetUse() )
                Elements[i].mSetWeight(1.0/( (double) count ));
            else
                Elements[i].mSetWeight(0.0);
        }
        return EXIT_SUCCESS;
    }

    if( ( method=="magmin" ) || ( method=="magroam" ) )
    {
        double min=100000.0;
        double sum=0.0;
        double temp=0.0;;
        for(int i=0;i<Elements.size();i++)
        {
            if( Elements[i].mGetUse() )
                {count++;
                    if( method=="magmin" )
                        temp = Elements[i].mGetMagMin();
                    else
                        temp = Elements[i].mGetMagMin();}
        }
    }
}
temp = Elements[i].mGetMagRoam();

  sum += temp;
  if(temp < min)
    min = temp;
}

//Make reference magnitude 1db below minimum magnitude
min -= 1.0;

  sum = sum - ( min*count ) ;

for(i=0;i<Elements.size();i++)
{
  if( Elements[i].mGetUse() )
  {
    if( method=="magmin" )
      Elements[i].mSetWeight( (Elements[i].mGetMagMin() -
                           min)/sum );
    else
      Elements[i].mSetWeight( (Elements[i].mGetMagRoam() -
                           min)/sum );
  }
  else
  Elements[i].mSetWeight( 0.0 );
}

return EXIT_SUCCESS;
}

std::cerr << "Error: AmbiguityFunc::mNormalizeWeight()." <<
  std::endl;
std::cerr << "Invalid method: " << method << "." << std::endl;
exit(EXIT_FAILURE);
}

void AmbiguityFunc::mSetResiduals(const mwArray & sv)
{
  for(int i=0;i<Elements.size();i++)
  {
    Elements[i].mSetResidual(sv);
  }
}

double AmbiguityFunc::operator()( const mwArray & sv )
{
  double sum = 0.0;
  for(int i=0;i<Elements.size();i++)
  {
    if( Elements[i].mGetUse() )
    {
double AmbiguityFunc::operator()( const mwArray & sv, mwArray & gradf )
{
    double sum = 0.0;
    mwArray gradfi;

    gradf = zeros(3,1);
    gradfi = zeros(3,1);

    for(int i=0;i<Elements.size();i++)
    {
        if( Elements[i].mGetUse() )
        {
            sum += Elements[i](sv,FBFExp,gradfi);
            gradf = gradf + gradfi;
        }
    }

    return sum;
}

AFElem::AFElem(const Station & sta, const MeasEle& meas) :
Meas(meas), Sta(sta),
    Use(false), BaseThPhase(0.0), Wavenumber(0.0), Residual(0.0),
    PhaseRes(0.0),
    Weight(0.0), NeverUse(false), NeverReport(false)
{
    Weight = meas.mGetMagRoam();
}

int AFElem::mPrint(std::ostream & OutStream) const
{
    char sep=';';
    if(OutStream == std::cout)
        sep = ' ';

    Meas.mPrint();
    OutStream << std::setw(3) << Use << sep;
    OutStream << std::setw(20) << std::setprecision(10) << Weight <<
    std::endl;
    Sta.mPrint();
    return EXIT_SUCCESS;
}
int AFElem::mPrintId(std::ostream& OutStream) const
{
    if(!NeverReport)
        OutStream << Meas.mGetId() << ",";
    return EXIT_SUCCESS;
}

int AFElem::mPrintResidual(std::ostream& OutStream) const
{
    if(!NeverReport)
        OutStream << setw(7) << fixed << setprecision(4) << Residual << ",";
    return EXIT_SUCCESS;
}

int AFElem::mPrintPhaseResMeters(std::ostream& OutStream) const
{
    if(!NeverReport)
        OutStream << setw(7) << fixed << setprecision(4) << PhaseRes/Wavenumber << ",";
    return EXIT_SUCCESS;
}

int AFElem::mPrintWeights(std::ostream& OutStream) const
{
    if(!NeverReport)
        OutStream << setw(7) << fixed << setprecision(4) << Weight << ",";
    return EXIT_SUCCESS;
}

bool AFElem::operator<(AFElem & rhs) const
{
    if(Use)
    {
        if(rhs.Use)
            return (Residual < rhs.Residual);
        else
            return true;
    }else{
        if(rhs.Use)
            return false;
        else
            return (Residual < rhs.Residual);
    }
}

bool AFElem::mInit(const double & baseeast, const double & basenorth)
B.2 Navigation Algorithm Program

Use = Meas.mGetUse() && Sta.mGetUse() && (! NeverUse);
Wavenumber = ( Meas.mGetFreqBase() )/C;
BaseThPhase = Sta.mCalcThPhase(baseeast,basenorth,Wavenumber);

return Use;
}

void AFElem::mSetResidual(const mwArray & sv)
{
    PhaseRes = Meas.mGetPhaseFD() -
                ( Sta.mCalcThPhase(sv(1),sv(2),Wavenumber ) + Wavenumber*sv(3)
                          - BaseThPhase );

    //Make sure phaseres is closest cycle to zero
    PhaseRes = fmod(PhaseRes,TPI);
    if(PhaseRes > PI)
        PhaseRes -= TPI;
    if(PhaseRes < -PI)
        PhaseRes += TPI;

    Residual = cos(PhaseRes);
}

AFElem& AFElem::operator=(const AFElem & rhs)
{
    if (this == &rhs) return *this;

    std::cerr << "Error: AFElem::operator=()!" << std::endl;
    std::cerr << "Operator= should not be used for this class!" << std::endl;

    Weight = rhs.Weight;
    Use = rhs.Use;

    return *this;
}

double AFElem::operator()( const mwArray & sv, const int & fbfexp)
{
    PhaseRes = Meas.mGetPhaseFD() -
                ( Sta.mCalcThPhase(sv(1),sv(2),Wavenumber ) + Wavenumber*sv(3)
                          - BaseThPhase );

    //Make sure phaseres is closest cycle to zero
    PhaseRes = fmod(PhaseRes,TPI);
    if(PhaseRes > PI)
        PhaseRes -= TPI;
    if(PhaseRes < -PI)
        PhaseRes += TPI;

    Residual = cos(PhaseRes);
if(fbfexp==0)  
    return ( Weight * Residual);
else  
    return ( Weight * pow(0.5+0.5*Residual,(double) fbfexp) );

}  

double AFElem::operator()( const mwArray & sv, const int & fbfexp,  
mwArray & gradf)  
{  
double dfdx = 0.0;  
double dfdy = 0.0;  
double f = 0.0;  

    PhaseRes = Meas.mGetPhaseFD() -  
        ( Sta.mCalcThPhase( sv(1),sv(2),Wavenumber,dfdx,dfdy )  
        + Wavenumber*sv(3) - BaseThPhase );  

    //Make sure phaseres is closest cycle to zero  
    PhaseRes = fmod(PhaseRes,TPI);  
    if(PhaseRes > PI)  
        PhaseRes -= TPI;  
    if(PhaseRes < -PI)  
        PhaseRes += TPI;  

    const double wsinval = Weight*sin(PhaseRes);  

    Residual = cos(PhaseRes);  

    f = Weight*Residual;  

    gradf(1) = wsinval*dfdx;  
    gradf(2) = wsinval*dfdy;  
    gradf(3) = wsinval*Wavenumber;  

    if(fbfexp != 0)  
    {  
        double fnml = 0.0;  

        f /= Weight;  
        fnml = pow(f,fbfexp-1);  
        gradf = gradf*( fbfexp*fnml );  
        f = Weight*f*fnml;  
    }  

    return f;  
}  

///////////////////////////////////////////////////////////////////////////////////////////  
// afmax.cpp
#include <iostream>
#include <iomanip>

#define MSWIND    //Tells Matlab libraries we are using Windows
#define MSVC      //Tells Matlab libraries we are using Visual C++
#include "matlab.hpp"
#include "afmax.h"

using namespace std;

AFMax::AFMax(AmbiguityFunc & af, const double sf)
    : BFGSMin(3,1.0e-6*fabs(sf),1.0e-4,100.0,200,20.0),
      AF(af), ScaleFactor(sf),
      MaxNumMin(1000), NumBestMin(10), NumMin(0),
      MaxFuncValToCalc(0.2), MinSpToCalcM(10.0)
    {
      Mins = zeros(NumRows+2,MaxNumMin);
      BestMins = zeros(NumRows+2,NumBestMin);
    }

int AFMax::mPrintBestMins(std::ostream& OutStream) const
    {
      for(int i=0;i<NumBestMin;i++)
        {
          OutStream << ",,,,,,;
          OutStream << BestMins(1,i+1) << "," << BestMins(2,i+1) << "," << BestMins(3,i+1) << ",";
          OutStream << BestMins(4,i+1) << "," << BestMins(5,i+1) << endl;
        }
      return EXIT_SUCCESS;
    }

int AFMax::mPrintLocHeader(std::ostream& OutStream) const
    {
      double thr;
      OutStream << "Date,Time,Epoch Number,Elapsed Time (minutes),";
      OutStream << "East (meters),North (meters),Receiver Clock Offset (meters),Ambiguity Function Value,AF (percent),";
      OutStream << "Number of Signals Used in Search,"
      for(int j=0;j<6;j++)
        {
          OutStream <<",",
        }
    }
thr = ((double) (j+1))*10.0;
OutStream << "Number of residuals < " << thr << " meters,"
OutStream << endl;
return EXIT_SUCCESS;
}

int AFMax::mPrintMins(std::ostream& OutStream, int num) const
{

double GloMinVal;
double MinVal;
double thr;
int i;
int j;

if(NumMin < num)
    num = NumMin;

GloMinVal = AF( Mins(colon(1,3),1) );

for(i=0;i<num;i++)
{
    MinVal = AF( Mins(colon(1,3),i+1) );

    if(i)
        OutStream << ",,,,"
    OutStream << Mins(1,i+1) << "," << Mins(2,i+1) << "," << Mins(3,i+1) << ",";
    OutStream << MinVal << ",";
    OutStream << MinVal/GloMinVal << "," << Mins(5,i+1) << ",";
    for(j=0;j<6;j++)
    {
        thr = ((double) (j+1))*10.0;
        OutStream << AF.mGetResCnt( thr ) << ",";
    }
    OutStream << endl;
}
return EXIT_SUCCESS;
}

int AFMax::mPrintMinsNormTable(std::ostream& OutStream) const
{

    int i,j;
mwArray NormTable;

    NormTable = zeros(NumMin+1,NumMin+1);

    for(i=0;i<NumMin;i++)
    {
        NormTable(1,i+2) = Mins(4,i+1);
        NormTable(i+2,1) = Mins(4,i+1);
    
}
for(i=0;i<NumMin;i++)
{
    for(j=0;j<i;j++)
    {
        NormTable(i+2,j+2) = norm(Mins( colon(1,3),i+1 ) - Mins( colon(1,3),j+1 ) );
        NormTable(j+2,i+2) = NormTable(i+2,j+2);
    }
    NormTable(i+2,i+2) = 20000.0;
}

OutStream << std::fixed << std::setprecision(3);
for(i=0;i<NumMin+1;i++)
{
    for(j=0;j<NumMin+1;j++)
    {
        OutStream << NormTable(i+1,j+1) << "",
    }
    OutStream << endl;
}

return EXIT_SUCCESS;
}

bool AFMax::mTooCloseToMin(const mwArray & point, const double & threshold) const
{
    bool TooClose=false;

    for(int i=0;i<NumMin;i++)
    {
        if( norm(point-Mins(colon(1,3),i+1)) < threshold)
        {
            TooClose = true;
            break;
        }
    }
    return TooClose;
}

bool AFMax::mAddMin(const mwArray & point)
{
    if( mTooCloseToMin(point,1.0) )
        return false;
    else
    {
        Mins( colon(1,3),NumMin+1 ) = point;
        Mins(4,NumMin+1) = AF(point);
Appendix B: Source Code

```cpp
Mins(5,NumMin+1) = AF.mGetNumUsed();
NumMin++;
return true;
}
}

double AFMax::mComputeGrad(const mwArray & x, mwArray & dx)
{
    double afv = 0.0;
    afv = AF(x, dx);
    dx = dx * ScaleFactor;
    return afv*ScaleFactor;
}

int AFMax::mFindAll(const mwArray & Center, const mwArray & Range, const double StepPercent)
{
    int i,j,k;
    int istop,jstop,kstop;
    double x,y,t;
    double xstart,ystart,tstart;
    double xstop,ystop,tstop;
    double StepM;
    mwArray Test;
    Test = zeros(3,1);
    StepM = StepPercent*C/1.7e6;

    //Calculate starting values
    xstart = Center(1) - Range(1)/2.0;
    ystart = Center(2) - Range(2)/2.0;
    tstart = Center(3) - Range(3)/2.0;

    //Calculate number of steps required
    istop = ((int) (Range(1)/StepM)) + 2;
    jstop = ((int) (Range(2)/StepM)) + 2;
    kstop = ((int) (Range(3)/StepM)) + 2;

    //Calculate actual stopping values
    xstop = xstart + ((double) istop-1.0)*StepM;
    ystop = ystart + ((double) jstop-1.0)*StepM;
    tstop = tstart + ((double) kstop-1.0)*StepM;

    std::cout << "X range: " << xstart << " to " << xstop << std::endl;
```

242
B.2 Navigation Algorithm Program

```cpp
std::cout << "Y range: " << ystart << " to " << ystop << std::endl;
std::cout << "T range: " << tstart << " to " << tstop << std::endl;

//Calculate total number of steps required
std::cout << "Search requires " << (istop*jstop*kstop) << " steps." << std::endl;

double FuncVal = 0.0;
NumMin = 0;
t=tstart;
for(k=0;k<kstop;k++)
{
x=xstart;
for(i=0;i<istop;i++)
{
y=ystart;
for(j=0;j<jstop;j++)
{
    Test(1)=x;
    Test(2)=y;
    Test(3)=t;
    FuncVal = AF(Test);
    if(FuncVal < MaxFuncValToCalc)
    {
        FuncVal = mFindMin(Test);
        mAddMin(Test);
    }
    y += StepM;
}
x += StepM;
}
t += StepM;
return EXIT_SUCCESS;
}

double AFMax::mFindMaxExh(mwArray & Center, const mwArray & Range,
        const double StepPercent, MatlabEng * me)
{
    int i,j,k;
    int istop,jstop,kstop;
    double x,y,t;
    double xstart,ystart,tstart;
    double xstop,ystop,tstop;
    double StepM;
```
Appendix B: Source Code

mwArray Test;

Test = zeros(3,1);

StepM = StepPercent*C/1.7e6;

// Calculate starting values
xstart = Center(1) - Range(1)/2.0;
ystart = Center(2) - Range(2)/2.0;
tstart = Center(3) - Range(3)/2.0;

// Calculate number of steps required
istop = ((int) (Range(1)/StepM)) + 2;
jstop = ((int) (Range(2)/StepM)) + 2;
kstop = ((int) (Range(3)/StepM)) + 2;

// Calculate actual stopping values
xstop = xstart + ((double) istop-1.0)*StepM;
ystop = ystart + ((double) jstop-1.0)*StepM;
tstop = tstart + ((double) kstop-1.0)*StepM;

std::cout << "X range: " << xstart << " to " << xstop << std::endl;
std::cout << "Y range: " << ystart << " to " << ystop << std::endl;
std::cout << "T range: " << tstart << " to " << tstop << std::endl;

// Calculate total number of steps required
std::cout << "Search requires " << (istop*jstop*kstop) << " steps." << std::endl;

// Matlab output
double *pData;
if(me)
{
    pData = new double[istop*jstop*kstop];
}

double FuncVal = 0.0;
double MaxFuncVal = -1.0;

NumMin = 0;

t=tstart;
for(k=0;k<kstop;k++)
{
    x=xstart;
    for(i=0;i<istop;i++)
    {
        y=ystart;
        for(j=0;j<jstop;j++)
{ 
    Test(1) = x; 
    Test(2) = y; 
    Test(3) = t; 
    FuncVal = AF(Test); 
    if(me) 
        ( *(pData + (k*istop + i)*jstop + j) ) = FuncVal; 
    
    if(FuncVal > MaxFuncVal) 
    { 
        MaxFuncVal = FuncVal; 
        Center = Test; 
    } 
    
    y += StepM; 
} 
    x += StepM; 
} 
    t += StepM; 
} 
if(me) 
{ 
    double * pTRamp; 
    double * pXRamp; 
    double * pYRamp; 

    pTRamp = new double[kstop]; 
    pXRamp = new double[istop]; 
    pYRamp = new double[jstop]; 

    t = tstart; 
    for(k=0;k<kstop;k++) 
    { 
        (*(pTRamp+k)) = t/C; 
        t += StepM; 
    } 
    x = xstart; 
    for(i=0;i<istop;i++) 
    { 
        (*(pXRamp+i)) = x; 
        x += StepM; 
    } 

    y = ystart; 
    for(j=0;j<jstop;j++) 
    { 
        (*(pYRamp+j)) = y; 
        y += StepM; 
    } 

    me->mPut(pXRamp,pYRamp,pTRamp,pData,istop,jstop,kstop);
std::cout << "Press any key to continue!" << endl; 
fgetc(stdin);

delete pData;
delete pTRamp;
delete pXRamp;
delete pYRamp;
}

return MaxFuncVal;
}

int AFMax::mOddManBestMins()
{
    mwArray test;
    int i;

    test = zeros(1,3);

    for(i=0;i<NumBestMin;i++)
    {
        AF.mInit();
        while( ( BestMins(4,i+1) ) < 0.95 )
        {
            test = BestMins( colon(1,3),i+1 );

            AF.mSetResiduals( test );
            AF.mDontUseOddMan(1);
            AF.mNormalizeWeight();

            BestMins(4,i+1) = mFindMin(test)/ScaleFactor;
            BestMins( colon(1,3),i+1 ) = test;
            BestMins(5,i+1) = AF.mGetNumUsed();
        }
    }
    return EXIT_SUCCESS;
}

int AFMax::mPickBest(mwArray & sv)
{
    mSortMinsAF();
    sv = Mins(colon(1,3),1);

    return EXIT_SUCCESS;
}

int AFMax::mPickBOTB(mwArray & Best)
{
    mSortBestMinsNumUsed();
    Best = BestMins( colon(1,3), NumBestMin );
return EXIT_SUCCESS;
}

/////////////////////////////////////////////////
// bfgs.cpp
/////////////////////////////////////////////////

/////////////////////////////////////////////////
// BFGS minimization source file
/////////////////////////////////////////////////

#include <iostream>
#include <iomanip>
#include <limits>
#define MSWIND    //Tells Matlab libraries we are using Windows
#define MSVC     //Tells Matlab libraries we are using Visual C++
#include "matlab.hpp"
#include "bfgs.h"

using namespace std;

BFGSMin::BFGSMin( const int & numrows, const double & gtol, const
double & tol,
    const double & alf, const double & sm, const int & im, const
double & rt)
    : NumRows(numrows), GTol(gtol), Tol(tol), Alf(alf),
        StepMax(sm), IterMax(im), ResThr(rt),
        Eps(std::numeric_limits<double>::epsilon())
{
    g = zeros(NumRows,1);
    dg = zeros(NumRows,1);
    hdg = zeros(NumRows,1);
    H = zeros(NumRows,NumRows);
    xnew = zeros(NumRows,1);
    p = zeros(NumRows,1);
}

int BFGSMin::mLineSearch(const mwArray &x0, const mwArray &g,
mwArray &p, const double &f0,
    mwArray &x, double &f)
{
    bool firstpass=true;

double a;
    double b;
    double lam1;
    double lam2;
    double lammin;
    double lamtmp;
    double rootarg;

double f2;
double rv1;
double rv2;
double slope;
double normp;

normp = norm(p);
if(normp > StepMax)
    p = p*(StepMax/normp);

slope = sum(times(g,p));

if(slope > 0.0)
{
    std::cout << "Roundoff problem in LineSearch - Slope (=" <<
    slope << ") positive!" << endl;
}

lammin =
Tol/max(abs(rdivide(transpose(p),max(vertcat(abs(transpose(x0)),o
nes(size(transpose(x0)))))),));
lam1 = 1.0;
for(int i=0;i<10000;i++)
{
    x = x0 + lam1*p;
    f = mCompute(x);
    if(lam1 < lammin)
    {
        x = x0;
        break;
    }
    else
    {
        if(f <= f0 + Alf*lam1*slope)
        {
            break;
        }
        else
        {
            if(firstpass)
            lamtmp = -slope/(2.0*(f-f0-slope));
            else
            {
                rv1 = f-f0-lam1*slope;
                rv2 = f2-f0-lam2*slope;
                a = (rv1/(lam1*lam1)-rv2/(lam2*lam2))/(lam1-lam2);
                b = ((-lam2*rv1)/(lam1*lam1)+(lam1*rv2)/(lam2*lam2))/(lam1-
            lam2);
                if(a==0.0)
                    lamtmp = -slope/(2.0*b);
                else
                {
                    }
rootarg = b*b - 3.0*a*slope;
if(rootarg < 0.0)
    lamtmp = 0.5 * lam1;
else
    {
        if(b <= 0.0)
            lamtmp = (-b+sqrt(rootarg))/(3.0*a);
        else
            lamtmp = -slope/(b+sqrt(rootarg));
    }
}
if(lamtmp > 0.5*lam1)
    lamtmp = 0.5*lam1;
}
lam2 = lam1;
f2 = f;
firstpass = false;
if(lamtmp > 0.1*lam1)
    lam1 = lamtmp;
else
    lam1 = 0.1*lam1;
}
return EXIT_SUCCESS;

double BFGSMin::mFindMin(mwArray &x)
{
    int num;
    int numtot;
    bool gtolend = false;
    bool tolend = false;
    double fac;
    double fad;
    double fae;
    double f;
    double fret;
    double sumdg;
    double sump;
    double test;
    H = eye(NumRows,NumRows);
    f = mComputeGrad(x,g);
    p = -g;
    numtot = 0;
    for(num=0;num<IterMax;num++)
    {
        mLineSearch(x,g,p,f,xnew,fret);
f = fret;
p = xnew-x;
x = xnew;

test = norm(p,'inf');
if(test < Tol)
{
    tolend = true;
    break;
}

dg = g;
mComputeGrad(x,g);

test = norm(g,'inf');
if(test < GTol)
{
    gtolend = true;
    break;
}

dg = g-dg;
hdg = mtimes(H,dg);

fac = sum(times(dg,p));
fae = sum(times(dg,hdg));
sumdg = sum(times(dg,dg));
sump = sum(times(p,p));

if( fac > sqrt(Eps*sumdg*sump) )
{
    fac = 1.0/fac;
    fad = 1.0/fae;

    dg = fac*p-fad*hdg;
    H = H + fac*mtimes(p,transpose(p)) -
    fad*mtimes(hdg,transpose(hdg));
    H(1,2) = H(2,1);
    H(1,3) = H(3,1);
    H(2,3) = H(3,2);
}

p = -mtimes(H,g);

//Start Over ????
if(num > 15)
{
    H = eye(NumRows,NumRows);
    f = mComputeGrad(x,g);
    p = -g;
    num = 0;
B.2 Navigation Algorithm Program

}
numtot++;

}
return fret;
}

zahl

/////////////////////////////////////////////////
// coordsys.cpp
/////////////////////////////////////////////////

/////////////////////////////////////////////////
// Coordinate system transformation source file
/////////////////////////////////////////////////

#include <cmath>
#include "../include/constants.h"
#include "../include/coordsys.h"

CoordSys::CoordSys(const double & originlat,
const double & originlon,
const double & originheight):
OriginLat(originlat),
OriginLon(originlon),
OriginHeight(originheight),
SMajorAxis(6378137.0),
eSq(2.0/298.257223563 -
(1.0/298.257223563)*(1.0/298.257223563))
{
OriginECEF = zeros(3,1);
Rotatet2Loc = zeros(3,3);
mGetECEF(OriginECEF,OriginLat,OriginLon,OriginHeight);

Rotatet2Loc(1,1) = -sin(OriginLon);
Rotatet2Loc(1,2) = cos(OriginLon);
Rotatet2Loc(1,3) = 0.0;
Rotatet2Loc(2,1) = -sin(OriginLat)*cos(OriginLon);
Rotatet2Loc(2,2) = -sin(OriginLat)*sin(OriginLon);
Rotatet2Loc(2,3) = cos(OriginLat);
Rotatet2Loc(3,1) = cos(OriginLat)*cos(OriginLon);
Rotatet2Loc(3,2) = cos(OriginLat)*sin(OriginLon);
Rotatet2Loc(3,3) = sin(OriginLat);
}

int CoordSys::mGetENU(double & east, double & north, double & up,
const double & lat, const double & lon, const double & height)
{

Appendix B: Source Code

mwArray temp;

temp = zeros(3,1);

mGetECEF(temp,lat,lon,height);

temp = temp - OriginECEF;
temp = Rotate2Loc*temp;

east = temp(1);
north = temp(2);
up = temp(3);

return EXIT_SUCCESS;
}

void CoordSys::mGetGeo(const mwArray & sv,mwArray & latlon) {

double east;
double north;
double up;

double lat;
double lon;
double height;

east = sv(1);
north = sv(2);
up = 0.0;

CoordSys::mGetGeo(lat,lon,height,east,north,up);

latlon(1) = lat;
lattlon(2) = lon;
}

int CoordSys::mGetGeo(double & lat, double & lon, double & height,
const double & east, const double & north, const double & up) {
mwArray temp;

temp = zeros(3,1);

temp(1) = east;
temp(2) = north;
temp(3) = up;

temp = mldivide(Rotate2Loc,temp);
temp = temp + OriginECEF;

double x=temp(1);
double y=temp(2);
double z=temp(3);
double rho;
double roc;
double infnorm;

mwArray TestECEF;

TestECEF = zeros(3,1);

//Geodetic longitude does not require iteration
lon = atan2(y,x);

//Initialize geodetic latitude and height to that of the origin
lat = OriginLat;
height = OriginHeight;

infnorm = 1.0;

while(infnorm > 1.0e-7) //About a centimeter of latitude
{
    rho = sqrt(x*x+y*y);
    roc = mGetRoc(lat);

    height = ( rho/cos(lat) ) - roc;
    lat = atan2(z/rho,1-eSq*(roc/(roc+height)));

    mGetECEF(TestECEF,lat,lon,height);
    TestECEF = TestECEF - temp;
    TestECEF(3) = TestECEF(3)*1.0e-5;

    infnorm = norm(TestECEF,"inf");
}

return EXIT_SUCCESS;

}

double CoordSys::mGetRoc(double lat)
{
    return SMajorAxis/sqrt( 1.0 - eSq*sin(lat)*sin(lat) );
}

int CoordSys::mGetECEF(mwArray & ecef, double lat, double lon,
double h)
{
    double Roc;
    Roc = mGetRoc(lat);
ecef(1) = (Roc+h)*cos(lat)*cos(lon);
ecef(2) = (Roc+h)*cos(lat)*sin(lon);
ecef(3) = ((1-eSq)*Roc+h)*sin(lat);

return EXIT_SUCCESS;
}

////////////////////////////////////////////////////////////////////////////////
// dynamics.cpp
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// Receiver dynamics source file
////////////////////////////////////////////////////////////////////////////////

#include <iostream>
#include <iomanip>
#define MSWIND //Tells Matlab libraries we are using Windows
#define MSVC    //Tells Matlab libraries we are using Visual C++
#include "matlab.hpp"
#include "dynamics.h"
#include "meas.h"
#include "kalman.h"

using namespace std;

Dynamics::Dynamics(const mwArray & SVInit,const mwArray & SVRateInit)
{
    KFilts.reserve(3);
    KFilts.clear();

    //Initialize a Kalman filter for each free variable
    //0-East, 1-North, 2-Clock
    KFilts.push_back(
        Kalman(0.1,0.5,5.0,SVInit(1,1),SVRateInit(1,1),0.0,5.0,1.0,0.1)
    );
    KFilts.push_back(
        Kalman(0.1,0.5,5.0,SVInit(2,1),SVRateInit(2,1),0.0,5.0,1.0,0.1)
    );
    KFilts.push_back(
        Kalman(0.001,0.1,5.0,SVInit(3,1),SVRateInit(3,1),0.0,50.0,10.0,0.1)
    );

    AllPredSV = zeros(3,3);
    AllPredSVSigma = zeros(3,3);

    AllFiltSV = zeros(3,3);
    AllFiltSVSigma = zeros(3,3);
void Dynamics::mPrint(std::ostream & OutStream) const
{
    int i;
    int j;

    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            OutStream << setw(6) << fixed << setprecision(4) <<
            AllPredSV(i+1,j+1) << "",;
        }
    }
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            OutStream << setw(6) << fixed << setprecision(4) <<
            AllPredSVSigma(i+1,j+1) << "",;
        }
    }
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            OutStream << setw(6) << fixed << setprecision(4) <<
            AllFiltSV(i+1,j+1) << "",;
        }
    }
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            OutStream << setw(6) << fixed << setprecision(4) <<
            AllFiltSVSigma(i+1,j+1) << "",;
        }
    }
}

void Dynamics::mPredict(mwArray & PredSV, mwArray & PredSVSigma)
{
    mwArray NewSV;
    mwArray NewSVSigma;

    NewSV = zeros(3,1);
    NewSVSigma = zeros(3,1);

    for(int i=0;i<KFilts.size();i++)
    {

Appendix B: Source Code

```cpp
KFilts[i].mPredict(NewSV, NewSVSigma);

PredSV(i+1,1) = NewSV(1,1);
PredSVSigma(i+1,1) = NewSVSigma(1,1);

AllPredSV(colon(),i+1) = NewSV;
AllPredSVSigma(colon(),i+1) = NewSVSigma;
}
}

void Dynamics::mUpdate(const mwArray & SVIn, const double & AFVal, const int & NumUsed)
{
    double sigma;
    sigma = 500000.0;
    if(AFVal > 0.8)
        sigma = (300/TPI)*acos(AFVal);
    if(NumUsed < 15)
        sigma = 500000.0;

    mwArray SV;
    mwArray SVSigma;

    SV = zeros(3,1);
    SVSigma = zeros(3,1);

    for(int i=0;i<KFilts.size();i++)
    {
        KFilts[i].mFilter(SVIn(i+1,1), sigma, SV, SVSigma);
        AllFiltSV(colon(),i+1) = SV;
        AllFiltSVSigma(colon(),i+1) = SVSigma;
    }
}

#include <iostream>
#include <string>
#include <fstream>
#include <iomanip>

#define MSWIND  //Tells Matlab libraries we are using Windows
#define MSVC    //Tells Matlab libraries we are using Visual C++
#include "matlab.hpp"
```
```cpp
#include "../include/garmin.h"
#include "../include/constants.h"
#include "./tower/meas.h"

using namespace std;

Garmin::Garmin(const std::string fn) : pFile(new std::ofstream(fn.c_str(),
    std::ios::out|std::ios::trunc))
{
    (*pFile) << "H SOFTWARE NAME & VERSION" << endl;
    (*pFile) << "I PCX5 2.09" << endl;
    (*pFile) << endl;
    (*pFile) << "H R DATUM                IDX DA            DF
    DX            DY            DZ" << endl;
    (*pFile) << "M G WGS 84               121 +0.000000e+00
    +0.000000e+00 +0.000000e+00 +0.000000e+00 +0.000000e+00" << endl;
    (*pFile) << endl;
    (*pFile) << "H COORDINATE SYSTEM" << endl;
    (*pFile) << "U LAT LON DM" << endl;
    (*pFile) << endl;
    (*pFile) << "H IDNT   LATITUDE    LONGITUDE    DATE      TIME
    ALT   DESCRIPTION                              PROXIMITY
    SYMBOL ;waypts" << endl;
}

Garmin::mPrint(const mwArray & latlon, const MeasSetBase & msb)
{
    double ddegs;
    int degs;
    double mins;
    double Height = -9999;
    char hemi;
    char fillc;
    time_t TimeStamp;
    TimeStamp = msb.mGetTimeStampT();
    string TimeString(ctime(&TimeStamp),0,24);
    fillc = pFile->fill();
    pFile->fill('0');
    (*pFile) << "W ";
    degs = (int) msb.mGetElapsedTimeMin();
    mins = 100.0*(msb.mGetElapsedTimeMin() - (double) degs);
    (*pFile) << "A";
```
Appendix B: Source Code

(*pFile) << setw(2) << degs << "-";
(*pFile) << setw(2) << fixed << setprecision(0) << mins << " ";

//Determine latitude hemisphere and write to file
ddegs = latlon(1);
if(ddegs >= 0.0)
    hemi = 'N';
else
{
    hemi = 'S';
    ddegs = -ddegs;
}
(*pFile) << hemi;

//Convert decimal radians to whole degrees and decimal minutes
mGetDegMin(degs,mins,ddegs/Deg2Rad);

(*pFile) << setw(2) << degs;
(*pFile) << setw(10) << fixed << setprecision(7) << mins << " ";

//Determine longitude hemisphere and write to file
ddegs = latlon(2);
if(ddegs >= 0.0)
    hemi = 'E';
else
{
    hemi = 'W';
    ddegs = -ddegs;
}
(*pFile) << hemi;

//Convert decimal radians to whole degrees and decimal minutes
mGetDegMin(degs,mins,ddegs/Deg2Rad);

(*pFile) << setw(3) << degs;
(*pFile) << setw(10) << fixed << setprecision(7) << mins << " ";

(*pFile) << "12-OCT-01 00:00:00 ";
(*pFile) << (int) Height << " ";
pFile->fill(' ');
(*pFile) << setw(41) << left << TimeString << " ";
(*pFile) << right << "0.00000e+00 18";
(*pFile) << endl;

return EXIT_SUCCESS;
}
Garmin::mPrint(const double & latd, const double & lond,
const std::string & WName, const std::string & CommentString)
{
    double ddegs;
    int degs;
    double mins;
    double Height = -9999;
    char hemi;
    char fillc;

    (*pFile) << "W ";

    string WNameTrunc(WName);
    WNameTrunc.resize(6);

    (*pFile) << setw(7) << left << WNameTrunc << right;
    fillc = pFile->fill();
PFile->fill('0');

    //Determine latitude hemisphere and write to file
    ddegs = latd;
    if(ddegs >= 0.0)
    {
        hemi = 'N';
    }
    else
    {
        hemi = 'S';
        ddegs = -ddegs;
    }
    (*pFile) << hemi;

    //Convert decimal radians to whole degrees and decimal minutes
    mGetDegMin(degs,mins,ddegs/Deg2Rad);

    (*pFile) << setw(2) << degs;
    (*pFile) << setw(10) << fixed << setprecision(7) << mins << " ";

    //Determine longitude hemisphere and write to file
    ddegs = lond;
    if(ddegs >= 0.0)
    {
        hemi = 'E';
    }
    else
    {
        hemi = 'W';
        ddegs = -ddegs;
    }
    (*pFile) << hemi;

    //Convert decimal radians to whole degrees and decimal minutes
Appendix B: Source Code

```cpp
mGetDegMin(degs, mins, ddegs/Deg2Rad);

(*pFile) << setw(3) << degs;
(*pFile) << setw(10) << fixed << setprecision(7) << mins << " ";

(*pFile) << "12-OCT-01 00:00:00 ";
(*pFile) << (int) Height << " ";
pFile->fill(' ');
(*pFile) << setw(41) << left << CommentString << " ";
(*pFile) << right << "0.00000e+00       18";
(*pFile) << endl;
pFile->fill(fillc);

return EXIT_SUCCESS;
}

int Garmin::mGetDegMin(int & degs, double & mins, const double & ddegs)
{
    degs = (int) ddegs;
    mins = (ddegs - (double) degs)*60.0;

    return EXIT_SUCCESS;
}

#include <iostream>
#define MSWIND  //Tells Matlab libraries we are using Windows
#define MSVC    //Tells Matlab libraries we are using Visual C++
#include "matlab.hpp"
#include "kalman.h"

Kalman::Kalman(const double & beta, const double & sigma, const double & dt,
                const double & pos, const double & vel, const double & acc,
                const double & possig, const double & velsig, const double & accsig)
    : Beta(beta), Sigma(sigma), DeltaT(dt)
{
    SDA = zeros(3,3);
    SDQ = zeros(3,3);
```
const double b2 = Beta*Beta;
const double b3 = Beta*b2;
const double b4 = Beta*b3;
const double bt = Beta*DeltaT;
const double bt2 = bt*bt;
const double bt3 = bt2*bt;
const double embt = exp(-Beta*DeltaT);
const double embt2 = embt*embt;
const double var = Sigma*Sigma;

SDA(1,1) = 1.0;
SDA(1,2) = DeltaT;
SDA(1,3) = ((embt - 1.0)/b2)+(DeltaT/Beta);
SDA(2,1) = 0.0;
SDA(2,2) = 1.0;
SDA(2,3) = (1.0 - embt)/Beta;
SDA(3,1) = 0.0;
SDA(3,2) = 0.0;
SDA(3,3) = embt;

SDQ(1,1) = (2.0*bt+(2.0/3.0)*bt3-2.0*bt2-embt2-4.0*embt*bt+1.0)*(var/b4);
SDQ(1,2) = (-2.0*bt-2.0*embt+2.0*embt*bt+bt2+embt2+1.0)*(var/b3);
SDQ(1,3) = (-2.0*bt*embt-embt2+1.0)*(var/b2);
SDQ(2,1) = SDQ(1,2);
SDQ(2,2) = (-embt2+4.0*embt+2.0*bt-3.0)*(var/b2);
SDQ(2,3) = (-2.0*embt+embt2+1.0)*(var/Beta);
SDQ(3,1) = SDQ(1,3);
SDQ(3,2) = SDQ(2,3);
SDQ(3,3) = (-embt2+1.0)*var;

SV = zeros(3,1);
SV(1,1) = pos;
SV(2,1) = vel;
SV(3,1) = acc;

Obs = zeros(1,3);
Obs(1,1) = 1.0;

ObsT = zeros(3,1);
ObsT = ctranspose(Obs);

ECov = zeros(3,3);
ECov(1,1) = possig*possig;
ECov(2,2) = velsig*velsig;
ECov(3,3) = accsig*accsig;

KGain = zeros(3,1);

randn("state",sum(100.0*clock_func()));
Void Kalman::mFilter(const double & Meas, const double & Sigma, mwArray & sv, mwArray & sigout)
{
    double spsw;
    spsw = 1.0/( ECov(1,1) + Sigma*Sigma );
    KGain(1,1) = ECov(1,1)*spsw;
    KGain(2,1) = ECov(2,1)*spsw;
    KGain(3,1) = ECov(3,1)*spsw;
    SV = SV + KGain*(Meas - Obs*SV);
    ECov = ECov - KGain*Obs*ECov;
    sv = SV;
    sigout = sqrt(diag(ECov));
}

Void Kalman::mPredict(mwArray & sv, mwArray & sig)
{
    SV = SDA*SV;
    ECov = SDA*ECov*ctranspose(SDA) + SDQ;
    sv = SV;
    sig = sqrt(diag(ECov));
}

Void Kalman::Test(std::ostream& OutStream)
{
    SV = SDA*SV;
    SV(3,1) = SV(3,1) + Sigma*randn(1);
    OutStream << SV(1,1) << ",";
}

Kalman& Kalman::operator=(const Kalman & rhs)
{
    if (this == &rhs) return *this;
    std::cerr << "Error: Kalman::operator=()!" << std::endl;
    std::cerr << "Operator= should not be used for this class!" << std::endl;
    return *this;
}

// matlabeng.cpp
*/
#include <vector>
#include <iostream>
#define MSWIND //Tells Matlab libraries we are using Windows
#define MSVC    //Tells Matlab libraries we are using Visual C++
#include "engine.h"
#include "../include/matlabeng.h"

using namespace std;

MatlabEng::MatlabEng()
{
    std::cout << "Opening Matlab engine..." << endl;
    pEng = engOpen("\0");
    std::cout << "Matlab engine opened successfully!" << endl;
}

MatlabEng::~MatlabEng()
{
    std::cout << "Press return to close Matlab Engine." << endl;
    fgetc(stdin);
    engClose(pEng);
}

int MatlabEng::mPlot(const std::vector<double>& x, const std::vector<double>& y, const int num)
{
    mxArray *pX;
    mxArray *pY;
    double *pXr;
    double *pYr;
    int i;
    int imax = x.size();
    pX = mxCreateDoubleMatrix(1, imax, mxREAL);
    pY = mxCreateDoubleMatrix(1, imax, mxREAL);
    pXr = mxGetPr(pX);
    pYr = mxGetPr(pY);
    switch(num)
    {
    case 1 :
        mxSetName(pX, "x1");
        mxSetName(pY, "y1");
        break;
    case 2 :
        
    
}
Appendix B: Source Code

```
mxSetName(pX, "x2");
mxSetName(pY, "y2");
break;
case 3 :
    mxSetName(pX, "x3");
    mxSetName(pY, "y3");
break;
default :
    mxSetName(pX, "x0");
    mxSetName(pY, "y0");
}

for(i=0;i<imax;i++)
{
    *(pXr+i)=x[i];
    *(pYr+i)=y[i];
}

engPutArray(pEng,pX);
engPutArray(pEng,pY);

switch(num)
{
    case 1 :
        engEvalString(pEng, "plot(x1,y1,'mo');");
        break;
    case 2 :
        engEvalString(pEng, "plot(x2,y2,'g*');");
        break;
    case 3 :
        engEvalString(pEng, "plot(x3,y3,'r+');");
        break;
    default :
        engEvalString(pEng, "plot(x0,y0,'bd');");
}

mxDestroyArray(pX);
mxDestroyArray(pY);

return EXIT_SUCCESS;
}

int MatlabEng::mPut(const double * const pData, const int num)
{
    mxArray *pY;
    double *pYr;

    int i;

    pY = mxCreateDoubleMatrix(1,num,mxREAL);
pYr = mxGetPr(pY);
```
B.2 Navigation Algorithm Program

mxSetName(pY, "y1");

for(i=0;i<num;i++)
{
    *(pYr+i)=*(pData+i);
}

engPutArray(pEng,pY);

mxDestroyArray(pY);

return EXIT_SUCCESS;

}  

int MatlabEng::mPut(const double * const pXin, const double *
const pYin, const double * const pTin,
const double * const pZin,
const int m, const int n, const int p)
{
    mxArray *pX;
    mxArray *pY;
    mxArray *pT;
    mxArray *pZ;

double *pXr;
    double *pYr;
    double *pTr;
    double *pZr;

int *pDims;

pDims = new int[3];

(*pDims) = m; (*pDims+1) = 1; (*pDims+2) = 0;
pX = mxCreateNumericArray(2,pDims,mxDOUBLE_CLASS,mxREAL);
pXr = mxGetPr(pX);
mxSetName(pX, "x");

(*pDims) = n; (*pDims+1) = 1; (*pDims+2) = 0;
pY = mxCreateNumericArray(2,pDims,mxDOUBLE_CLASS,mxREAL);
pYr = mxGetPr(pY);
mxSetName(pY, "y");

(*pDims) = p; (*pDims+1) = 1; (*pDims+2) = 0;
pT = mxCreateNumericArray(2,pDims,mxDOUBLE_CLASS,mxREAL);
pTr = mxGetPr(pT);
mxSetName(pT, "t");

(*pDims) = m; (*pDims+1) = n; (*pDims+2) = p;
pZ = mxCreateNumericArray(3,pDims,mxDOUBLE_CLASS,mxREAL);
pZr = mxGetPr(pZ);
Appendix B: Source Code

```c
mxSetName(pZ, "z");
dele pDims;

for (int i=0; i<m; i++)
{
    (*(pXr+i)) = (*(pXin+i));
}

for (i=0; i<n; i++)
{
    (*(pYr+i)) = (*(pYin+i));
}

for (i=0; i<p; i++)
{
    (*(pTr+i)) = (*(pTin+i));
}

int q = m*n*p;
for (i=0; i<q; i++)
{
    (*(pZr+i)) = (*(pZin+i));
}

engPutArray(pEng,pX);
engPutArray(pEng,pY);
engPutArray(pEng,pT);
engPutArray(pEng,pZ);

mxDestroyArray(pX);
mxDestroyArray(pY);
mxDestroyArray(pT);
mxDestroyArray(pZ);

return EXIT_SUCCESS;
}
```

// meas.cpp

// AM signal measurements source file

#include <vector>
#include <functional>
#include <numeric>
#include <algorithm>
#include <iostream>

```c
```

266
B.2 Navigation Algorithm Program
#include
#include
#include
#include
#include

<sstream>
<iomanip>
<cmath>
<time.h>
<limits>

#include "../include/constants.h"
#include "meas.h"
#include "station.h"
using namespace std;
MeasFile::MeasFile(const std::string fn) : pFile(new
std::ifstream(fn.c_str()))
{
if( !(*pFile))
{
std::cerr << "Can't open file " << fn << "." << std::endl;
exit(EXIT_FAILURE);
}
std::cout << "File " << fn << " opened successfully!" <<
std::endl;
HavePos = false;
Lat = 0.0;
Lon = 0.0;
if( ( pFile->peek() ) == 'P' )
{
HavePos = true;
mCheckLineType('P');
*pFile >> Lat;
*pFile >> Lon;
Lat *= Deg2Rad;
Lon *= Deg2Rad;
}
double
double
double
double
double

c1;
c2;
StartTime;
EpochNum;
TriggerPeriod;

mCheckLineType('T');
*pFile >> StartTime;
*pFile >> EpochNum;
*pFile >> TriggerPeriod;
PulseTimeStamp = ((double) StartTime) + (TriggerPeriod*((double)
EpochNum));

267


mCheckLineType('S');
*pFile >> c1;
*pFile >> c2;

PulseTime = ( (c1+c2)/2.0 )*C;
}

bool MeasFile::mGetPos(double & lat, double & lon)
{
    if( HavePos )
    {
        lat = Lat;
        lon = Lon;
    }
    else
    {
        std::cout << "Position not found in file." << endl;
        exit(EXIT_FAILURE);
    }
    return true;
}

bool MeasFile::mReadEpoch(MeasSetSS & mss)
{
    //Look for next T line in file - bail if not first character
    if( mCheckLineType('T') )
    {
        mss.mLoadData(pFile,*this);
        return true;
    }
    return false;
}

bool MeasFile::mCheckLineType(char lt) const
{
    char LineType;

    *pFile >> LineType;
    if(LineType != lt)
    {
        if( pFile->eof() )
            return false;
        else
        {
            std::cerr << lt << " line not found." << std::endl;
            exit(EXIT_FAILURE);
        }
    }
    return true;
}
int MeasSetBase::mPrintDateTime(ostream & OutStream) const
{
    time_t timestamp;

    timestamp = (time_t) (TimeStamp+0.5);

    //This is a hack to compensate for a bug in the phase program
    // timestamp += 5;

    std::string timestring(ctime( &timestamp ));
    std::istringstream timestream(timestring);

    std::string dayofweek;
    std::string month;
    std::string day;
    std::string time;
    std::string year;

    timestream >> dayofweek;
    timestream >> month;
    timestream >> day;
    timestream >> time;
    timestream >> year;

    char sep=',';
    if(OutStream == std::cout)
        sep = ' ';

    OutStream << day << " " << month << " " << year << sep << time
            << sep;

    //Also print elapsed time in minutes and epoch number
    OutStream << EpochNum << sep;
    double etime;
    etime = ( (double) (timestamp - StartTime) )/60.0;
    OutStream << setw(7) << fixed << setprecision(4) << etime << sep;

    return EXIT_SUCCESS;
}

int MeasSetSS::mPrint(ostream & OutStream) const
{
    for(int i=0;i<StationData.size();i++)
    {
        StationData[i].mPrint(OutStream);
    }
    return EXIT_SUCCESS;
}

int MeasSetSS::mPrintIds(ostream & OutStream) const
Appendix B: Source Code

```cpp
int MeasSetSS::mPrintFreqs(std::ostream & OutStream) const
{
    for(int i=0;i<StationData.size();i++)
    {
        if(StationData[i].mGetUse() && StationData[i].mGetReport())
            StationData[i].mPrintFreq(OutStream);
    }
    OutStream << std::endl;
    return EXIT_SUCCESS;
}

int MeasSetSS::mPrintMags(std::ostream & OutStream) const
{
    for(int i=0;i<StationData.size();i++)
    {
        if(StationData[i].mGetUse() && StationData[i].mGetReport())
            StationData[i].mPrintMag(OutStream);
    }
    OutStream << std::endl;
    return EXIT_SUCCESS;
}

MeasSetSS::MeasSetSS()
{
    StationData.reserve(StationNum);
    StationData.clear();

    int StationId = StationIdMin;
    double nomfreq;

    for(int i=0;i<StationNum;i++)
    {
        nomfreq = StationId * StationId2NomFreq;
        StationData.push_back( MeasElemSS(StationId,nomfreq) );
        StationId += StationIdSpacing;
    }

    HarmData.reserve(HarmonicNum);
    HarmData.clear();
}
```
for(i=0;i<HarmonicNum;i++)
{
    nomfreq = i * HarmonicId2NomFreq;
    HarmData.push_back( MeasElemSS(i+1,nomfreq) );
}

int MeasSetSS::mLoadData(std::ifstream * pFile, const MeasFile & mf)
{
    double TriggerPeriod;

    *pFile >> StartTime;
    *pFile >> EpochNum;
    *pFile >> TriggerPeriod;

    TimeStamp = ((double) StartTime) + (TriggerPeriod*((double) EpochNum));

    int Num;
    double id2nom;

    //Check for Q line in data file
    mf.mCheckLineType('Q');
    *pFile >> Num;
    *pFile >> id2nom;

    if( Num != StationData.size() )
    {
        std::cerr << "Error: MeasSetSS::mLoadData!" << std::endl;
        std::cerr << "Number of stations in data file is different from number in MeasSetSS." << std::endl;
        exit(EXIT_FAILURE);
    }
    for(int i=0;i<Num;i++)
    {
        mf.mCheckLineType('R');
        StationData[i].mLoadData(pFile);
    }

    //Check for B line in data file
    mf.mCheckLineType('B');
    *pFile >> Num;
    *pFile >> id2nom;

    if( Num != HarmData.size() )
    {
        std::cerr << "Error: MeasSetSS::mLoadData!" << std::endl;
        std::cerr << "Number of harmonics in data file is different from number in MeasSetSS." << std::endl;
        exit(EXIT_FAILURE);
    }
}
for(i=0;i<Num;i++)
{
    mf.mCheckLineType('C');
    HarmData[i].mLoadData(pFile);
}

return EXIT_SUCCESS;
}

int MeasSetSS::mSetUse(const std::vector<int> & ids)
{
    int j=0;
    for(int i=0;i<StationData.size();i++)
    {
        if( StationData[i].mGetId() == ids[j] )
        {
            StationData[i].mUse();
            j++;
        }
    }
    return EXIT_SUCCESS;
}

int MeasSetSS::mDontUse(const std::vector<int> & ids)
{
    int i=0;
    int j=0;
    for(j=0;j<ids.size();j++)
    {
        for(i=0;i<StationData.size();i++)
        {
            if( StationData[i].mGetId() == ids[j] )
            {
                StationData[i].mDontUse();
                break;
            }
        }
    }
    return EXIT_SUCCESS;
}

void MeasSetSS::mNeverReport(const std::vector<int> & ids)
{
    int i=0;
    int j=0;
    for(j=0;j<ids.size();j++)
    {
        for(i=0;i<StationData.size();i++)
        {
B.2 Navigation Algorithm Program

```c++
if( StationData[i].mGetId() == ids[j] )
{
    StationData[i].mNeverReport();
    break;
}
}
}

MeasSetFD::MeasSetFD() : EpochPeriod(0.0), EstFreqRatio(0.0),
ClockOffsetRate(0.0)
{
    Data.reserve(StationNum);
    Data.clear();

    int StationId = StationIdMin;
    double nomfreq;

    for(int i=0;i<StationNum;i++)
    {
        nomfreq = StationId * StationId2NomFreq;
        Data.push_back( MeasElemFD(StationId,nomfreq) );
        StationId += StationIdSpacing;
    }
}

int MeasSetFD::mLoadData(const MeasSetSS & lhs, const MeasSetSS & rhs)
{
    //Calculate elapsed time since last time mLoadData was last
    //called
    //This is for advancing the clock offset based on the freq ratio
    EpochPeriod = rhs.mGetTimeStamp() - TimeStamp;

    EpochNum = rhs.mGetEpochNum();
    TimeStamp = rhs.mGetTimeStamp();
    StartTime = rhs.mGetStartTime();

    for(int i=0;i<Data.size();i++)
    {
        Data[i].mLoadData( lhs.mGetMeasSta(i),rhs.mGetMeasSta(i) );
    }

    return EXIT_SUCCESS;
}

void MeasSetFD::mDontUse(const std::vector<int> & ids)
{
    int i=0;
    int j=0;
```
for(i=0;i<Data.size();i++)
{
    if(j >= ids.size() )
        break;
    if( Data[i].mGetId() == ids[j] )
        { Data[i].mDontUse();
            j++;
        }
}

void MeasSetFD::mDontUseWeak(const double & ThreshDB)
{
    for(int i=0;i<Data.size();i++)
    {
        if( Data[i].mGetMagMin() < ThreshDB)
            { Data[i].mDontUse();
            }
    }
}

// This function does not account for Doppler shift
int MeasSetFD::mEstimateFreqRatio()
{
    std::vector<double> fr;
    fr.reserve( Data.size() );
    fr.clear();

    int istop = Data.size();
    for(int i=0;i<istop;i++)
    {
        if( Data[i].mGetUse() )
            { fr.push_back( Data[i].mGetFreqRatio() - 1.0 );
            }
    }
    sort(fr.begin(),fr.end());

    //Set estimated frequency ratio equal to the median freq ratio
double NewEstFreqRatio = fr[ ( fr.size() )/2 ];
    if( fabs(EstFreqRatio) > std::numeric_limits<double>::epsilon() )
    
        ClockOffsetRate = ( ( NewEstFreqRatio + EstFreqRatio )/2.0 )*C;
    else
        ClockOffsetRate = NewEstFreqRatio*C;

    EstFreqRatio = NewEstFreqRatio;
return EXIT_SUCCESS;
}

int MeasSetFD::mInvalidateFR(const double & MaxFreqDevPpm)
{
    int istop = Data.size();
    for(int i=0;i<istop;i++)
    {
        if( Data[i].mGetUse() )
        {
            if( fabs(Data[i].mGetFreqRatio() - 1.0 - EstFreqRatio) >
                MaxFreqDevPpm/(Data[i].mGetFreqBase()))
                Data[i].mDontUse();
        }
    }
    return EXIT_SUCCESS;
}

const MeasElemFD & MeasSetFD::mGetStaRef(const int & id) const
{
    for(int i=0;i<Data.size();i++)
    {
        if( id == Data[i].mGetId() )
            return Data[i];
    }
    std::cerr << "Error MeasSetFD::mGetStaRef: Measurement with ID 
"<< id << " not found!" << std::endl;
    exit(EXIT_FAILURE);
}

const MeasElemFD * MeasSetFD::mGetStaPtr(const int & id) const
{
    for(int i=0;i<Data.size();i++)
    {
        if( id == Data[i].mGetId() )
            return &(Data[i]);
    }
    std::cerr << "Error MeasSetFD::mGetStaRef: Measurement with ID 
"<< id << " not found!" << std::endl;
    exit(EXIT_FAILURE);
}

int MeasSetFD::mPrint(std::ostream & OutStream) const
{
    for(int i=0;i<Data.size();i++)
    {
        Data[i].mPrint(OutStream);
    }
Appendix B: Source Code

```cpp
Appendix B: Source Code

MeasElemBase& MeasElemBase::operator=(const MeasElemBase & rhs)
{
    if (this == &rhs) return *this;

    if(Id != rhs.Id)
    {
        std::cerr << "Error: MeasElemBase::operator=()!" << std::endl;
        std::cerr << "Operator = attempted on objects with different Id's." << std::endl;
        exit(EXIT_FAILURE);
    }
    Use = rhs.Use;

    return *this;
}

int MeasElemBase::mPrint(std::ostream & OutStream) const
{
    char sep=',';
    if(OutStream == std::cout)
        sep = ' ';

    OutStream << setw(5) << Id << sep;
    OutStream << setw(3) << Use << sep;
    return EXIT_SUCCESS;
}

int MeasElemSS::mPrint(std::ostream & OutStream) const
{
    char sep=',';
    if(OutStream == std::cout)
        sep = ' ';

    MeasElemBase::mPrint(OutStream);
    OutStream << setw(15) << setprecision(10) << Freq/MHz2RPS << sep;
    OutStream << setw(15) << setprecision(10) << Phase/Deg2Rad << sep;
    OutStream << setw(15) << setprecision(10) << Mag << std::endl;
    return EXIT_SUCCESS;
}

MeasElemSS& MeasElemSS::operator=(const MeasElemSS & rhs)
{
    if (this == &rhs) return *this;

    return *this;
}
```

276
MeasElemBase::operator=(rhs);
Freq = rhs.Freq;
Mag = rhs.Mag;
Phase = rhs.Phase;
return *this;
}

MeasElemFD& MeasElemFD::operator=(const MeasElemFD & rhs)
{
    if (this == &rhs) return *this;

    MeasElemBase::operator=(rhs);

    FreqBase = rhs.FreqBase;
    MagRoam = rhs.MagRoam;
    PhaseFD = rhs.PhaseFD;
    FreqRatio = rhs.FreqRatio;

    return *this;
}

int MeasElemFD::mLoadData(const MeasElemSS & lhs, const MeasElemSS & rhs)
{
    Use = ( (lhs.mGetUse()) && (rhs.mGetUse()) );
    FreqBase = rhs.mGetFreq();
    //FreqRatio is >1 if rhs has a faster clock than the lhs 
    //FreqRatio is initially set to observed frequency in the
    constructor
    FreqRatio = (lhs.mGetFreq()) / FreqBase;
    PhaseFD = lhs.mGetPhase() - rhs.mGetPhase();
    MagRoam = lhs.mGetMag();
    if( MagRoam < rhs.mGetMag() )
        MagMin = MagRoam;
    else
        MagMin = rhs.mGetMag();
    return EXIT_SUCCESS;
}

int MeasElemFD::mPrint(std::ostream & OutStream) const
{
    char sep = ',';
    if (OutStream == std::cout)
        sep = ' ';

    MeasElemBase::mPrint(OutStream);
    OutStream << setw(15) << setprecision(10) << FreqBase/MHz2RPS << sep;
    OutStream << setw(20) << setprecision(10) << FreqRatio - 1.0 << sep;
Appendix B: Source Code

```cpp
OutStream << setw(15) << setprecision(10) << PhaseFD/Deg2Rad << sep;
OutStream << setw(15) << setprecision(10) << MagRoam << std::endl;
return EXIT_SUCCESS;
}

int MeasElemSS::mLoadData(std::ifstream * pFile) {
    int idtest;
    *pFile >> idtest;
    if(idtest != Id) {
        std::cerr << "Error: MeasElemSS::mLoadData!" << std::endl;
        std::cerr << "Id in file " << idtest << " does not match " << std::endl;
        std::cerr << "id in class " << Id << ":." << std::endl;
        exit(EXIT_FAILURE);
    }
    *pFile >> Freq;
    *pFile >> Phase;
    *pFile >> Mag;

    // Data files have frequency in MHz not in radians/sec
    Freq *= MHz2RPS;

    return EXIT_SUCCESS;
}

/////////////////////////////////////////////////
// param.cpp
/////////////////////////////////////////////////

#include <string>
#include <fstream>
#include <iostream>
#include "param.h"

Param::Param(const std::string fn) : pFile(new std::ifstream(fn.c_str())),
    NumToSkip(12), MaxNumEpoch(0), UseMatlab(false), MaxFreqDevPPM(1.0),
    
```
ElimBadMeas(0), MinAF(0.97), DoSearch(true), SearchRangeMeters(100.0),
   ClockSearchRangeMeters(100.0), StepSizePercent(0.4)
{
   if( !(*pFile))
   {
      std::cerr << "Can't open file " << fn << "." << std::endl;
      exit(EXIT_FAILURE);
   }

   StationsNotUsed.reserve(10);
   StationsNotUsed.clear();

   int numstations;
   int numstationsnr;
   int stationid;
   int i;

   std::cout << "Parameter file " << fn << " opened successfully!"
   << std::endl;

   std::string VarInName;

   while( !( *pFile ).eof() )
   {
      *pFile >> VarInName;

      if( VarInName == "NumToSkip")
      {
         *pFile >> NumToSkip;
      }
      if( VarInName == "MaxNumEpoch")
      {
         *pFile >> MaxNumEpoch;
      }
      if( VarInName == "UseMatlab")
      {
         *pFile >> UseMatlab;
      }
      if( VarInName == "MaxFreqDevPPM")
      {
         *pFile >> MaxFreqDevPPM;
      }
      if( VarInName == "UseOddManOut")
      {
         *pFile >> ElimBadMeas;
      }
      if( VarInName == "MinAF")
      {
         *pFile >> MinAF;
      }
if(VarInName == "DoSearch")
{
    *pFile >> DoSearch;
}
if(VarInName == "SearchRangeMeters")
{
    *pFile >> SearchRangeMeters;
}
if(VarInName == "ClockSearchRangeMeters")
{
    *pFile >> ClockSearchRangeMeters;
}
if(VarInName == "StepSizePercent")
{
    *pFile >> StepSizePercent;
}
if(VarInName == "NumStationsDontUse")
{
    *pFile >> numstations;
}
if(VarInName == "StationsDontUse")
{
    for(i=0; i<numstations; i++)
    {
        *pFile >> stationid;
        StationsNotUsed.push_back(stationid);
    }
}
if(VarInName == "NumStationsDontReport")
{
    *pFile >> numstationsnr;
}
if(VarInName == "StationsDontReport")
{
    for(i=0; i<numstationsnr; i++)
    {
        *pFile >> stationid;
        StationsNotReported.push_back(stationid);
    }
}
B.2 Navigation Algorithm Program

```cpp
#include <vector>
#define MSWIND  //Tells Matlab libraries we are using Windows
#define MSVC    //Tells Matlab libraries we are using Visual C++
#include "matlab.hpp"
#include "rateest.h"
#include "station.h"
#include "meas.h"
#include "../include/constants.h"

RateEst::RateEst(const StationSet & ss, const MeasSetFD & msf)
{
    std::vector<int> ids;
    int size;
    ss.mGetIds(ids);
    size = ids.size();
    Meas.clear();
    Meas.reserve(size);
    Stas.clear();
    Stas.reserve(size);
    Obs = ones(size, 3);
    MeasVec = zeros(size, 1);
    Weight = zeros(size, size);
    Residuals = zeros(size, 1);
    for(int i=0;i<size;i++)
    {
        Meas.push_back( msf.mGetStaPtr( ids[i] ) );
        Stas.push_back( ss.mGetStaPtr( ids[i] ) );
    }
}

void RateEst::mEstimateRates(mwArray & svin, mwArray & svdot)
{
    double East;
    double North;
    double EastUnit;
    double NorthUnit;
    double WeightI;
    East = svin(1,1);
    North = svin(2,1);
    for(int i=0;i<Stas.size();i++)
```
Appendix B: Source Code

```c++
{
    Stas[i]->mCalcUnit(East,North,EastUnit,NorthUnit);
    Obs(i+1,1) = EastUnit/C;
    Obs(i+1,2) = NorthUnit/C;
    Obs(i+1,3) = 1.0;

    MeasVec(i+1) = Meas[i]->mGetFreqRatio();
    WeightI = Meas[i]->mGetMagMin() - 50.0;

    if(WeightI < 0.0)
    {
        WeightI = 0.0;
    }

    Weight(i+1,i+1) = WeightI;
}

svdot = pinv(Weight*Obs)*Weight*MeasVec;

Residuals = Obs*svdot - MeasVec;

svdot(1) = svdot(1)/svdot(3);
svdot(2) = svdot(2)/svdot(3);
svdot(3) = (svdot(3) - 1.0)*C;
}

void RateEst::mGetInvalidIds(std::vector<int> & ids, const double threshppm) const
{
    ids.clear();
    ids.reserve( Meas.size() );

    for(int i=0;i<Meas.size();i++)
    {
        if( fabs(Residuals(i+1)) > threshppm/( Meas[i]->mGetFreqBase() ) )
            ids.push_back( Meas[i]->mGetId() );
    }
}
```

```cpp
#include <string>
#include <iostream>
#include <fstream>
```
#include <iomanip>
#include <vector>
#include <limits>

#include "../include/constants.h"
#include "../include/coordsys.h"
#include "station.h"

using namespace std;

TowerEx::TowerEx(std::ifstream & File)
{
    File >> Id;
    File.ignore();
    File >> TowerId;
    File.ignore();
    File >> ExId;
    File.ignore();
    File >> Field;
    File.ignore();
    File >> Phase;
    File.ignore();
    File >> TowerHeight;
    File.ignore();

    Phase *= Deg2Rad;

    // Scale field by tower height if tower height is in database
    if(TowerHeight > 1.0)
        Field *= TowerHeight;
}

int TowerEx::mPrint(std::ostream& OutStream) const
{
    char sep=',';

    if(OutStream == std::cout)
        sep = ' ';

    OutStream << setw(5) << Id << sep;
    OutStream << setw(3) << TowerId << sep;
    OutStream << setw(3) << ExId << sep;
    OutStream << setw(10) << setprecision(5) << Field << sep;
    OutStream << setw(10) << setprecision(5) << Phase << sep;
    OutStream << setw(10) << setprecision(5) << TowerHeight <<
    std::endl;

    return EXIT_SUCCESS;
}
AntEx::AntEx(std::ifstream & File, const int nt) : NumTowers(nt)
{
    Data.reserve(NumTowers);
    Data.clear();

    for(int i=0;i<NumTowers;i++)
    {
        Data.push_back( TowerEx(File) );
    }
}

const double & AntEx::mGetField(const int & id) const { return Data[id].mGetField();}

const double & AntEx::mGetPhase(const int & id) const { return Data[id].mGetPhase();}

int AntEx::mPrint(std::ostream& OutStream) const
{
    for(int i=0;i<Data.size();i++)
    {
        Data[i].mPrint(OutStream);
    }
    return EXIT_SUCCESS;
}

TowerPos::TowerPos(std::ifstream & File) : East(0.0), North(0.0), Up(0.0)
{
    File >> Id;
    File.ignore();
    File >> TowerId;
    File.ignore();
    File >> Lat;  
    File.ignore();
    File >> Lon; 
    File.ignore();
    File >> Height;
    File.ignore();

    Lat *= Deg2Rad; 
    Lon *= Deg2Rad; 
}

double TowerPos::mCalcDist(const double & ea, const double & no, 
                             double & deltaeast, double & deltanorth) const
{
    deltaeast = (ea - East);
    deltanorth = (no - North);

    return sqrt(deltaeast*deltaeast + deltanorth*deltanorth);
B.2 Navigation Algorithm Program

```cpp
int TowerPos::mSetENU(CoordSys & lcs)
{
    lcs.mGetENU(East, North, Up, Lat, Lon, Height);
    return EXIT_SUCCESS;
}

int TowerPos::mSetGeo(CoordSys & lcs)
{
    lcs.mGetGeo(Lat, Lon, Height, East, North, Up);
    return EXIT_SUCCESS;
}

int TowerPos::mPrint(std::ostream& OutStream) const
{
    char sep=';';

    if(OutStream == std::cout)
        sep = ' ;';

    OutStream << setw(5) << Id << sep;
    OutStream << setw(3) << TowerId << sep;
    OutStream << setw(13) << setprecision(10) << Lat/Deg2Rad << sep;
    OutStream << setw(13) << setprecision(10) << Lon/Deg2Rad << sep;
    OutStream << setw(8) << setprecision(5) << Height << sep;
    OutStream << setw(10) << setprecision(7) << East << sep;
    OutStream << setw(10) << setprecision(7) << North << sep;
    OutStream << setw(8) << setprecision(5) << Up;
    OutStream << std::endl;

    return EXIT_SUCCESS;
}

Station::Station(std::ifstream & File) : CurrentEx(0), CenterEast(0.0), CenterNorth(0.0)
{
    File >> Id;
    File.ignore();
    File >> NumTowers;
    File.ignore();
    File >> NumEx;
    File.ignore();

    for(int i=0;i<3;i++)
    {
        File >> Tod2Ex[i];
        File.ignore();
    }
}
```
if( Tod2Ex[i] > NumEx || Tod2Ex[i] < 0)
{
    std::cerr << "Error Station::Station: Invalid exitation number" << std::endl;
    exit(EXIT_FAILURE);
}

// We really want exitation numbers to start at 0 not 1
// Ex of -1 will signify no operation
Tod2Ex[i]--;

CurrentEx = Tod2Ex[0];

if (CurrentEx < 0)
    Use = false;
else
    Use = true;

for(i=0;i<3;i++)
{
    File >> Power[i];
    File.ignore();
}

ExData.reserve(NumEx);
TPData.reserve(NumTowers);
ExData.clear();
TPData.clear();

File >> GndFuncB;
File.ignore();
File >> GndFuncM;
File.ignore();

void Station::mCalcCenter()
{
    int count = 0;

    CenterEast = 0.0;
    CenterNorth = 0.0;

    for(int i=0;i<NumTowers;i++)
    {
        if( ExData[CurrentEx].mGetField(i) >
            std::numeric_limits<double>::epsilon() )
        {
            CenterEast += TPData[i].mGetEast();
            CenterNorth += TPData[i].mGetNorth();
            count++;
        }
CenterEast /= ( (double) count );
CenterNorth /= ( (double) count );

void Station::mCalcUnit(const double & east, const double & north,
                          double & eastunit, double & northunit) const
{
    double dist;

    eastunit = CenterEast - east;
    northunit = CenterNorth - north;

    dist = sqrt( eastunit*eastunit + northunit*northunit );

    eastunit /= dist;
    northunit /= dist;
}

int Station::mLoadTPData(std::ifstream & File)
{
    for(int i=0;i<NumTowers;i++)
    {
        TPData.push_back( TowerPos(File) );
    }
    return EXIT_SUCCESS;
}

int Station::mLoadExData(std::ifstream & File)
{
    for(int i=0;i<NumEx;i++)
    {
        ExData.push_back( AntEx(File,NumTowers) );
    }
    return EXIT_SUCCESS;
}

int Station::mSetENU(CoordSys & LCS)
{
    for(int i=0;i<NumTowers;i++)
    {
        TPData[i].mSetENU(LCS);
    }
    return EXIT_SUCCESS;
}

double Station::mCalcThPhase(const double & east, const double & north,
                             const double & Wavenumber) const
{ double ss = 0.0; double sc = 0.0; 

double field = 0.0; double scarg = 0.0; double dist = 0.0; 

for(int i=0;i<NumTowers;i++) 
{ dist = TPData[i].mCalcDist(east,north); field = ( ExData[CurrentEx].mGetField(i) )/dist; scarg = ExData[CurrentEx].mGetPhase(i) - Wavenumber*dist; scarg += Deg2Rad*GndFuncB*exp(-GndFuncM*dist); ss += field*sin(scarg); sc += field*cos(scarg); 
} 

double phase; phase = atan2(ss,sc); return phase; 
}

double Station::mCalcThPhase(const double & east, const double & north, const double & Wavenumber, double & dwrteast, double & dwrtnorth) const 
{ double ss = 0.0; double sc = 0.0; double sspe = 0.0; double scpe = 0.0; double sspn = 0.0; double scpnt = 0.0; double sscomp = 0.0; double sccomp = 0.0; double field = 0.0; double scarg = 0.0; double distrads = 0.0; double deltaeast = 0.0; double deltanorth = 0.0; 

for(int i=0;i<NumTowers;i++) 
{ distrads = Wavenumber*TPData[i].mCalcDist(east,north,deltaeast,deltanorth); field = ( ExData[CurrentEx].mGetField(i) )/ distrads; scarg = ExData[CurrentEx].mGetPhase(i) - distrads; scarg += Deg2Rad*GndFuncB*exp(-GndFuncM*distrads/Wavenumber); sscomp = field*sin(scarg); 
}
sccomp = field*cos(scarg);
ss += sccomp;
sc += sccomp;
distrads /= Wavenumber*Wavenumber;
sspe -= sccomp*deltaeast/distrads;
scpe += sccomp*deltaeast/distrads;
sspn -= sccomp*deltanorth/distrads;
scpnn += sccomp*deltanorth/distrads;
}

double sssc = 0.0;

sssc = ss*ss + sc*sc;
dwrteast = (sc*sspe - ss*scpe)/sssc;
dwrtnorth = (sc*sspn - ss*scpnn)/sssc;

double phase;
phase = atan2(ss,sc);

return phase;

int Station::mPrint(std::ostream& OutStream) const
{
    char sep=',*;

    if(OutStream == std::cout)
        sep = ' ';

    OutStream << setw(5) << Id << sep;
    OutStream << setw(3) << NumTowers << sep;
    OutStream << setw(3) << NumEx << sep;

    for(int i=0;i<3;i++)
    {
        OutStream << setw(3) << Tod2Ex[i] << sep;
    }

    for(i=0;i<3;i++)
    {
        OutStream << setw(9) << setprecision(4) << Power[i] << sep;
    }

    OutStream << std::endl;

    for(i=0;i<NumTowers;i++)
    {
        TPData[i].mPrint(OutStream);
    }
}
for(i=0;i<NumEx;i++)
{
    ExData[i].mPrint(OutStream);
}

return EXIT_SUCCESS;

StationSet::StationSet(const std::string sfn,const std::string
tfn,const std::string efn)
{
    Data.reserve(MaxStations);
    Data.clear();

    std::ifstream StationFile(sfn.c_str());
    std::ifstream TowerFile(tfn.c_str());
    std::ifstream ExFile(efn.c_str());

    if(!StationFile)
    {
        std::cerr << "Can't open file " << sfn << "." << std::endl;
        exit(EXIT_FAILURE);
    }
    else
        std::cout << "File " << sfn << " opened successfully!" << std::endl;

    if(!TowerFile)
    {
        std::cerr << "Can't open file " << tfn << "." << std::endl;
        exit(EXIT_FAILURE);
    }
    else
        std::cout << "File " << tfn << " opened successfully!" << std::endl;

    if(!ExFile)
    {
        std::cerr << "Can't open file " << efn << "." << std::endl;
        exit(EXIT_FAILURE);
    }
    else
        std::cout << "File " << efn << " opened successfully!" << std::endl;

    int i = 0;
    while( !StationFile.eof() ) 
    {
        Data.push_back( Station(StationFile) );

        Data[i].mLoadTPData(TowerFile);
        Data[i].mLoadExData(ExFile);
    }
if( Data[i].mGetUse() )
    i++;
else
    Data.pop_back();
}

int StationSet::mGetIds(std::vector<int> & ids) const
{
    ids.clear();
    ids.reserve( Data.size() );
    for(int i=0;i<Data.size();i++)
    {
        ids.push_back( Data[i].mGetId() );
    }
    return EXIT_SUCCESS;
}

int StationSet::mSetENU(CoordSys & LCS)
{
    for(int i=0;i<Data.size();i++)
    {
        Data[i].mSetENU(LCS);
    }
    return EXIT_SUCCESS;
}

const Station & StationSet::mGetStaRef(const int & id) const
{
    for(int i=0;i<Data.size();i++)
    {
        if( id == Data[i].mGetId() )
            return Data[i];
    }
    std::cerr << "Error StationSet::mGetStaRef: Station with ID " << id << " not found!" << std::endl;
    exit(EXIT_FAILURE);
}

const Station * StationSet::mGetStaPtr(const int & id) const
{
    for(int i=0;i<Data.size();i++)
    {
        if( id == Data[i].mGetId() )
            return &( Data[i] );
    }
    std::cerr << "Error StationSet::mGetStaPtr: Station with ID " << id << " not found!" << std::endl;
    exit(EXIT_FAILURE);
}

void StationSet::mCalcCenter()
Appendix B: Source Code

```cpp
{ 
  for(int i=0;i<Data.size();i++)
  { 
    Data[i].mCalcCenter();
  }
}

int StationSet::mPrint(std::ostream& OutStream) const
{ 
  for(int i=0;i<Data.size();i++)
  { 
    Data[i].mPrint(OutStream);
  }
  return EXIT_SUCCESS;
}

int StationSet::mPrintNomFreqs(std::ostream& OutStream) const
{ 
  for(int i=0;i<Data.size();i++)
  { 
    Data[i].mPrintNomFreq(OutStream);
  }
  OutStream << std::endl;
  return EXIT_SUCCESS;
}
```
Bibliography


