Three-Dimensional Target Modeling with Synthetic Aperture Radar

A Thesis Presented to the
Electrical Engineering Department Faculty of
California Polytechnic State University
San Luis Obispo, California

In Partial Fulfillment
Of the Requirements for the Degree of
Master of Science in Electrical Engineering

By
John R Hupton
January 2009

© 2008

John R Hupton

ALL RIGHTS RESERVED

Committee Membership

TITLE: Three-Dimensional Target Modeling with Synthetic Aperture Radar

AUTHOR: John R Hupton

DATE SUBMITTED: January 2009

COMMITTEE CHAIR: Dr. John Saghri, Associate Professor

COMMITTEE MEMBER: Dr. Jane Zhang, Assistant Professor

COMMITTEE MEMBER: Dr. Wayne Pilkington, Assistant Professor

Abstract

Three-Dimensional Target Modeling with Synthetic Aperture Radar

John R Hupton

Conventional Synthetic Aperture Radar (SAR) offers high-resolution imaging of a target region in the range and cross-range dimensions along the ground plane. Little or no data is available in the range-altitude dimension, however, and target functions and models are limited to two-dimensional images. This thesis first investigates some existing methods for the computation of target reflectivity data in the deficient elevation domain, and a new method is then proposed for three-dimensional (3-D) SAR target feature extraction.

Simulations are implemented to test the decoupled least-squares technique for high-resolution spectral estimation of target reflectivity, and the accuracy of the technique is assessed. The technique is shown to be sufficiently accurate at resolving targets in the third axis, but is limited in practicality due to restrictive requirements on the input data.

An attempt is then made to overcome some of the practical limitations inherent in the current 3-D SAR methods by proposing a new technique based on the direct extraction of 3-D target features from arbitrary SAR image inputs. The radar shadow present in SAR images of MSTAR vehicle targets is extracted and used in conjunction with the radar beam depression angle to compute physical target heights along the range axis. Multiple inputs of elevation data are then merged to forge rough 3-D target models.

The project is a continuation of prior SAR research at Cal Poly under Dr. John Saghri with the sponsorship of Raytheon Space & Airborne Systems.

Acknowledgements

This thesis project would not have been possible without the generous support, advice and assistance from Dr. John Saghri. I also express my gratitude to Raytheon for sponsoring Cal Poly's SAR research and allowing many students such as myself to study exciting topics and participate in meaningful projects. Dr. Saghri deserves additional thanks for his efforts in obtaining Raytheon's support.

I thank Dr. Wayne Pilkington and Dr. Jane Zhang for their participation and advice as members of the thesis committee.

I thank Jeff Gerfen and Neil Hockaday for providing countless opportunities to gain experience in multiple fields of engineering, and for their trust in my abilities these last few years. I have learned more in working for them than I ever expected to, and will take away a great deal of useful skills thanks to their projects.

I thank the other members of Cal Poly's SAR and BalloonSat research groups, Dustin Blackwell, Robert Hursig, Jessica Kiefer, Scott Seims and Matthew Schlutz, for their continued assistance and participation, and for the supportive fellowship we have developed with one another on these projects.

Finally I would like to thank my parents, sisters, roommates, and friends for all of their motivation, encouragement and financial support during my time at Cal Poly, and for their patience with my continued graduation delays.

Table of Contents

TABLE OF FIGURES	X
1 INTRODUCTION	1
1.1 Document Overview	1
1.2 3-D SAR Methods	2
1.3 Proposed Shadow Technique for 3-D SAR Feature Extraction	2
1.4 Additional Study with Non-SAR Imaging	3
1.5 Software Platform	4
2 BACKGROUND	5
2.1 Synthetic Aperture Radar	5
2.2 Platform and Geometry	6
2.3 Target Functions and Image Reconstruction	7
2.4 Image Set	9
2.5 Shadows	10
2.6 Automatic Target Recognition	11
2.7 3-D SAR	12
2.8 Research at Cal Poly	13
3 THREE-DIMENSIONAL SAR METHODS	14
3.1 Concept	14
3.2 Direct Fourier Inversion	15
3.2.1 Circular SAR (CSAR)	15
3.2.2 Elevation Circular SAR (E-CSAR)	19
3.3 Dual-Antenna Interferometric SAR	22
3.4 High-Resolution Spectral Estimation	27
3.4.1 RELAX Algorithm	28
3.4.2 Decoupled Least-Squares for Maximum Likelihood Estimates	34
3.5 Summary	41
4 HIGH-RESOLUTION SPECTRAL ESTIMATION SIMULATIONS	42
4.1 Implementation	42
4.1.1 Input Test Data	42
4.1.2 Gradient-Based Cost Function Minimization	44
4.1.3 Dynamic Reflector Quantity Estimation	47
4.1.4 Plotting and Elevation Mapping	48
4.2 Testing and Analysis	50
4.2.1 Output Plots and Elevation Maps	50
4.2.2 Accuracy and Sensitivity	58
4.3 Conclusions	68
5 IMAGE PROCESSING METHODS	69
5.1 Level Thresholding	69
5.2 Edge Detection	70
5.2.1 Edge Detection Methods	70
5.2.2 Canny Edge Detection	70

5.3	Hough Transform	72
5.4	Blobbing	74
5.5	Median Filtering	75
5.6	Masked Filtering	77
5.6.1	Process	77
5.6.2	Masks	77
6	SHADOW TECHNIQUE FOR 3-D SAR FEATURE EXTRACTION	79
6.1	Purpose	79
6.2	Concept	80
6.3	Process Description	83
6.3.1	Step One: Object Detection	83
6.3.2	Step Two: Shadow Detection	85
6.3.3	Step Three: 3-D Heights Slice Creation	87
6.4	Testing and Complications	88
6.5	Slice Model Output	90
7	3-D TARGET MODELING FOR THE SHADOW TECHNIQUE	92
7.1	Rotation and Orientation	92
7.1.1	Automatic Rotation	93
7.1.2	Embedded MSTAR Rotation	95
7.2	Fusion Technique and Model Forming	96
7.3	Results	97
7.3.1	Model Output	97
7.3.2	Measurement and Accuracy	102
7.3.3	Practical Limits to Comparison with Spectral Estimation	108
8	ADDITIONAL STUDY WITH NON-SAR MODELING	109
8.1	Process Description	109
8.1.1	Step One: First-Pass Edge Detection	110
8.1.2	Step Two: First-Pass Hough Transform	110
8.1.3	Step Three: Border Mask Creation	112
8.1.4	Step Four: Second-Pass Edge Detection	113
8.1.5	Step Five: Second Pass Hough Transform	114
8.1.6	Step Six: Edge Image Creation	114
8.1.7	Step Seven: Finding an Inner Vertex	114
8.1.8	Step Eight: Border Vertex Filtering	117
8.1.9	Step Nine: Object Recreation in 2-D Space	119
8.1.10	Step Ten: Object Recreation in 3-D Space	121
8.2	Results	125
9	CONCLUSIONS AND FUTURE WORK	126
9.1	High-Resolution Spectral Estimation	126
9.1.1	Technique Comparison	126
9.1.2	Dynamic Reflector Quantity Estimation	127
9.2	Shadow Technique	128
9.2.1	Incorporation with ATR Algorithms	130
9.2.2	Process Improvements	130
9.2.3	Slice Quality and Failure Detection	132
9.3	Additional Study with Non-SAR 3-D Modeling	132
9.4	Summary and Final Thoughts	132

BIBLIOGRAPHY	134
APPENDIX A: SPECTRAL ESTIMATION SIMULATION CODE	137
Driver File: dls3D.m	137
File: getPixelData.m	141
File: getImageData.m	142
File: get2S1Data.m	143
File: getSLICYData.m	144
File: getConstraints.m	146
File: estimate.m	147
APPENDIX B: SPECTRAL ESTIMATION SIMULATION TEST MATRIX	148
APPENDIX C: SHADOW TECHNIQUE TEST SETS	154
Set 1: 2S1 at 30° with 4 Inputs and MSTAR Rotation	154
Set 2: BRDM_2 at 45° with 3 Inputs and Automatic Rotation	155
Set 3: D7 at 15° with 4 Inputs and Automatic Rotation	156
Set 4: 2S1 at 15° with 5 Inputs and Automatic	157
Set 5: BRDM_2 at 15° with 4 Inputs and Automatic Rotation	158
Set 6: BRDM_2 at 30° with 5 Inputs and Automatic Rotation	159
Set 7: ZSU_23_4 at 45° with 5 Inputs and Automatic Rotation	160
Set 8: 2S1 at 45° with 6 Inputs and MSTAR Rotation	161
Set 9: BRDM_2 at 17° with 3 Inputs and MSTAR Rotation	162
Set 10: 2S1 at 17° with 3 Inputs and MSTAR Rotation	163
Set 11: 2S1 at 15° with 3 Inputs and MSTAR Rotation	164
Set 12: ZSU_23_4 at 17° with 3 Inputs and MSTAR Rotation	165
Set 13: ZSU_23_4 at 15° with 3 Inputs and MSTAR Rotation	166
Set 14: D7 at 17° with 5 Inputs and Automatic Rotation	167
Set 15: T62 at 17° with 3 Inputs and MSTAR Rotation	168
Set 16: T62 at 17° with 4 Inputs and MSTAR Rotation	169
Set 17: BTR_60 at 17° with 3 Inputs and Automatic Rotation	170
Set 18: BTR_60 at 17° with 4 Inputs and MSTAR Rotation	171
Set 19: BTR_60 at 15° with 4 Inputs and MSTAR Rotation	172
Set 20: T62 at 15° with 5 Inputs and Automatic Rotation	173
Set 21: D7 at 17° with 3 Inputs and MSTAR Rotation	174
Set 22: ZSU_23_4 at 30° with 4 Inputs and Automatic Rotation	175
APPENDIX D: SHADOW TECHNIQUE CODE	176
Driver File: sar3D.m	176
File: getObject.m	180
File: getShadow.m	182
File: getSlice.m	183
File: fuseSlices.m	184
File: doHough.m	184
File: blobsClose.m	185
APPENDIX E: NON-SAR MODELING IMAGES	186
Test Image 1	186
Test Image 2	188
Test Image 3	190
Test Image 4	192

Test Image 5	194
Test Image 6	196
Test Image 7 (Failure)	198
Test Image 8 (Failure)	199

APPENDIX F: NON-SAR MODELING CODE	200
--	------------

Driver File: reg3D.m	200
File: filterEdges.m	201
File: doHough.m	201
File: getMask.m	203
File: getObjVertices.m	204
File: recreateObj2D.m	208
File: recreateObj3D.m	211
File: panView.m	216

Table of Figures

Figure 2.1: SAR Geometry	6
Figure 2.2: SAR Image Reconstruction Signal and Variable Descriptions	9
Figure 3.1: CSAR Platform	15
Figure 3.2: CSAR T-72 Target Slices at Various Elevations.....	18
Figure 3.3: E-CSAR T-72 Target Slices at Various Elevations	21
Figure 3.4: E-CSAR T-72 2-D Elevation Map	22
Figure 3.5: Dual-Antenna Interferometric SAR Geometry.....	23
Figure 3.6: Dual-Antenna Interferometric SAR Altitude Computation.....	24
Figure 3.7: Dual-Antenna Interferometric SAR Image and Interferogram.....	25
Figure 3.8: Dual-Antenna Interferometric SAR 3-D Surface Plot of Elevation Map	26
Figure 3.9: Dual-Antenna Interferometric SAR Elevation Sensitivity	26
Figure 3.10: RELAX Algorithm Top-level Flow Chart for $L = 3$	30
Figure 3.11: RELAX Algorithm Amplitude and Elevation Computation Sub-process	31
Figure 3.12: RELAX Algorithm Side-View Output Images	33
Figure 3.13: Decoupled Least-Squares Process for Maximum Likelihood Estimates.....	37
Figure 3.14: Decoupled Least Squares Pre-processing Interpolation Method.....	39
Figure 3.15: Decoupled Least-Squares Estimated Points in 3-D Space.....	40
Figure 4.1: Simulation: Reflector Points with Fabricated Tank Target Input; $M = 10, K = 3$	51
Figure 4.2: Simulation: Reflector Points with MSTAR 2S1 Target Input; $M = 3, K = 3$	52
Figure 4.3: Simulation: Reflector Points with MSTAR SLICY Target Input; $M = 9, K = 3$	53
Figure 4.4: Simulation: Elevation Map with Fabricated Tank Input; $M = 10, K = 3$	54
Figure 4.5: Simulation: Elevation Map with MSTAR 2S1 Target Input; $M = 3, K = 3$	54
Figure 4.6: Simulation: Elevation Map with MSTAR SLICY Target Input; $M = 9, K = 3$	55
Figure 4.7: MSTAR SLICY Target Example Photograph	55
Figure 4.8: Maximum Resolvable Altitude in Simulation	56
Figure 4.9: Simulation Accuracy: Input Samples M	59
Figure 4.10: Simulation Accuracy: Expected Reflectors K	60
Figure 4.11: Simulation Accuracy: Variance of Noise σ^2	61
Figure 4.12: Simulation Accuracy: Spread of Depression Angles.....	62
Figure 4.13: Simulation Example of a High-Altitude Spurious Reflector	63
Figure 4.14: Simulation Example of Spurious Reflectors with Dynamic K Estimation	66
Figure 4.15: Simulation Example of Spurious Reflectors with K overestimated ($K = 4$).....	66
Figure 5.1: Canny Edge Detection Example Images	72
Figure 5.2: Hough Transform Example Images.....	74
Figure 5.3: Blobbing Example Images	75
Figure 5.4: Median Filtering Example Images	76
Figure 5.5: Polygonal Mask Creation Example Images.....	78
Figure 6.1: Shadow Technique Radar Shadow Geometry	81
Figure 6.2: Shadow Technique Per-Column Target Object Height Computation	82
Figure 6.3: Shadow Technique MSTAR Input Example Images.....	83
Figure 6.4: Shadow Technique Object Detection Example Images.....	85
Figure 6.5: Shadow Detection Example Images.....	87
Figure 6.6: Jagged-Edge Shadow Example Images.....	89
Figure 6.7: Partial Shadow Detection Example Images.....	89
Figure 6.8: Shadow Technique Output Example Image Set	90
Figure 6.9: Shadow Technique Output Example Slice Image	91
Figure 7.1: Rotation of Asymmetric Target Example Images	94
Figure 7.2: MSTAR Header Excerpt	95
Figure 7.3: MSTAR Target Photographs.....	98
Figure 7.4: Shadow Technique 3-D Model Output Example #1: 2S1	99
Figure 7.5: Shadow Technique 3-D Model Output Example #2: BTR_60.....	100
Figure 7.6: Shadow Technique 3-D Model Output Example #3: BTR_60.....	101

Figure 7.7: Shadow Technique 3-D Model Test Set Measurement Results	103
Figure 7.8: Accuracy of Shadow Technique 3-D Model Test Sets.....	104
Figure 7.9: Accuracy of Shadow Technique 3-D Model Test Sets By Rotation Extraction Method.....	105
Figure 7.10: Accuracy of Shadow Technique 3-D Model Test Sets By Depression Angle	105
Figure 7.11: Accuracy of Shadow Technique 3-D Model Test Sets By Number of Input Images.....	105
Figure 7.12: Accuracy of Shadow Technique 3-D Model Test Sets By Target Vehicle	106
Figure 8.1: Non-SAR Algorithm First-Pass Hough Transform Example Image	111
Figure 8.2: Non-SAR Algorithm Arithmetic Solution of Two Lines	117
Figure 8.3: Non-SAR Algorithm Vertex Finding Example Images.....	118
Figure 8.4: Non-SAR Algorithm Outer Vertex Angles Example Image	120
Figure 8.5: Non-SAR Algorithm 3-D Space Transformation Arithmetic.....	122
Figure 8.6: Non-SAR Algorithm Example Output Image	124

1 Introduction

Much research has been done at California Polytechnic State University (Cal Poly) in the realm of Synthetic Aperture Radar (SAR) imaging with Dr. John Saghri, and in particular with Automatic Target Recognition (ATR). This thesis project focused on expanding that research into new realms and experimenting with the prospect of extracting three-dimensional (3-D) models and target features from available two-dimensional (2-D) SAR data. Current SAR and ATR techniques lie primarily in the two ground plane dimensions of an imaged SAR region, and some methods will be discussed for the extraction of data in the third dimension: the elevation domain of the target region.

1.1 Document Overview

This thesis can be broken down into four primary topics: a summary of the existing documented methods for three-dimensional SAR, a set of simulations testing one of these methods, a new technique for 3-D SAR, and the testing and analysis of the new technique.

Because the context of this project at Cal Poly is aimed at target recognition and long-range reconnaissance, the characteristics of ATR applications are directly considered when summarizing the existing methods for 3-D SAR. It is shown that there are existing methods which meet some of the practical requirements of an ATR reconnaissance system, but no existing technique is yet completely sufficient. This is demonstrated through simulations which highlight the capabilities and shortcomings of a leading technique for 3-D SAR in the context of target modeling for ATR.

Following the simulation assessments, a new technique for 3-D SAR feature extraction and target modeling is proposed and tested. The new technique seeks to meet those requirements which the existing methods fall short of, and in turn aims to create suitable 3-D target models given a relatively small number of arbitrary 2-D SAR image inputs. Put simply, the new technique is designed to be able to create target models given an input data set that would be practically attainable in a reconnaissance application.

1.2 3-D SAR Methods

Chapter 3 includes a review of past methods for the creation of 3-D SAR images, target models, and SAR data in the range-altitude dimension. Some of these methods are shown to be useful in the creation of rough 3-D models of targets and target regions, but are somewhat limited by restrictive platform and data requirements, making them often unsuitable for an ATR application.

Chapter 4 gives a description of the implementation and results of a set of simulations on one of the reviewed 3-D SAR techniques: high-resolution spectral estimation. The code and test data for these simulations are included in Appendices A and B.

1.3 Proposed Shadow Technique for 3-D SAR Feature Extraction

A new technique is proposed with the goal of circumventing some of the restrictions and limitations of prior 3-D SAR methods. The technique aims to extract as much 3-D information from a SAR image as possible, and involves expanding upon earlier Cal Poly SAR research, as well creating new methods for the processing of SAR images.

This “shadow technique” utilizes the extraction of the radar shadow ^[15] and the object boundaries from a SAR image of a target, and used the geometry of the radar’s depression angle and basic trigonometry to compute an estimate of the target’s per-column height.

A more in-depth description of the algorithm developed can be found in Chapters 6 and 7 and the code itself can be found in Appendix D. Chapter 5 outlines a set of important image processing procedures which were used as part of the shadow technique.

The data and information extracted from a single SAR image of a target was used to create rough 3-D models of the physical targets. The models were rotatable and viewable in the MATLAB environment and could be compared to images of the actual target vehicles.

This process is written to accommodate any SAR image input, and it is not confined to the test base (MSTAR image set) that was used in this project, but can be used with any arbitrary SAR input set, as long as a target region and shadow region are available.

1.4 Additional Study with Non-SAR Imaging

To gain a better understanding of the tools and processes with which 3-D information could be extracted from a 2-D SAR image, the minor portion of research consisted of an attempt at extracting 3-D information from a simple, grayscale, higher resolution non-SAR image set. The goal was to extract the 3-D measurements of a basic figure in a 2-D image matrix and translate the shape to a 3-D image matrix containing the same figure.

Several grayscale images were collected of a basic box shape in a relatively simple background environment and an algorithm was written to extract the figure's 3-D characteristics. The description of this algorithm can be found in Chapter 7, and the code and images can be found in Appendices E and F.

1.5 Software Platform

All of the software was written in the MATLAB programming environment using MATLAB version 7.5.0, available from The Mathworks Inc. This platform choice was made for three reasons: the large number of image processing methods built into the environment, its built-in ability to handle matrices (images), and the ease with which images (both 2-D and 3-D) could be viewed, displayed and analyzed. In addition, this environment significantly reduces the amount of time needed for debugging and altering a top-level algorithm.

2 Background

2.1 Synthetic Aperture Radar

Radio detection and ranging (radar) has been in common use in remote sensing and military applications since the middle of the 20th century ^[22], and is a very useful tool in measuring many features and attributes about a target or object from a great distance.

Radar in general is based upon the premise that transmitted electromagnetic waves are reflected by many objects, and the reflected waves can be received by the transmitter and processed to determine physical position, target material reflectivity, and other information.

Synthetic Aperture Radar expands upon this concept to create high-resolution 2-D imagery that can be post-processed even further in order to determine even more useful information, such as the size or type of the reflecting target ^[4]. SAR works by placing the radar antenna upon a moving platform so that radar returns and data can be collected about a single target or location from multiple vantage points at different times. The collected set of data is then post-processed and various natural phenomena are accounted for (including the Doppler effect) in order to produce an accurate and relatively high resolution 2-D image. The term “synthetic aperture” refers to the effective virtual aperture that occurs because of the data collected at different viewpoints, which yields a much better resolution in the cross-range (or azimuth) axis (perpendicular to the radar beam) and yields image quality that would be possible only with a much larger physical radar antenna aperture ^[12].

2.2 Platform and Geometry

In the context of the SAR study that has been done at Cal Poly, the assumed radar platform for the creation of any SAR images is a moving aerial platform with a side-looking radar antenna^[7]. This common platform is very useful for imaging slow moving target objects, because the relative speed of a ground vehicle is much smaller compared to the speed of the radar platform, which can be any high-speed aircraft (e.g. U2, Global Hawk, etc.)^[12].

The image in Figure 2.1 shows a geometric description of the radar platform and the target object relative to each other^[10]. In the image, the platform and antenna are currently at the point of minimum distance from the target itself, referred to as the point of minimum slant range.

Figure 2.1: SAR Geometry

The SAR imaging technique shown in the figure is referred to as stripmap SAR, in which the radar beam is always pointed in the same direction relative to the platform, and the

area illuminated by the beam is constantly changing. Spotlight-mode SAR imaging is another type in which the antenna direction is constantly adjusted such that the radar beam is always illuminating a particular target or area of interest.

It is important to note at this point that the angle between the ground plane and the radar beam will become important later on. This angle is referred to as the depression angle; it is used as part of the shadow technique for 3-D SAR feature extraction described in Chapters 6 and 7.

2.3 Target Functions and Image Reconstruction

The desired output of a SAR is a reconstruction of the true target function, a 2-D function representing the radar reflectivity of the illuminated target region at all points in the ground plane. This function, or image, is typically denoted by $f(x,y)$ where x and y represent the range and cross-range dimensions. The range dimension increases away from the antenna platform, while the cross-range dimension increases with the direction of movement of the platform.

The available data collected by the SAR antenna system consists of a function typically designated $s(t,u)$, where t represents time in the range dimension (fast-time) and u represents the synthetic aperture time domain of the cross-range dimension (slow-time). This is simply the signal returned to the radar antenna from the target region.

Reconstruction of the target function, when dealing with spotlight-mode SAR, typically involves fast-time matched filtering, in this case multiplying the frequency domain version of the returned signal $s(t,u)$ by a modified, phase-shifted form of the complex

conjugate of the transmitted radar signal. The phase-shifting is simply to account for the fact that there is a constant central target position and a non-constant platform position when dealing with spotlight-mode SAR.

The two-dimensional frequency domain version of the reconstructed target function for spotlight-mode SAR, then is ^[19]

$$F[k_x(\omega, k_u), k_y(\omega, k_u)] = S(\omega, k_u) \cdot P^*(\omega) \cdot \exp(jk_x X_c + jk_y Y_c). \quad (2.1)$$

In Equation 2.1, $k_x(\omega, k_u)$ and $k_y(\omega, k_u)$ are domain transformations necessary for the creation of 2-D target function reconstructions in the desired x - y Cartesian plane. These transformations are ^[19]

$$k_x(\omega, k_u) = \sqrt{4k^2 - k_u^2}; \quad k_y = k_u. \quad (2.2)$$

Figure 2.2 shows the descriptions of each signal and variable shown in the reconstruction formulas for Equations 2.1 and 2.2. The only inputs that need to be known to reconstruct the target image as a 2-D spatial (image) domain function are the measured signal, the transmitted signal, and the central position of the illuminated target region, a known and measureable feature of spotlight-mode SAR.

The actual spatial domain target function is obtained by performing 2-D Inverse Discrete Fourier Transform (or Fast Fourier Transform) on the function $F[k_x(\omega, k_u), k_y(\omega, k_u)]$. This function $f(x, y)$ can then be plotted in 2-D space as a regular grayscale image.

$S(\omega, k_u)$	The measured signal returned to the SAR antenna
$P^*(\omega)$	The matched filter complex conjugate of the transmitted radar signal.
k_x	Spatial frequency domain in the range dimension.
k_y	Spatial frequency domain in the cross-range dimension.
ω	Fast-time frequency domain (frequency domain of t).
k_u	Slow-time frequency domain (frequency domain of u).
X_c	Central position of the target region in the range dimension.
Y_c	Central position of the target region in the cross-range dimension.
k	Radar wavenumber: $k = \omega/c$.

Figure 2.2: SAR Image Reconstruction Signal and Variable Descriptions

2.4 Image Set

The same image set used for the previous Cal Poly research was used in this project. The Moving and Stationary Target Acquisition and Recognition (MSTAR) data is a large set of publicly-available spotlight-mode SAR images of eight known target vehicle types. For each target type, imagery is available at multiple depression angles, including 15°, 17°, 30° and 45°. The target is orientated in the center of each MSTAR image at angled rotations between 0° and 359°.

The eight different target types are named 2S1, BRDM2, BTR60, D7, SLICY, T62, ZIL_131, and ZSU_23_4. Seven of the targets are actual vehicles, but the SLICY target was designed as a very simple, stationary radar target with many standard radar-reflecting shapes in order to assist in the development of SAR algorithms ^[7]. This target also has SAR images created from additional depression angles: 16°, 29°, 31°, 43°, and 44°.

MSTAR images are of a low pixel resolution, typically either 54 x 54 or 158 x 158, and are created by the SAR processing algorithms to have an apparent viewing angle from directly above the target (a straight-down look). The radar position with respect to the image is below the target.

The MSTAR data set was collected using a 9.6GHz radar center frequency and a 0.6GHz radar bandwidth. The expected physical resolution per pixel in the images is 0.3047 meters, or approximately 1 foot, and the pixel spacing is approximately 0.2 meters.

2.5 Shadows

Shadows are an important feature of SAR imagery that results directly from the geometric relationship between the radar platform and the target. Essentially, while the SAR image itself is created to be a straight-down viewpoint of the target, the actual radar which collected the image data was outside of the image region and it illuminated the target from a certain angle (the depression angle). From an imaging perspective, this physical aspect means that there are certain regions in the image to which the radar beam did not actually penetrate (since it could not penetrate through a target object). Because

the beam did not reach these regions, no radar scattering points were detected by the SAR processing algorithms and most of the pixels in these regions register as very dark.

In addition, radar shadows are relatively immune to a SAR phenomenon known as speckle noise. This type of image noise is caused by constructive interference from the processing of multiple wavefront data points in the raw SAR data. Essentially, an object in the image's target region produces radar backscatter with multiple pulses and is processed multiple times. Target shadows are somewhat resistant to this effect since the beam never penetrates into the shadow region and thus no backscatter occurs.

SAR shadows are described in greater detail in Chapters 6 and 7 where they are put to use as the primary tool in the "shadow technique" for extracting 3-D information of the physical height of target objects in SAR imagery.

2.6 Automatic Target Recognition

There are many applications for systems and algorithms which can automatically identify a target in a given SAR image, and most of the SAR research done at Cal Poly thus far has been in the context of Automatic Target Recognition ^[7]. The various different extracted image and target features are used separately (or in some sort of combination) in order to determine the type of target vehicle present in the image. The determined type is then compared to the true type of the input image and the results after all test images are processed are placed in a confusion matrix in order to assess the classification accuracy. A high measure of classification accuracy is extremely desirable in the quest to discern between friend and foe.

2.7 3-D SAR

Classical SAR systems utilize the motion of a physical antenna aperture to produce a second synthetic aperture in the axis of motion, referred to as the “slow-time” or cross-range axis ^[19]. This usually consists of a satellite or aerial platform moving along the cross-range axis, and yields high-resolution 2-D image reconstruction (range and cross-range) of a target region. Physical platform movement in a second domain (elevation) or input from multiple platforms at varied elevations allows for the extraction of data in the third target dimension and the creation of 2-D images at multiple elevations of the target region ^[19]. This produces a 3-D SAR image with typical SAR resolution in the range and cross-range axes, but much lower resolution in the range-altitude axis. Resolution in the elevation dimension is severely limited compared to the range and cross-range axes primarily due to physical and mathematical limitations of the system ^[19].

Chapter 3 discusses the basics of image reconstruction in the third dimension (through an extension of 2-D Fourier reconstruction), and shows some methods which have been used to overcome the Fourier resolution limits. Chapters 6 and 7 discuss a new method, developed as part of this thesis, which explores the possibility of directly extracting 3-D target features from arbitrary 2-D SAR image data. This new method is an attempt to overcome all of the barriers to practical implementation that the existing methods for 3-D SAR currently face.

2.8 Research at Cal Poly

SAR research at Cal Poly has encompassed the extraction of several 2-D image features from the available MSTAR image sets, including: rectangular fits, peaks, corners, edges, and several others. Additionally, there has been research on the feasibility of each of these features in creating accurate ATR algorithms, often using the Hausdorff distance as a classification metric ^[7]. Scott Seims is currently researching the use of Dempster-Shaffer probability theory and its applications in altering the ATR classification scheme.

Relative to this thesis project, the shadow technique for 3-D feature extraction described in Chapters 6 and 7 requires the extraction of the shadow region in a given SAR image. Dr. John Saghri and Andrew DeKelaita have created a basic method for shadow extraction, and the shadow extraction technique used in the later Chapters is loosely based upon that method ^[15].

3 Three-Dimensional SAR Methods

This chapter discusses the mathematics behind three-dimensional SAR and outlines three method types for creating 3-D SAR images: direct Fourier inversion, dual-antenna interferometric SAR, and high-resolution spectral estimation. Some examples of prior research are discussed for each of the three types of techniques.

3.1 Concept

As mentioned briefly in Chapter 2, physical motion of the platform in the elevation domain produces the capability to resolve targets in the range-altitude axis, providing data in the deficient z axis (depth or elevation). This capability is based on the same principles of aperture synthesis that allow for the resolution of targets in the y axis based on motion in the cross-range domain ^[19].

Three-dimensional SAR by direct Fourier inversion refers to the extension of the basic SAR formulas and equations to account for data in the range-altitude domain, yielding 3-D spatial and Fourier functions, rather than just 2-D functions ^[19]. The method of interferometric 3-D SAR based on a dual-antenna platform extends 2-D SAR image reconstruction by accounting for phase differences between the two antennas in the elevation dimension. The third method presented, high-resolution spectral estimation, encompasses more than one sub-technique for accurately estimating the contribution of multiple radar scatterers at different 3-D elevations in each 2-D (x,y) location of a SAR image. All three methods attempt to provide sufficient resolution in the data-deficient z axis.

3.2 Direct Fourier Inversion

The target function of a standard SAR image can be viewed as a three-dimensional target function $f(x,y,z)$ containing nonzero components only at $z = 0$, meaning the reconstructed image is the two-dimensional function $f(x,y,0)$. In reality, the image created at $f(x,y,0)$ has radar reflectivity contributions from reflectors at other altitudes above the $z = 0$ plane. A low-resolution three-dimensional SAR image can be created by mathematically quantifying target signatures above this plane in the reconstructed 2-D image $f(x,y,0)$ ^[19].

3.2.1 Circular SAR (CSAR)

The Circular SAR (CSAR) system shown in Figure 3.1 ^[2] involves an aerial platform rotating around a given target region, or a target “turntable” rotating adjacent to a fixed antenna platform.

Figure 3.1: CSAR Platform

In the side view of Figure 3.1, the radar elevation is shown to be equal to $Z_c + v$, where v is the new synthetic aperture in the elevation domain. In this section on CSAR, this synthetic aperture is ignored and the radar is assumed to be at a fixed altitude Z_c . Section 3.2 deals with Elevation Circular SAR (E-CSAR) in which the elevation synthetic aperture is included.

Consider the standard linear SAR platform in which the radar platform is moving along the line $x = X_c$ parallel to the cross-range y axis on the $z = Z_c$ plane. If u represents the slow-time synthetic aperture, the distance to a target reflector located at (x, y, z) is

$$\sqrt{(x - X_c)^2 + (y - u)^2 + (z - Z_c)^2}. \quad (3.1)$$

The target region appears, based on this formula, to be at a fixed slant-range, and reconstruction is performed irrespective of the target altitude. Because of this, the system has no capability to resolve targets in the range-altitude domain^[19].

A Circular SAR system, however, uses the circular slow-time synthetic aperture θ instead of the linear u . The distance to a target reflector, then, is given by the formula

$$\sqrt{(x - R_g \cos \theta)^2 + (y - R_g \sin \theta)^2 + (z - Z_c)^2}, \quad (3.2)$$

where R_g is the circular radius shown in Figure 3.1. In this equation, the slant-range does depend on the slow-time aperture θ ^[19]. The Fourier reconstruction on a CSAR target function at some altitude z is equal to

$$F_{xy}(k_x, k_y, z) = F(k_x, k_y, 0) \cdot \exp \left(-j \sqrt{k_x^2 + k_y^2} \cdot \tan \theta_z z \right) \quad (3.3)$$

The variable θ_z is the slant angle for mid-altitude, equal to $\theta_z = \arctan(Z_c / R_g)$, and k_x and k_y represent the spatial frequency domains for the range and cross-range axes. The function $F_{xy}(k_x, k_y, 0)$ is the two-dimensional Fourier reconstruction of the target function at $z = 0$ given by standard SAR reconstruction techniques (applied to the CSAR platform). The Fourier reconstruction of the target function at an altitude z is found by multiplying this 2-D ground plane reconstruction by the exponential phase function shown in Equation 3.3. The target function and desired image $f(x, y, z)$ can therefore be reconstructed using the 2-D inverse Discrete Fourier Transform (DFT) of $F_{xy}(k_x, k_y, z)$ ^[19].

This function is an approximation of the true target function and shows that a 2-D target image can be reconstructed for a given altitude z . The function itself contains a focused image of the target reflections from *all* altitudes at the given z plane, meaning that the output image matrix will appear smeared and out-of-focus because there are contributions in the image from other altitudes ^[19].

An experiment was performed by the Georgia Tech Research Institute (GTRI) containing CSAR data of a T-72 tank on a circular “turntable.” The method of direct inversion of a 2-D spatial Fourier target function was applied to the CSAR data to create 2-D SAR images of the target at various altitudes. These images are referred to as horizontal target slices ^[19].

Figure 3.2 shows three T-72 target slices at three different altitudes, where the z value shown is in meters and is referenced to the z value of the central altitude of the target region instead of the ground plane ^[19].

$z = -1.25\text{m}$
$z = 0.25\text{m}$
$z = 0.75\text{m}$

Figure 3.2: CSAR T-72 Target Slices at Various Elevations

Two calibration corner reflectors are included in the T-72 data and are also shown in the diagram of Figure 3.1. The three slice images show that physical reflectors at one elevation contribute to reflectivity data at all other elevations. This contribution is most evident with the tank cannon (seen jutting out to the right of the main body). At $z = 0.75\text{m}$ above the central target altitude, nearest to where the cannon is located, it appears in focus, but at other elevations it is spread into two locations. This phenomenon is due to the physical reflectivity of the cannon surface ^[19].

These images demonstrate that 3-D data can be extracted directly from raw CSAR returns using Fourier inversion of the target distribution, but no accurate 3-D model can be directly created from this data since reflections at different elevations appear in multiple slice images. The 3-D SAR techniques presented in Section 3.4 offer some solutions to the problem of estimating spectral contributions of reflectors at multiple elevations.

3.2.2 Elevation Circular SAR (E-CSAR)

The CSAR platform is expanded upon with linear motion of the platform in the elevation domain, effectively creating a second slow-time synthetic aperture, shown as v in Figure 3.1. This type of SAR is called Elevation Circular SAR (E-CSAR), and involves a spiral-like motion of the antenna platform.

Using the same system model as was used with CSAR, the measured SAR signal is represented by the 3-D function $s(t, \theta, v)$, where t is the fast-time domain, θ is the circular synthetic aperture, and v is the elevation synthetic aperture. This signal is transformed into the spatial frequency domain and becomes $S(\omega, \xi, k_v)$, where ξ and k_v are the spatial frequency domain variables for the circular and elevation synthetic apertures ^[19].

Image reconstruction for E-CSAR involves Fourier decompositions in both synthetic aperture domains. In the elevation domain, the equation that describes the Fourier response from the target is ^[19]

$$S(\omega, \theta, k_v) = P(\omega) \exp(jk_v Z_c) \iint F_z(x, y, k_v) \cdot \exp\left(-jk_r \sqrt{(x - R_g \cos \theta)^2 + (y - R_g \sin \theta)^2}\right) dx dy. \quad (3.4)$$

In this equation, $F_z(x, y, k_v)$ is the one-dimensional Fourier Transform of the target function with respect to the elevation synthetic aperture, and $P(\omega)$ is the Fourier Transform of the transmitted signal. The spatial frequency with respect to radius is equal to $k_r = \sqrt{4k^2 - k_v^2}$, where k is the radar wave number, or ω / c .

The complete Fourier Transform of the E-CSAR signal and the reconstruction of the 3-D Fourier target function then become ^[2]

$$S(\omega, \xi, k_v) = \mathcal{F}_{(\theta)}\{S(\omega, \theta, k_v)\} \\ F(k_x, k_y, k_z) = P^*(\omega) \cdot \exp(-jk_v Z_c) \cdot \mathcal{F}_{(\xi)}^{-1}\left\{S(\omega, \xi, k_v) \cdot H_{\xi}^{(2)*} \cdot (k_r R_g) \exp\left(-j\frac{\pi \xi}{2}\right)\right\} \quad (3.5)$$

In this equation for the 3-D Fourier target reconstruction, $H_{\xi}^{(2)*}$ is the complex conjugate of the Hankel function of the second kind, ξ order ^[2]. The output domain with spatial frequencies k_x , k_y , and k_z is defined by the following linear transformations: ^[2]

$$k_x(\omega, \theta, k_v) = \sqrt{4k^2 - k_v^2} \cos \theta; \\ k_y(\omega, \theta, k_v) = \sqrt{4k^2 - k_v^2} \sin \theta; \quad k_z(\omega, \theta, k_v) = k_v. \quad (3.6)$$

The spatial target function and desired image $f(x,y,z)$ can be obtained by performing the 3-D inverse Fourier Transform on the Fourier reconstruction.

The CSAR turntable data of the T-72 tank collected by the Georgia Tech Research Institute was used by the Institute in E-CSAR simulations and a 3-D matrix was created of the target region ^[2]. The Fourier resolution in the elevation axis was much higher than that of CSAR because of the addition of the second synthetic aperture.

Figure 3.3 shows three reconstructed slices of this data at various elevations, referenced to the central altitude of the target region ^[2].

$z = 0\text{m}$
$z = 0.38\text{m}$
$z = -0.38\text{m}$

Figure 3.3: E-CSAR T-72 Target Slices at Various Elevations

Figure 3.4 ^[2] shows an elevation map that was created based on the 3-D T-72 E-CSAR data. The elevation map is a 2-D matrix constructed from the altitude location of the maximum value of the 3-D target function for a given (x,y) location. The map appears to roughly follow the general pattern of the physical structure of the T-72 target (similar in shape and size to the T62 MSTAR target as shown in Figure 7.3).

Figure 3.4: E-CSAR T-72 2-D Elevation Map

3.3 Dual-Antenna Interferometric SAR

Another method for the collection of 3-D SAR data of a target region is interferometric SAR, which attempts to quantify information in the data-deficient range-altitude axis by assessing the phase difference in radar returns between two different antennas displaced in the elevation dimension ^[6]. This system is similar in some ways to stereo SAR, in

which two completely separate platforms and antennas are choreographed to pass by the target region from opposite sides and simultaneously collect conflicting lateral data.

In this section, a dual-antenna interferometric SAR system described by A. Currie is reviewed ^[6]. This system consists of a single aerial radar platform outfitted with four antennas mounted at different elevations on the platform, creating two separate transmit and receive channels ^[6].

The premise of interferometric SAR is that each pixel in a given SAR image shows information about the relative amplitude of target reflectivity only at that location, and the phase information is ignored for display in the image. Given two SAR images of the same target region from different elevations created at the same time, however, the phase difference between two corresponding pixels may contain useable information about the elevation domain of the target region ^[6].

A diagram of the dual-imaging system is shown in Figure 3.5 ^[6].

Figure 3.5: Dual-Antenna Interferometric SAR Geometry

In the diagram, the target region located at range x and altitude z is being imaged simultaneously by antennas A_1 and A_2 , located at slant ranges r_1 and r_2 . The physical positions of the two antennas are related by the distance d and the angles α and θ . The difference in phase between the two antennas then is ^[6]

$$\Phi = \frac{2\pi}{\lambda}(r_1 - r_2) = \frac{2\pi}{\lambda} \left[\sqrt{r_1^2 + d(d + 2r_1 \cos(\alpha + \theta))} - r_1 \right]. \quad (3.7)$$

In a practical SAR system, the parameters that are measurable include the platform altitude H (shown in the figure), the primary slant range r_1 , and the phase difference Φ between two imaged points in the target region ^[6]. Figure 3.6 shows the arithmetic that can be used to calculate the height z of the target point currently being imaged. The pixel can be plotted in 3-D space when coupled with the current range point x and the information from the platform slow-time aperture in the cross-range axis y .

Given H , r_1 , and Φ :

$$r_2 = r_1 + \frac{\Phi \lambda}{2\pi}$$

$$\theta = \cos^{-1} \left(\frac{r_2^2 - d^2 - r_1^2}{2dr_1} \right) - \alpha$$

$$z = H - r_1 \cos \theta$$

$$x = \sqrt{r_1^2 - (H - z)^2}$$

Figure 3.6: Dual-Antenna Interferometric SAR Altitude Computation

The image that can be created using this altitude computation is a 2-D matrix consisting of a “sheet” or surface function plotted in 3-D space, similar to the elevation map used with the GTRI data. The surface plotted is the corresponding (x,y) radar range and cross-

range data, while the z location each pixel is plotted at was computed with the equations in Figure 3.6.

Figure 3.7 shows a portion of a standard 2-D SAR image of a hilly region created with a dual-antenna SAR interferometer. Next to this SAR image is the interferogram that corresponds to the same image region, showing the phase information between the two antennas. An elevation map and 3-D surface plot created from the SAR image and its interferogram is shown in Figure 3.8 ^[6].

Figure 3.7: Dual-Antenna Interferometric SAR Image and Interferogram

Figure 3.8: Dual-Antenna Interferometric SAR 3-D Surface Plot of Elevation Map

The sensitivity of the computation of the z elevation value for each (x,y) location with respect to system parameters is important to quantify how accurate a particular elevation computation would be in a practical environment. The sensitivity of z with respect to the measurable, physical, and dependent parameters is shown in Figure 3.9 ^[6].

$\frac{\partial z}{\partial H} = 1$	$\frac{\partial z}{\partial r_1} = -\cos \theta$	$\frac{\partial z}{\partial \Phi} = -\frac{\lambda r_1 \sin \theta}{2\pi d \sin(\alpha + \theta)}$
$\frac{\partial z}{\partial d} = \frac{r_1 \sin \theta}{d \tan(\alpha + \theta)}$	$\frac{\partial z}{\partial \alpha} = -r_1 \sin \theta$	$\frac{\partial z}{\partial \lambda} = -\frac{r_1 \sin \theta}{\lambda \tan(\alpha + \theta)}$

Figure 3.9: Dual-Antenna Interferometric SAR Elevation Sensitivity

The equations in Figure 3.9 show that the computed height is highly sensitive to the measurable parameters in the interferometer, and errors in the input parameters will lead to incorrect elevation computations ^[6]. An accurate system will yield accurate results, but the system overall is sensitive to rapid changes in platform elevation with respect to the ground plane, and aerial platforms especially are very sensitive to inconsistencies in the ground plane between the platform and the target region ^[3].

3.4 High-Resolution Spectral Estimation

Section 3.2 discussed the use of Direct Fourier inversion of SAR frequency data in order to obtain a low-resolution 3-D function, sampled at various target elevations. In order to obtain a better resolution, a significant number of radar passes over the target region would be required, but this requirement is somewhat impractical. The dual-antenna SAR interferometer discussed in Section 3.3 can create a 3-D elevation map with only two inputs (on a single platform), but is sensitive to changes in platform altitude, creates only a surface representation of the target view, and would likely not be effective in imaging small, man-made target objects ^[3].

Additionally, neither of these methods accounts for the fact that complex terrain and many man-made objects in target regions (such as MSTAR target vehicles) contain multiple reflectors at different altitudes in the (x,y) grid ^[3]. The presence of multiple reflectors was visibly evident in the 3-D slices of CSAR data from Section 3.2.1, but the data in that form cannot be readily used to create accurate target models without consideration of the relative altitudes of the various reflectors.

Two techniques for high-resolution spectral estimation are presented in this section. The purpose of high-resolution spectral estimation is to directly account for the presence of multiple radar reflectors in the target region. Estimates are sought for three parameters: the number of reflectors in the z domain of a given (x,y) location, the reflective amplitude of each reflector, and the relative altitude of each reflector in the z axis.

3.4.1 RELAX Algorithm

Three similar algorithms have been developed for the purpose of spectral estimation in radar target regions: CLEAN, RELAX, and ESPRIT ^{[3] [9]}. The RELAX relaxation algorithm is covered in this section and the results from a prior practical experiment are shown ^[3].

The RELAX algorithm intends to iteratively account for the contributions from individual reflectors in the z axis until the total accumulated energy accounted for is above some arbitrary noise threshold. Once the algorithm has completed its run, the only remaining energy from the original frequency samples is assumed to be noise energy. One immediately evident advantage to this technique is that the noise itself does not need to be identified and the type is irrelevant. Thus, the algorithm can be effective in many different noise environments ^[3].

The algorithm assumes a signal $s(t)$ is composed of L complex sinusoids, each located at altitude z_i with additive noise signal $n(t)$ ^[3]. The signal can be expressed as

$$s(t) = \sum_{i=1}^L g_i e^{-j2\pi z_i t} + n(t). \quad (3.8)$$

A further assumption states that for a given (x,y) location in the radar target region there are at most L nonzero reflectors and the total reflectivity as a function of altitude z (or altitude spatial frequency k_z) is

$$g_{x,y}(z) = \sum_{i=1}^L g_{x,y,i} \delta(z - z_{x,y,i}) \xleftrightarrow{\mathcal{F}(z)} G_{x,y}(k_z) = \sum_{i=1}^L g_{x,y,i} e^{-j2\pi z_{x,y,i} k_z}. \quad (3.9)$$

In this formula, the reflectors at a given (x,y) location are denoted by $z_{x,y,i}$, and their contributions sum to comprise the total reflectivity $g_{x,y}(z)$ for some given z in the (x,y) grid. These functions are effectively the data produced by Fourier inversion of individual Fourier sheets acquired by a single SAR pass^[3]. Thus, it is necessary to acquire more than $2L$ SAR sample sets of the target region in order to accurately estimate both the reflectivity and elevation of each reflector in the z axis of a given (x,y) grid location^[3].

A flow chart of the RELAX algorithm is provided in Figure 3.10. In the flow chart, a discrete-time signal of frequency samples $A[n]$ is assumed (where $0 \leq n \leq N$), with the number of reflectors L in the z axis equal to 3 or less (in this case). Two arbitrary noise cutoffs γ_1 and γ_2 are set to terminate the algorithm when enough of the total signal energy is accounted for.

As the flow diagram indicates, the algorithm is flexible enough to account for a varied number of reflectors, with L indicating only the maximum number allowable. The noise cutoffs are used to effectively yield an estimate for the number of reflectors present in the z region, since the process will be aborted once a threshold has been met and the current number of reflectors accommodates for the majority of the signal energy.

The sub-process shown in the flow chart in which g_i and z_i are computed for the individual values of i is described in greater detail in Figure 3.11 ^[3].

Figure 3.10: RELAX Algorithm Top-level Flow Chart for $L = 3$

Figure 3.11: RELAX Algorithm Amplitude and Elevation Computation Sub-process

The algorithm could be extended further for larger values of L . For example, for $L = 4$, there would simply be a third arbitrary noise cutoff γ_3 and there would be an additional convergence loop when calculating g_1 and z_1 through g_4 and z_4 ^[3].

The algorithm depends upon rotating between computations of different g_i and z_i values and re-accounting for their contributions until the point at which g_1 through g_3 and z_1 through z_3 no longer change and the loop has reached convergence. For each additional increase in the number of allowable reflectors L , an additional convergence loop will be required.

In practice, the available SAR data comes in the form of polar Fourier samples of a signal $G(k_x, k_y, k_z)$. These samples are linearly interpolated onto a discrete Cartesian plane and a two-dimensional inverse DFT is performed in the range and cross-range dimensions to

obtain discrete samples of the signal $G(x, y, k_z)$, which is the signal that will be used as the input to the RELAX algorithm for each given (x, y) location in the image matrix.

A prior simulation of the RELAX algorithm used 3-D *xpatch* data for two different target vehicle models: a tank and an airplane. The *xpatch* software tool creates realistic radar signatures for a wide variety of man-made and natural targets.

Figure 3.12 shows a side view of the simulation results for both the tank and the airplane targets^[3]. Three images are provided for each target: the real-world target model, the image that would be created if methods of direct Fourier inversion were applied, and the image created by plotting the (g_i, z_i) scatterer data estimated by the RELAX algorithm.

The most significant feature of the RELAX images is the lack of vertical smearing present when compared to the image formed by Fourier inversion. The number of points on the targets that is accounted for is approximately equal between the two images (although the RELAX algorithm did pick up one sample of the tank cannon), but the RELAX algorithm accurately accounted for multiple scatterers at different elevations in the z axis, and the out-of-focus smearing was removed.

To produce these models and images, 15 sheets of 128 x 128 Fourier data were created at various elevations with two different frequency bands, centered around 9.75GHz and 9.8GHz^[3].

A grayscale side-view image of a tank, showing its turret, main gun, and tracks.	A grayscale side-view image of a fighter jet, showing its wings, tail, and fuselage.
A grayscale image showing the Fourier inversion result for the tank model, appearing as a noisy, abstract pattern.	A grayscale image showing the Fourier inversion result for the plane model, appearing as a noisy, abstract pattern.
A grayscale image showing the RELAX algorithm result for the tank model, appearing as a noisy, abstract pattern.	A grayscale image showing the RELAX algorithm result for the plane model, appearing as a noisy, abstract pattern.
<p>Tank model (top)</p> <p>Fourier inversion image (middle)</p> <p>RELAX image (bottom)</p>	<p>Plane model (top)</p> <p>Fourier inversion image (middle)</p> <p>RELAX image (bottom)</p>

Figure 3.12: RELAX Algorithm Side-View Output Images

3.4.2 Decoupled Least-Squares for Maximum Likelihood Estimates

A second technique for high-resolution spectral estimation of radar scatterer amplitudes and altitudes is presented by Walter S. Kuklinski and Andrea L. Kraay^[8]. This technique aims to provide the Maximum Likelihood (ML) estimate of scatterer strengths and locations based on a decoupled least-squares process of amplitude estimation. Like the RELAX algorithm, data is processed with frequency domain samples available in the z domain for each given location in the (x,y) grid, and z locations and strengths are sought iteratively.

The environment is modeled in the same way, with a slight variation. For each (x,y) location in the range/cross-range grid, the frequency domain data is modeled as

$$S_{x,y}(\omega_z) = \sum_{k=1}^K a_k^* e^{-j\omega_z z_k} = \mathbf{a}^H \mathbf{v}(\omega_z) = \mathbf{v}^H(\omega_z) \mathbf{a}. \quad (3.10)$$

Here, the scatterer strengths are denoted by a_k^* instead of g_i , the number of scatterers for a given (x,y) is denoted K instead of L , and frequency is kept in radian form (for indexing simplicity later). The signal model is also written in matrix form, where the superscript H represents the matrix Hermitian operator (transpose conjugate)^[8]. Placing the frequency domain data samples in a matrix form allows for the formulation of least-squares Maximum Likelihood estimates in a matrix equation.

As before, the number of reflectors K , the reflector strength matrix $\mathbf{a}$, and the reflector locations z_1 through z_K are the parameters to be estimated. The estimates altogether form

an estimated version of the frequency domain data signal referred to as $\hat{S}(\omega_z)$ such that the estimates adhere to a least-squares model and the cost function C is minimized: ^[8]

$$C = E \left[|S(\omega_z) - \hat{S}(\omega_z)|^2 \right]. \quad (3.11)$$

In order to obtain a least-squares estimate for the scatterer strengths, the matrices in the model are expanded and a suitable model for environment noise is added ^[8]. The new data sample signal (with noise added) is referred to as matrix $\mathbf{y}$, a vertical matrix indexed based on different values of the frequency ω_z . A new matrix $\mathbf{V}$ is defined, which encompasses the $\mathbf{v}$ matrix for all samples of ω_z . The noise model used is a zero-mean complex Gaussian white noise vector, and the overall system model, in expanded matrix form, is ^[8]

$$\mathbf{y} = \mathbf{V}^H \mathbf{a} + \mathbf{n}$$

$$\begin{bmatrix} y(\omega_{z1}) \\ y(\omega_{z2}) \\ \vdots \\ y(\omega_{zM}) \end{bmatrix} = \begin{bmatrix} \mathbf{v}^H(\omega_{z1}) \\ \mathbf{v}^H(\omega_{z2}) \\ \vdots \\ \mathbf{v}^H(\omega_{zM}) \end{bmatrix} \mathbf{a} + \mathbf{n} \quad (3.12)$$

The collected SAR data amounts to M frequency samples for each (x,y) location in the grid, and thus the dimensions of each matrix are:

$$\mathbf{y} = M \times 1 \quad \mathbf{V} = M \times K \quad \mathbf{a} = K \times 1 \quad \mathbf{n} = M \times 1.$$

By least-squares, the maximum likelihood estimate of scatterer amplitudes, equivalent to an estimated version of matrix $\mathbf{a}$, is ^[8]

$$\hat{\mathbf{a}}_{ML} = (\mathbf{V}\mathbf{V}^H)^{-1}\mathbf{V}\mathbf{y}. \quad (3.13)$$

Overall, the primary inputs required to the system are SAR frequency data samples in the z domain for each (x,y) location. The noise is already accounted for in the Maximum Likelihood formula for the estimate in Equation 3.13. The only other undetermined parameter is the number of scatterers K in z for each (x,y) , and this parameter is assumed to be a small number and is simply kept constant for all locations in the range/cross-range grid ^[8].

In order to determine the position in the z axis of the estimated scatterers, the possible positions of the reflective scatterers must be limited to a finite set of positions. However, this limitation does not need to adhere to the Fourier resolution (equal to the height of the region z to be estimated divided by the number of frequency samples M). A suitable value for the minimum spacing of scatterers when using the decoupled least-squares technique is equal to one-fourth of the Fourier resolution limit ^[8].

The least-squares technique centers on minimizing the cost function C , and the formula for the scatterer amplitude estimate $\hat{\mathbf{a}}_{ML}$ relies on the position of each scatterer z_k (through the computation of the $\mathbf{v}$ matrices). For each (x,y) , every possible physical position configuration of K reflectors in the z grid is considered and a mean-squared error (the cost function C) is computed for each configuration. Once this exhaustive search is completed, the minimum error configuration is chosen as the Maximum Likelihood estimate of reflector position and amplitude, and a set of points in z has been collected for the given (x,y) location ^[8].

To avoid the large set of computations required for this exhaustive search, a gradient-based algorithm could be used after each computation of $\hat{\mathbf{a}}_{ML}$ and the current estimates

for scatterer locations could be updated based on this algorithm (instead of simply considering all possible locations). As Kuklinski and Kraay note, this gradient technique would have to utilize the gradients in a multi-dimensional function of mean-squared error plotted against the various indexes of the scatterer locations. Thus, for K scatterers, the cost function would be plotted against K dimensions ^[8].

Both exhaustive searching and gradient-based minimization will yield the Maximum Likelihood estimates for strength and location, and a generalized flow diagram showing the process is shown in Figure 3.13.

Figure 3.13: Decoupled Least-Squares Process for Maximum Likelihood Estimates

In the procedure of Figure 3.13, the decision to continue looping with new reflector locations is based on the method of minimization: exhaustive search or gradient-based search. The parameters and constraints of the search algorithm will be used for this decision in the gradient method, and the exhaustive search will complete when *all* possible reflector locations have been tested.

Prior to processing, the raw SAR frequency data must be placed in the correct form, similar to the pre-processing that was done for the RELAX algorithm. The data samples must be interpolated and placed in the (x, y, ω_z) dimensions for the creation of the $\mathbf{y}$ matrix. The interpolation is a function of several parameters: the number of resolution bins in the range dimension of SAR data collection N_x ; the mean, minimum, and maximum depression angles of data collection $(\phi_0, \phi_{\min}, \phi_{\max})$; the number of SAR data collections M ; and the bandwidth and center frequency of the radar. It also varies directly with the index of the range bin being processed x , which increases with physical range from the radar. The formula to compute the frequency spacing of the ω_z samples, in radians per meter, is ^[8]

$$d\omega_z(x) = \frac{4\pi}{c} \left[f_c + \frac{BW}{N_x} \left(x - \frac{(N_x - 1)}{2} \right) \right] \cdot \frac{[\tan(\phi_0 - \phi_{\min}) - \tan(\phi_0 - \phi_{\max})]}{M}. \quad (3.14)$$

The center frequency of ω_z samples is equal to $\omega_c = 2\pi \left(\frac{2f_0}{c} \right)$ and the formula for frequency sampling allows the set of M standard (x, y) SAR images to be linearly interpolated onto the (x, y, ω_z) grid as shown in Figure 3.14 ^[8]. The samples of ω_z determined from this interpolation do not depend upon the index in the cross-range y

dimension, so the figure shows only the two dimensions needed. It is worth noting that the formula for $d\omega_z(x)$ implies that all frequency samples must be *evenly spaced*. This means that the set of depression angles that the input data sets were collected at must also be evenly spaced in order to obtain correct results.

Figure 3.14: Decoupled Least Squares Pre-processing Interpolation Method

Kuklinski and Kraay created a 3-D image matrix of a tank model using $M = 9$ SAR data sets from *xpatch* simulation data. The minimum scatterer spacing was set to one-fourth the Fourier resolution limit. Figure 3.15^[8] shows two side views of the reflectors found by the decoupled least-squares technique overlaid on top of the view of the target model. The reflector points are highlighted green. Next to the decoupled least-squares model are the Fourier inversion estimates (in red) with the typical vertical smearing suppressed by

including only points that contain 75% of the total energy for a given (x,y) bin ^[8]. The unlabeled axes in the figure are the range and cross-range dimensions.

Figure 3.15: Decoupled Least-Squares Estimated Points in 3-D Space

The images in Figure 3.15 show that the decoupled least-squares technique for high-resolution spectral estimation of target reflectors yielded a significantly larger number of points than the Fourier inversion technique, and since the allowable spacing between

scatterers was one-fourth the Fourier spacing, the individual scatterer altitudes were more accurate.

One detriment of the technique that is obvious when viewing these images is that a noticeable number of erroneous point scatterers were plotted, especially when compared to the Fourier model. However, there is a greater concentration of points around the scattering centers of the target object, and a much larger percentage of the target is resolved, with important features of the model easily highlighted ^[8].

3.5 Summary

This chapter presented three types of 3-D SAR imaging techniques, including two methods for high-resolution spectral estimation of target reflector contributions. The RELAX algorithm and decoupled least-squares techniques account for a greater number of variables, target types, and target scattering signatures than Fourier inversion and interferometry did, and they yielded more promising results.

Chapters 6 and 7 explore a new method for the extraction of 3-D information from SAR imagery. The new technique attempts to surmount some of the restrictive constraints that are characteristic of Fourier inversion, dual-antenna interferometry, and high-resolution spectral estimation. Three-dimensional target feature extraction is explored through the exploitation of image attributes and the geometry of spotlight-mode SAR data collection.

4 High-Resolution Spectral Estimation Simulations

To test the decoupled least-squares technique for high-resolution spectral estimation of reflector contributions, some simple simulations were written in the MATLAB environment. These simulations were executed with varied input parameters to determine the sensitivity of the technique to changes in data collection, system modeling, and environment. Inputs to the simulations included both actual SAR data (MSTAR) and fabricated complex radar scattering data coupled with additive noise.

4.1 Implementation

The implementation of the decoupled least-squares technique in simulation was based on the mathematical description of the technique by Kuklinski and Kraay^[8] and the flow chart and equations given in Section 3.4.2. The technique was altered slightly for the sake of testing it against some of the parameters, but the Maximum Likelihood method for the estimation of reflector locations and contributions is the same.

4.1.1 Input Test Data

Three sets of input data were created to test the decoupled least-squares algorithm; one was fabricated data based on the formula for the model of the complex scattering radar signal in Equation 3.10, and two were data from MSTAR target images which were interpolated into sets of frequency samples in the ω_z domain in the manner also described in Chapter 3.

The equation for the frequency-domain model of the radar scattering returns (from Equation 3.10) is repeated here ^[8]:

$$S(x, y, \omega_z) = \sum_{k=1}^K a_k^* e^{-j\omega_z z_k}. \quad (4.1)$$

For each (x, y) location, the set of input data is a vector of complex frequency-domain sums. Each complex sample of the $S(x, y, \omega_z)$ signal is the sum of the contributions of all scatterers in the range-altitude z axis at the given location in the (x, y) grid.

The fabricated data set was actually begun with a fabricated set of scatterer amplitudes and locations in the z axis for a 16 x 16 2-D grid of (x, y) locations. This set of scatterers in the (x, y) grid was made to roughly resemble a possible scattering pattern of a man-made tank-like target vehicle, complete with the surface scattering of the turret and cannon. For each (x, y) location in the input grid, the set of scatterer amplitudes and locations a_k and z_k are used in Equation 4.1 for $S(x, y, \omega_z)$ to create a frequency sample, and in each (x, y) location there are M samples. The value of ω_z for each sample is determined by the current range location x , the center frequency of data collection ω_c , and the frequency spacing between samples $d\omega_z$. This is shown by the following equations from Chapter 3:

$$\begin{aligned} \omega_c &= 2\pi \left(\frac{2f_0}{c} \right) \\ d\omega_z(x) &= \frac{4\pi}{c} \left[f_c + \frac{BW}{N_x} \left(x - \frac{(N_x - 1)}{2} \right) \right] \\ &\quad \cdot \frac{[\tan(\phi_0 - \phi_{\min}) - \tan(\phi_0 - \phi_{\max})]}{M}. \end{aligned} \quad (4.2)$$

A complex Gaussian White noise sample was added to each of the M frequency samples, just as described by Kuklinski and Kraay^[8]. The target region, then, is modeled as a man-made target present in a region of Gaussian White scatterer noise. The mean amplitude of these noise samples was zero and the standard deviation was initially equal to one, while the maximum amplitude of any “true” scatterer (before the noise was added) was 16.4. The standard deviation of the noise was varied as a test parameter to determine the technique’s ability to resolve scatterers in a variety of noise environments.

For the two simulation inputs of MSTAR data, the magnitude and phase of the complex SAR images were used as the frequency samples of the signal $S(x, y, \omega_z)$, meaning that it was assumed that all complex scatterer contributions were embedded in the complex value of the SAR images at each (x, y) location. No noise was added to these samples, since the environment and process noise were already part of the complex values in the SAR image.

Each SAR image was interpolated into the (x, y, ω_z) domain by determining ω_z from the formula for frequency spacing and the depression angle of the image relative to the others. To attain M frequency samples, M complex SAR images were required.

4.1.2 Gradient-Based Cost Function Minimization

As mentioned in Chapter 3, for each (x, y) location in the input matrix $S(x, y, \omega_z)$, the location and amplitude of the K assumed scatterers in z was found by an iterative search for the minimum of the mean-squared error cost function. Figure 3.13 shows a flow chart of the process for each grid location.

To find the set of scatterer locations and amplitudes that yields the minimum of the cost function, two methods were possible: exhaustive search of all possible configurations, and gradient-based function minimization. In the implemented simulations, a gradient-based method of minimization was utilized with the aid of MATLAB's Optimization Toolbox. The cost function to be minimized is computed with the following formula, where the $\mathbf{y}$ matrix is the set of frequency samples of $S(x, y, \omega_z)$ at the current (x, y) location, and $\hat{\mathbf{a}}_{ML}$ is the current estimate of scatterer amplitudes:

$$MSE_{x,y,a} = (\mathbf{y} - \mathbf{V}^H \hat{\mathbf{a}}_{ML})^H (\mathbf{y} - \mathbf{V}^H \hat{\mathbf{a}}_{ML}). \quad (4.3)$$

The MATLAB function `fmincon()` implements a constrained optimization based on a Sequential Quadratic Programming technique. Within each execution of the internal function loop, an estimate of the Hessian of the Lagrangian of the cost function is computed, and the next values of the input estimates are computed based on the direction of this second derivative.

The `fmincon()` function accepts a single-input, single-output arbitrary software method which returns a cost function to be minimized, as well as a set of linear constraints on the type of input the cost function accepts. In this case, the arbitrary function to be minimized takes in an input vector of K current estimates for reflector locations, computes the Maximum Likelihood estimate of reflector amplitudes (as described in Chapter 3), and returns the mean-squared error of the estimate.

The linear constraints placed on the minimization of the cost function are based on the upper bound for the maximum elevation that can be successfully imaged L_z and the

minimum spacing between scatterers dz (chosen to be one-fourth the Fourier resolution).

The equations for these two constraints are

$$L_z = \frac{2\pi}{d\omega_z} \text{ meters; } dz = \frac{L_z}{4 \cdot M} \text{ meters.} \quad (4.4)$$

For each (x,y) location, the cost function minimization is begun by initializing the reflector location estimates to some set of reasonable values, then calling the `fmincon()` function with that initial estimate and the set of linear constraints as parameters. The reason for initializing the reflector locations with a set of reasonable values is that it increases the likelihood of finding the global minimum of the cost function instead of a local minimum. This would not be a concern if an exhaustive search method were used for minimizing the cost function, but the risk of using a gradient-based search is that it does not necessarily ensure finding a global minimum. If, however, the general region of the global minimum is known, then initializing the gradient search in that region increases the probability of the function returning the global minimum.

As described by Kuklinski and Kraay, a good estimate of the region of the global minimum is obtained by initializing the reflector locations at the K highest values of the one-dimensional Inverse Discrete Fourier Transform (IDFT) of the frequency samples at the location (x,y) ^[8]. Equivalently, the initial estimates for scatterer locations are the K highest values of ^[8]

$$s_{x,y}(z) = \left| \frac{1}{M} \sum_{i=0}^{M-1} S(\omega_{z,i}) \cdot e^{j\omega_{z,i}z} \right|. \quad (4.5)$$

The gradient-based method of minimization requires far fewer function iterations and a much shorter runtime than the exhaustive method would, and still outputs the reflector locations that would yield the minimum value of the mean-squared error cost function.

4.1.3 Dynamic Reflector Quantity Estimation

The decoupled least-squares technique calls for a predetermined and constant estimate for K , the number of possible reflectors in the z axis for every (x,y) . The simulations performed by Kuklinski and Kraay assumed that the number of scatterers in z was $K = 3$ for every (x,y) position in the grid ^[8]. However, there are many situations in which there could be more or fewer reflectors in z , and better results may be attained with different estimates for K , so the decoupled least-squares technique was modified slightly to account for this.

The simulations were written to accommodate any estimate of K greater than or equal to $K = 1$, and additionally the simulations can accommodate a version of the decoupled least-squares estimation in which the number of reflectors in z is dynamically estimated in each (x,y) location. This dynamic estimation is based on the minimization of the same mean-squared error cost function that was used before.

When dynamic estimation of the quantity of reflectors is used, the `fmincon()` gradient-based minimization function is called three times for each (x,y) , once for each value of K between 2 and 4. Only these three values of K are used since considering too many would require significant additional processing time and because most man-made radar targets would consist of a set of scatterers of that relative amount ^[19].

For each value of K , the mean-squared error of the Maximum Likelihood estimate of scatterer locations and amplitudes is recorded, and once all three Maximum Likelihoods are recorded for $K = \{2,3,4\}$, the value of K which yielded the lowest mean-squared error is chosen as the global minimum. The minimized set of reflector locations and magnitudes that were computed with this K are then used as the best estimate for the reflectivity signature in z .

The simulations were executed with a variety of static values for K and using dynamic estimation of K , to determine the usefulness of this estimation.

4.1.4 Plotting and Elevation Mapping

The output matrix created by the decoupled least-squares process simulation of an (x,y) grid with a set of K estimated pairs of reflector amplitudes and their z locations. For viewing the target function that is described by this set of estimations, individual scatterers in the matrix are plotted one-by-one in a 3-D grid.

During the computation of the estimated scatterer locations and amplitudes, and for each (x,y) location in the grid, the energy in the input signal is computed (the energy in the matrix y). After computation of the estimates for all (x,y) points, the maximum input signal energy is found. Then upon plotting, to filter out the scatterers that are most likely products of process noise, the only (x,y) points plotted are those in which the energy contribution from all the estimated scatterers sums to an energy value greater than or equal to 25% of the maximum value of energy in the entire input matrix. This helps to ensure that the scatterers which are plotted contain enough signal energy to likely be part

of a true “physical” scatterer. Kuklinski and Kraay used a similar method for refining the plotted set of reflector points in the images of Figure 3.15 [8].

A similar threshold requirement is applied for individual scatterers in the set of K scatterers at each (x,y) that meets the energy requirement. If the amplitude of the reflector is not greater than or equal to 25% of the maximum reflector amplitude in the set of K , then the reflector will not be plotted. This is to prevent the plotting of extra reflectors at a given point when K happens to be overestimated at that point.

Additionally, for ease of viewing the estimated target structure, the scatterer at each (x,y) with the highest amplitude is plotted with a red dot, and other scatterers that pass the stricter threshold of 50% of the maximum amplitude are plotted with a blue dot. All scatterers with amplitudes between 25% and 50% of the maximum amplitude are plotted in 3-D space with green dots.

Just as important as the reflective signature of all scatterers is an estimate of the true physical elevation of the target at each (x,y) . To create an elevation map of the target region, the same method that was used to create an elevation map of the E-CSAR data in Chapter 3 is used with the decoupled least-squares estimation. For each (x,y) that meets the same 25% energy criteria of above, the z location of the scatterer with the highest amplitude is plotted in a 2-D elevation map matrix. This matrix is then viewable as a surface function in 3-D space.

4.2 Testing and Analysis

4.2.1 Output Plots and Elevation Maps

Figure 4.1 through Figure 4.3 show output plots for each of the three types of simulations, and Figure 4.4 through Figure 4.6 show the elevation maps. For each of these simulations, the number of scatterers K in each (x,y) was kept static at $K = 3$. The number of frequency samples M for the first simulation (with the fabricated tank input data) was $M = 10$, and the number of frequency samples (input images) M for the two MSTAR simulations was based on how many images were available of the two targets at different depression angles. For the MSTAR simulation with the 2S1 target, only $M = 3$ image samples were available with a spread between the angles of 30° . For the SLICY target simulation, $M = 9$ frequency samples were available, but were unevenly spaced between 15° and 45° , so the simulation adjusted to account for inconsistent changes in ω_z . Figure 4.7 shows a sample photograph of the SLICY target for reference.

The quality of the MSTAR simulations is considerably lower than that of the fabricated input, and the altitudes shown in the plots (measured in meters) are not accurate, as they are severely limited vertically. This is almost entirely due to the lack of available data as input to the process. From Equations 4.2 and 4.4 for the frequency spacing between images $d\omega_z$ and the maximum resolvable target altitude L_z ^[8], it is notable that both the number of input samples M and the spread between the depression angles of the inputs have a significant effect on the size of the region that can be imaged in z . Figure 4.8 shows a 3-D plot of the effect these two input factors have on L_z .

Figure 4.1: Simulation: Reflector Points with Fabricated Tank Target Input; $M = 10$, $K = 3$

Figure 4.2: Simulation: Reflector Points with MSTAR 2S1 Target Input; $M = 3$, $K = 3$

Figure 4.3: Simulation: Reflector Points with MSTAR SLICY Target Input; $M = 9$, $K = 3$

Figure 4.4: Simulation: Elevation Map with Fabricated Tank Input; $M = 10$, $K = 3$

Figure 4.5: Simulation: Elevation Map with MSTAR 2S1 Target Input; $M = 3$, $K = 3$

Figure 4.6: Simulation: Elevation Map with MSTAR SLICY Target Input; $M = 9$, $K = 3$

Figure 4.7: MSTAR SLICY Target Example Photograph

Figure 4.8: Maximum Resolvable Altitude in Simulation

The plot in Figure 4.8 shows the relationship between the maximum resolvable target altitude and the number and spread of the input images. It was created using the center frequency, radar bandwidth, and range resolution values of the MSTAR data set, such that only the depression angle spread and number of samples were varied.

As the plot shows, the resolvable altitude ceiling is influenced heavily by these two input parameters, and therefore the ability of the decoupled least-squares technique to accurately resolve scatterers will be severely hindered when either of these parameters is unfavorable. This relationship is demonstrated clearly in the two simulations which used all available MSTAR data of a target scene. In the 2S1 simulation, the number of

samples M was quite small and the spread between the minimum and maximum depression angles was large, yielding an extremely low altitude ceiling (0.1m) for the z region which could be imaged. In the SLICY simulation, the value of M was sufficiently large ($M = 9$), but the spread between the minimum and maximum depression angles was still very large (30°), leading to an improved (0.9m), but still largely insufficient maximum altitude.

With respect to the plots of the reflector locations and the elevation maps for each simulation, it is visible that in all simulations some spurious reflector points met the 25% energy and amplitude thresholds for plotting. The majority of these spurious points, however, did not appear in the elevation map since the map only shows the altitude of the highest amplitude reflector. Also, many of these spurious points were in the target region, and the set of (x,y) grid points that needed to be plotted was limited primarily to the target region, as no reflectors outside that region met the 25% energy threshold.

The exception is the simulation of the 2S1 MSTAR target, in which a slightly larger number of spurious reflectors fell outside of the target region. This is a product of two factors: the altitude ceiling and the relative reflectivity of the target (compared to the reflectivity of the SLICY target). The SLICY target is a set of corner reflectors intended to give high amplitude radar signatures, and the relative amplitude of the reflectors in that target would be higher with respect to environment noise than the reflectors of a 2S1 ^[7]. Furthermore, the maximum resolvable altitude in the 2S1 simulation was very low for reasons already mentioned, and at that elevation the reflectivity signature of the 2S1 target might not have enough energy to overpower the background noise.

4.2.2 Accuracy and Sensitivity

To assess the accuracy and sensitivity of the decoupled least-squares simulation against a variety of input parameters, four test metrics were used with the fabricated tank input data simulation. The first two metrics were the mean-squared error and median-squared error of the difference between the maximum amplitude scatterer at *all* points in the input and output grids. The second two metrics were the mean-squared error and median-squared error of the intended fabricated elevation of the target and the elevation values found in the output elevation map. The reason for using both mean- and median-squared errors is that the median-squared error was expected to be less sensitive to the existence of spurious reflector points. This proved to be the case, and made a difference in the resultant plotting patterns between the two error metrics for some of the input parameters.

The four input parameters tested were: the number of input samples M , the number of expected reflectors K (including dynamic estimation), the variance of the Gaussian sample noise σ^2 , and the spread between the minimum and maximum depression angles in the input set, measured in degrees. When varying one parameter, all other parameters were kept at their “default” values: $M = 10$, $K = 3$, $\sigma^2 = 1$, and angle spread = 4° .

Figure 4.9 through Figure 4.12 show the mean- and median-squared error plot pairs for all four tested input simulation parameters. The analysis and discussion of these plots and the results they show will follow. In Figure 4.10, the plots of the error metrics against the number of expected scatterers K , the x -axis value shown as ‘*’ denotes the use of dynamic estimation of the number of reflectors, as described in Section 4.1.3.

Figure 4.9: Simulation Accuracy: Input Samples M

Figure 4.10: Simulation Accuracy: Expected Reflectors K

Figure 4.11: Simulation Accuracy: Variance of Noise σ^2

Figure 4.12: Simulation Accuracy: Spread of Depression Angles

The plots of the error metrics with respect to the number of input samples M in Figure 4.9 show the greatest discrepancy between the mean- and median-squared error plotting patterns. The reason for this is directly related to the computation of the maximum resolvable altitude L_z , discussed in the previous section. An increase in M leads to a linear increase in L_z , which is helpful with respect to the accuracy of the altitude of most scatterers. However, it also means that the altitudes of the spurious and incorrect scatterers can be much larger. In fact, this is the case with many of the simulations run with a very large number of input samples M . Figure 4.13 shows a side view example of this with $M = 25$, and one spurious reflector at approximately $z = 11$ m.

Figure 4.13: Simulation Example of a High-Altitude Spurious Reflector

The direct result of the existence of even just a few high-altitude false reflectors is that the mean of the squared errors increases considerably. The median-squared error, however, is more resistant to changes with outlier points, and the plot of the median-squared errors with respect to M shows more a favorable (and expectable) pattern of decreasing error with an increase in the amount of input data.

The results shown in Figure 4.10 show the accuracy results with respect to the expected number of reflectors K . The fabricated input data target used for these simulations had between 1 and 4 reflectors at each (x,y) location, but the majority of the grid had 3 reflectors. Thus, the simulation results with $K = 3$ should yield the best results amidst the three simulations which used static values for K , and this is confirmed with three out of the four metrics in Figure 4.10.

A pattern that seems to arise out of the plots in Figure 4.10 is that it is better to underestimate K than overestimate it. There were approximately the same number of points in the input grid with $K = 2$ scatterers as there were points with $K = 4$ scatterers, and yet the errors when K was set to 2 in simulation were lower. This is likely because when overestimating K , it is a guarantee that at least one reflector will be incorrect. When K is underestimated, the reflectors found will (hopefully) be at locations at which there is a true reflector, and the estimated amplitudes of the resolved reflectors will simply be overestimated to account for the extra energy in the signal. Overestimating K will lead to an increase in the number of false reflectors and likely to a *decrease* in each reflector's amplitude, in turn leading to a direct increase in the error metrics.

The results shown with dynamic estimation of the number of reflectors are troubled with the same problem. These results are favorable when compared to $K = 4$, but there are still enough (x,y) points in the dynamic estimation simulations in which K was estimated to be 4 that the overall error metrics are higher than when K was underestimated. It is fairly clear, though, for all four metrics, that dynamically estimating K is better than overestimating it.

Figure 4.14 shows an example side view output plot when K was dynamically estimated. In this case, the vertical region which can be imaged, determined by L_z , is of a relatively desirable size, so the altitude of any spurious reflectors is limited. However, the number of these false reflectors is quite high, due to the number of points at which dynamic estimation of K yielded an overestimated number of reflectors.

Figure 4.15 shows a similar example side view, but with K statically estimated at 4. In this figure, there are even more spurious reflectors shown, since the value for K was overestimated at the vast majority of points in the (x,y) grid.

Even though the amplitude of these false scatterers is typically not high for dynamic K estimation (denoted by blue and green dots), their effect on the error metrics is clear, due only to the sheer number of them when compared to the number of false reflectors with K underestimated or accurately estimated.

In both Figure 4.14 and Figure 4.15, all reflectors shown above $z = 11\text{m}$ are false reflectors.

Figure 4.14: Simulation Example of Spurious Reflectors with Dynamic K Estimation

Figure 4.15: Simulation Example of Spurious Reflectors with K overestimated ($K = 4$)

The accuracy of the simulations with varied noise in Figure 4.11 followed an expectable trend, with all four error metrics increasing fairly consistently with increases in the variance of the Gaussian White noise samples. The error increases were relatively slow, and it appears that a good amount of input data ($M = 10$) can overcome a noisy set of frequency samples.

The changes in accuracy with respect to the spread between the minimum and maximum depression angle of the input set were much more extreme. As Figure 4.12 shows, the magnitude of the errors, particularly the median-squared errors, were higher in this set of tests than they were when varying any other parameter. The increases in all four errors between successive simulations were also more severe, even though the largest depression angle spread tested was not that large (10°).

Figure 4.12 demonstrates clearly why the simulations with MSTAR target data did not yield desirable results. The quality of output data will be considerably better if the set of passes of the SAR antenna over the target region is confined to a finely spaced grid of depression angles^[8]. A wider spread of depression angles directly leads to a significant reduction in the ability of the decoupled least-squares technique to accurately resolve scatterer altitudes and amplitudes.

The spread of depression angles with the MSTAR simulations is 30° , which is three times the maximum angle spread tested in Figure 4.12, and more than four times the maximum spread shown for the computation of L_z in Figure 4.8. Since the spread of depression angles with MSTAR was so wide, the $d\omega_z$ frequency spacing between each input image was simply too high to achieve precise results.

4.3 Conclusions

The method of high-resolution spectral estimation using the decoupled least-squares estimates of target reflector locations and magnitudes was sufficiently accurate in simulations where the set of input parameters was favorable. The technique also proved to be sufficiently insensitive to the magnitude of the input noise signal. However, in simulation with actual SAR data (the MSTAR target images), the decoupled least-squares technique was shown to be very sensitive to the type and amount of input data available.

This technique for spectral estimation clearly requires a decent number of input frequency samples (preferably more than 4 SAR images), and the spread between the minimum and maximum depression angles of these images must be kept very small (less than 10°). In addition, high-resolution spectral estimation requires that the target pose in every input image be the same, meaning that the entire data set be created from the same view of the target. Without these crucial criteria being met, the results will be inaccurate, unreadable, and flooded with false data. Conversely, if these criteria *can* be adequately met without difficulty, the results are enough to create accurate elevation maps and 3-D models of the target vehicles.

5 Image Processing Methods

The proposed shadow technique for 3-D SAR feature extraction and target modeling, described in Chapters 6 and 7, uses numerous common image processing tools and techniques. This chapter describes some of these tools in detail.

5.1 Level Thresholding

Basic level thresholding of an image allows for an extremely quick separation of different logical image regions. The most common use of thresholding is to create binary images (where pixels can be one of only two types) from grayscale images like those of MSTAR. The basic principle is simply to define one or more threshold points in the range of possible grayscale pixel intensities and classify all pixels that exist on one side of the threshold as one class and all pixels on the other side as another class, creating an image that is quantized considerably more than the input (e.g. from 256 possible intensities down to only 2).

When converting from grayscale to binary, the challenge is in selecting an appropriate threshold. This threshold can simply be the midpoint in possible values (127 for an 8-bit image) or it can be based on some more complicated formula, such as the point that yields 50% of the pixels in each class (based on the histogram distribution of the image).

MATLAB has an implementation of a grayscale-to-binary image thresholding function called `im2bw()`, which can be given a threshold or can compute one automatically. The automatic threshold computation uses Otsu's method for threshold selection, calculating an optimum threshold that minimizes the variances within the two output classes.

5.2 Edge Detection

5.2.1 Edge Detection Methods

The goal of edge detection is to highlight specific pixels in an image that appears on the edge of a logical image region. An example in the context of target recognition in SAR images would be a set of pixels on the edge of a target object. For years, many 3 x 3 and 5 x 5 image “masks” have existed to perform an edge detection operation on 2-D images, including the Prewitt, Sobel, Laplacian and Roberts masks ^[20]. These static edge detection masks work fairly well at producing an output set of edge pixels, but the method used throughout the algorithms of this project is the Canny Edge Detector. This multistage detector is used because it is very flexible and can adapt to any image environment, including those with large amounts of unpredictable noise—as is the case with SAR imagery.

5.2.2 Canny Edge Detection

The Canny algorithm can be broken down into four major steps:

- 1) Gaussian filtering:

The input image is filtered using a Gaussian matrix mask filter to reduce image noise. The resulting output image is a very slightly blurred version of the input image.

- 2) Computing intensity gradient:

The gradient of the image is computed in several directions (since edges can appear at any angle). Usually the diagonal gradients are quantized to reduce

the runtime of the algorithm, but a gradient is often collected at angles quantized by 1-, 5-, 10-, or 15-degree increments.

3) Non-maximum suppression:

If the gradient at a particular location and angle happens to be a local maximum, then it is considered in the next step; otherwise it is ignored. For example, if the 45 degree gradient at one location happens to be a local maximum compared to the gradient to the northwest and southeast directions, then it is a maximum and is considered.

4) Gradient thresholding is applied to the maxima that are passed from step 3 and any places that meet the upper (and lower) threshold criteria are edge pixels. Edge pixels are set to 1 and pixels not on edges are set to 0, creating a binary output image ^[1].

The Canny edge detection algorithm is implemented in MATLAB as the function `edge()`, which allows the specification of threshold criteria and the Gaussian filter parameter `sigma` (standard deviation of the filter).

Two sets of example images are provided in Figure 5.1, which shows an input image and the image that results after running Canny edge detection using MATLAB's default parameters. The top set shows a raw MSTAR input image and its edge detected output, and the second set shows a processed, binary MSTAR image with clearly defined image regions and its edge detected output.

Figure 5.1: Canny Edge Detection Example Images

5.3 Hough Transform

Another commonly used tool in image processing is the Hough Transform, used to identify prominent lines (and possibly other shapes) in a binary image. It is a 2-D discrete transform that transforms a binary image into the Hough domain, or Hough Space, another 2-D matrix in which each pixel location represents a possible line in the input image^[21].

The transform works by considering every line that could possibly exist in the input image matrix, and describes each line by two features: distance from the origin (ρ) and angle of orientation (θ). For each of these possible lines, the number of white pixels in the input binary image that are on the line are counted and the count is placed in the Hough Space matrix, where vertical rows correspond to increasing distance from the origin (or the center) and the horizontal columns correspond to increasing angles of orientation ^[1].

For example, if the gray level of a pixel in the Hough Space is 125 and its row and column are 15 and 56, then 125 different white pixels (or pixels with a binary value of 1) existed in the input image on the line that is a distance 15 from the origin (or center) and oriented at 56 degrees below the top of the image.

The Hough Transform can be adapted to seek out circles, rectangles, and other basic 2-D geometric shapes in images ^[21], but during research it was used only to find prominent lines. MATLAB has a sequence of built-in routines including `hough()`, `houghpeaks()`, and `houghlines()` which perform the Hough transform, seek out logical peaks in the Hough Space, and return the endpoints of the lines that the logical peaks describe. The endpoints are found by following the line defined by the ρ and θ values of the Hough peak and noting the points at which the pixels on the line in the input image are *not* white.

Figure 5.2 shows a sample input image and its corresponding Hough Space image. In this sample, two maxima in the Hough Space correspond to the two lines in the input image.

Figure 5.2: Hough Transform Example Images

5.4 Blobbing

Image blobbing involves iterating through an input binary image matrix and finding contiguous blocks of white pixels. Each new blob is given a new label number in an output label matrix so that the blobs themselves can be characterized, counted, and sized. Because blobbing algorithms are quite simple and very common, the challenge in the method is due to the requirement that the input image be binary. In the context of SAR, the input images are grayscale and the pixels that should be blobbed must be highlighted during the binary conversion process.

A very small (12 pixels by 12 pixels) example is shown in Figure 5.3. Both the input image and the label matrix are shown, with the numbers in the label matrix representing the numbered blobs.

Figure 5.3: Blobbing Example Images

5.5 Median Filtering

Median filtering is often used to filter outlier pixels such as shot noise in grayscale and binary images. The process works in a similar fashion to the older edge detection methods, by using a 3 x 3 or 5 x 5 mask that is iterated through the input image in a raster fashion ^[20]. The level of the center pixel underneath the mask is replaced with the median value of the levels of all 9 or 25 pixel levels surrounding it. Effectively, when an outlier pixel is in the center of the mask (an isolated pixel that is very bright or very dim compared to those around it), it is replaced by a level that is less extreme. While this does sometimes blur an image very slightly and removes input information, it is also very effective at removing “salt and pepper” noise as well as stray white pixels in binary images.

Figure 5.4 shows two examples of median filtered images. The first is a grayscale example in which it is clear that there is a reduction in sharpness in the desired regions of

the image but the “salt and pepper” noise is also completely removed. The second example is a binary image in which stray white pixels are filtered, but the edges of large objects are eroded slightly.

Median filtering is implemented in MATLAB for 2-D matrices with the function `medfilt2()`.

Figure 5.4: Median Filtering Example Images

5.6 Masked Filtering

5.6.1 Process

Masked Filtering, or Region-of-Interest Filtering, is actually just an expansion of any other type of image filter or process to suit more specific needs. The idea is to perform an operation on an image—but only a certain subsection of the image. This subsection (the mask) is not simply another smaller 2-D image, but can be of any shape and size, and the operations performed on it can include edge detection, median filtering, or even the Hough Transform. The mask region can also be split into multiple portions in different sectors of the input image.

This method involves only two steps: creating the mask and running the filter. The mask itself is simply a 2-D image matrix of the same size as the input image in which white pixels (or 1s) correspond to pixels in the region-of-interest, where the filtering should apply. Black pixels are in the region that the filter ignores (leaves unchanged).

Masked filtering is implemented in MATLAB with the function `roifilt2()`. The function requires an input image, a binary mask of the same pixel size, and a function handle for the filtering operation.

5.6.2 Masks

MATLAB has several helper methods for creating masks interactively, based on pixel color and a set of vertices. Masks can also be created “manually” through some other algorithm. One of the available helper functions is `roipoly()`, which creates a closed polygonal image mask from a set of points in an image. The only stipulation for the use

of the function is that the mask it creates will be a closed polygon with vertices in exactly the same order as they were provided. In other words, if the vertices are not provided in a circular order, then crossovers and discontinuous regions will occur in the mask.

Figure 5.5 demonstrates this problem, as both images in the figure represent masks created with the same set of points, but the points were provided in different order. The image on the right is most likely the desired mask, as its vertices were provided in the numbered order, proceeding in a circle, and a single enclosed region was created.

Figure 5.5: Polygonal Mask Creation Example Images

6 Shadow Technique for 3-D SAR Feature Extraction

6.1 Purpose

The specific purpose of experimenting with a new method in SAR 3-D target modeling is to seek out a 3-D SAR imaging method of decent quality that is not as limited as the techniques of Chapter 3 were. The high-resolution methods of spectral estimation both required the availability of a relatively large number of antenna passes over the target region (at specific, very finely spaced altitudes), and in all of these passes the target pose, or rotation with respect to the radar, had to be consistent. These requirements could be impractical or unrealistic to meet in a reconnaissance or target recognition application.

This chapter outlines the specifics of the 3-D SAR shadow technique's concept and how it was implemented in MATLAB simulations. The shadow technique's aim is to a simple, fast and flexible algorithm that allows for arbitrary SAR target inputs, without requiring consistent target pose rotations or difficult platform requirements. It should also require less knowledge of the target region and fewer data collections than were required by the methods of Chapter 3.

Chapter 7 discusses the latter portion of this 3-D SAR technique with regards to fusing multiple feature inputs into a 3-D target model.

The technique itself relies on the use of radar shadows present in the SAR images. As a result, any MSTAR images that do not appear to have a radar shadow would not apply to the technique. However, this situation was not accounted for when testing and using the technique in later chapters, since a great majority of the available MSTAR images do

have visible and detectable shadows. Even for the images that do not have shadows (or where the shadow is barely discernable), the image was still processed in order to fully characterize the abilities of the technique on all available test data.

6.2 Concept

The shadow technique exploits the geometry of the inherent cause of the radar shadow and the physical orientation with which the MSTAR images were created. In all MSTAR images, the location of the radar is below the image, meaning that the shadow always falls above the target in the image. Furthermore, the SAR platform itself makes the radar depression angle (the angle between the ground plane and the radar beam) always readily known. All of this information is employed to extract height information from individual 2-D MSTAR images. This height information is in the form of heights that come “out of the page” towards the viewer, and only the target heights are computed, leaving the noisy background areas of the image filtered and ignored.

The algorithm builds on the premise that the depression angle and the height of the target are both directly related to the length of the shadow as observed from the bottom of the image (the direction of the radar platform). The target object, ground plane, shadow and platform effectively form a system of like triangles that, if measurable can yield the physical height of the target itself.

Figure 6.1 demonstrates this geometry. The target is labeled “T” and the area not hit by the radar beam is highlighted in gray. MSTAR images have an apparent viewing angle

directly above the target object, so the shadow length that would be seen in an image is labeled directly below ^[17].

Figure 6.1: Shadow Technique Radar Shadow Geometry

In order to compute the height of the target object in Figure 6.1, the angle θ and the shadow length are both needed. The angle, which is simply the depression angle of the radar beam, is known as a parameter of the input image, but the length of the radar shadow is a characteristic of a given SAR image. Therefore, in software, with MSTAR images, the distance between the upper edge of the shadow and the upper edge of the target object needs to be found for every given column of pixels. With this information, the length of the radar shadow can be calculated for every pixel column in which the target and shadow both appear.

The trigonometric formula used to calculate the height of the target object in each column is shown in Figure 6.2. In the formula, the height of the object in a given column is denoted by the variable H_c , the image row of the target object's border is denoted by O_r ,

and the border row or the SAR shadow is S_r . The units of the computed height are in pixels (physical measures depend on the physical resolution of the input SAR image).

A notable aspect of the formula is that the portion $(O_r - S_r)$ calculates the shadow length positively since images are indexed with the origin (column = 0, row = 0) at the top left of the image. As a result, with the radar beam coming from below the MSTAR images and the shadow appearing on top, the border rows for the target object have higher row indices than the border rows for the shadow.

$$H_c = (O_r - S_r) \cdot \tan(\theta)$$

Figure 6.2: Shadow Technique Per-Column Target Object Height Computation

Figure 6.3 shows some example input MSTAR images that demonstrate some of the important factors regarding the extraction of the shadow and the height of the target object in each column. Each image is labeled with the radar depression angle at which the image was taken.

Three things are evident in these images:

- The radar shadow always appears above the target object because of the position of the radar.
- Some images are darker and it is more difficult to discern the shadow from the rest of the image (both visually and in software).
- The radar shadow increases in vertical length with a decrease in the radar depression angle.

Figure 6.3: Shadow Technique MSTAR Input Example Images

The MSTAR image set varies greatly, and the three images in Figure 6.3 show only a few of the aspects that make shadow and height extraction difficult, but the 3-D extraction algorithm itself attempts to deal with several of these issues.

6.3 Process Description

6.3.1 Step One: Object Detection

The first step in computing a heights matrix for a given input MSTAR image is to detect all pixels in the image that are part of the object. The purpose is to find the upper border rows of the object region as part of the formula in Figure 6.2.

To extract the object, the input image is first median-filtered in order to remove extreme outlier pixels. The filtered grayscale image is then converted to a binary image using automatic threshold selection. The resulting image is binary, and most of the target pixels are highlighted white, but a large number of other pixels from the background region are still present as well.

The remaining islands that are not part of the target blob are filtered out using a blobbing method. The image is blobbed through the method describe in Section 5.4 and the blobs are sorted in terms of their pixel size. The blob with the largest pixel area (most likely the bulk of the target object itself) is left untouched, and all tiny blobs (only a few pixels in size) are filtered out. Other large blobs are not yet filtered, but are separated to be checked.

These other large blobs are compared one-by-one to the largest blob in the image to determine if the target object might be split into two image blobs. If the centroid of a secondary blob is within a certain distance of the centroid of the largest blob, then it is left in the output image. The distance criterion used is the length of the longer axis of the largest blob, a figure computed easily using the MATLAB `regionprops()` function.

The reason for allowing secondary blobs to remain is that only a single black pixel can cause a separation in a target blob, meaning that a large portion of the target pixels could be left out if only the largest blob was kept. By keeping blobs that are relatively large and are near the largest blob, the software avoids losing vital information.

Just before the image is ready for output, it is median filtered again. This process reduces, to some degree, an edge outlier effect that occurs in fusing multiple MSTAR images to form a 3-D model, as discussed in Chapter 7.

Figure 6.4 shows an example of object detection using this method. The center image shows the results of using automatic-threshold selection for conversion to a binary image

and the image on the right shows the output image of the object detection stage, with two blobs in this case.

Figure 6.4: Shadow Technique Object Detection Example Images

6.3.2 Step Two: Shadow Detection

The key part of the SAR heights matrix creation is the extraction of the shadow from the input image. This portion of the process is very similar to the object detection process, but with some important differences.

To begin with, the only pixels that are of any concern are pixels that are in the same columns as pixels that were highlighted in step one as belonging to the target object. Since the underlying goal of both steps is to compute the height in the third dimension of the target object in each column, and the upper borders of both the shadow and object are required for this computation, if one of these pieces of information is unavailable in a certain column then there is no point in computing the other.

A mask is quickly created that allows masked filtering of all pixels in all the columns in which an object pixel from step one appeared, and the detection of the shadow occurs on all pixels inside this mask region. This process begins by converting the grayscale pixels from the input image to binary pixels, using a slightly different method from that in step one. The thresholding process is inverted and all pixels with a gray level *below* the threshold are set to white in the binary image. Also, automatic threshold selection is not used, but instead a threshold is selected based on the mean of all of the gray levels in the input image. The threshold level used is equal to 45% of this mean, a relatively-optimum value that was determined by experimentation with a varied set of MSTAR images. Any value between 25% and 75% worked relatively well.

After a binary image is created, the same blob filtration method from step one is used to select the pixels in the binary image that belong to the shadow. The shadow itself will be the largest blob of dark pixels, which were highlighted white by the inverted threshold process.

In addition to using the same blob filtration method as step one, this step also ends with median filtering, for the same reason of reducing extreme values on the edge of the fused 3-D model later on.

Figure 6.5 shows examples of resulting images from the major steps in the shadow detection stage. The same input image that was used in Figure 6.4 was used in this example, but the shadow detection stage yielded only a single output blob, rather than two blobs. The second image also shows that during the inverse binary thresholding step, only pixels in the same columns as the object are processed and considered.

Figure 6.5: Shadow Detection Example Images

6.3.3 Step Three: 3-D Heights Slice Creation

The final step is to use the images created in the first two steps and the radar depression angle to create what is called a “slice” in software. The slice is simply a 2-D matrix of the same size as the input images, but instead of each cell (or pixel) in the matrix containing a gray level each cell contains the value of a computed physical target object height (if one was found at that cell location).

Because each of the heights that can be computed using the formula in Figure 6.2 is specific to an entire column, the function that computes a slice operates in this manner. A loop iterates along the column axis of an empty image (all zeroes) and for every pixel location in a column that the object exists, the upper most object pixel and the upper most shadow pixel are found in that column. The column height is computed and that value is placed in that cell location.

The result is a 2-D matrix that, when viewed as an image looks exactly the same as the output from step one, the detected object image. However, the cell values are physical heights, and as such the matrix itself should be viewed as a 3-D histogram.

6.4 Testing and Complications

Twenty-two test sets were created of several different MSTAR vehicles at the various major radar depression angles, including 15° , 17° , 30° and 45° . Each of these sets contained between two and five input images of the same target object at the same depression angle but at various viewing rotations and various image qualities. These sets were used for testing both the shadow technique itself and the 3-D model-forming process described in Chapter 7. The images that make up the sets were selected at random, but were made to ensure that all depression angles and orientations were included. All twenty-two test sets and the output images are available in Appendix C.

When processed individually, the shadow technique successfully computes slice images for every image in each of the sets. The quality of these slice outputs obviously is directly related to the quality of the detection processes for the target object and shadow, and the images that have the poorest quality extraction with the least consistent and noisiest slice images are those in which the input MSTAR image was quite dark. In images like these, the shadow region itself is barely discernable from the background regions and the edges of the shadow become very inconsistent and jagged. An example of this inconsistency is shown in Figure 6.6, in which the target was detected correctly, but the shadow region is of poorer quality.

Figure 6.6: Jagged-Edge Shadow Example Images

Another problem that arises with darker input images is the loss of some section of the shadow. In the case shown in Figure 6.7, the detected shadow region is relatively consistent and has no jagged edges, but its size is severely underestimated, and the upper region of the shadow is lost. This partial detection will lead to incorrect computations of the target object height, since the upper edges of the shadow are used to make each height calculation.

Figure 6.7: Partial Shadow Detection Example Images

An attempt was made to use histogram equalization and adaptive histogram equalization in order to combat both issues that arise from dim input images, but neither technique was refined enough to cause a great performance increase in shadow detection without causing a decrease in the performance of other stages of the algorithm.

6.5 Slice Model Output

MATLAB offers numerous functions designed for the viewing of 2-D matrices as 3-D histograms, including `surf()`, which was used to create the 3-D surface shown in Figure 6.9. The input, detected object, detected shadow images and a small photograph of the actual target object are shown in Figure 6.8. This MSTAR image was taken at a 15° depression angle.

SAR Input	Detected Object	Detected Shadow	2S1 Target Photo

Figure 6.8: Shadow Technique Output Example Image Set

The shape of the slice in Figure 6.9 clearly shows how the heights themselves were computed once per column, and also shows at which rotation in the image the object appeared. The height in each column, therefore, is the maximum height observable on the object at that horizontal point when viewed by radar from below.

The view of the slice image has the same orientation and axes as the 2-D input matrix, and is viewed from the direction of the radar, but at a slightly offset angle.

The view itself appears to follow the relative shape of the physical target vehicle, but this does not occur with all slice images, particularly those in which the shadow detection clearly did not work very well (as described in the previous section). In many cases, though, major features of the target object are discernable.

Figure 6.9: Shadow Technique Output Example Slice Image

7 3-D Target Modeling for the Shadow Technique

A single heights slice image created by the shadow technique contains height information from only one perspective and orientation of the target object. This chapter focuses on what was done in the attempt to fuse multiple slice images of the same target object into a single 3-D model image with the physical heights varying in both the row and column axes (azimuth and range dimensions). Since in a practical or real-time setting, any set of multiple images of the same target that were available would probably be from the same radar platform, the depression angle of the images in the input set was kept the same for this experiment. The same sets used to test the shadow technique in Chapter 6 were used in 3-D model forming as well.

7.1 Rotation and Orientation

In order to form a model from multiple slice images of the same target, some measure of orientation is needed to properly combine the images. For example, given a pixel that is at some physical location on the target object in one image, the same physical location on the target might not be at the same pixel location in another image of the same target.

These two pixels, however, still need to be fused together to determine the target object height at that location, so the differences in target orientation between the two images must be determined.

During the testing of a 3-D model-forming algorithm, two methods were used to account for differing rotations of target objects in the input images. The first method involved automatic extraction of the orientation of the target object in the input images, and the

second method used information available in MSTAR image headers in order to determine the target orientation.

The end goal of both methods was to use the obtained angle of target object rotation and rotate the slice image matrix back in the opposite direction. If this process is performed with each input slice, then all of the slice images will have the same horizontal or vertical orientations and a cell location in one image will have a matching physical location on the target object as the same cell in another. The two slice images could therefore be fused together to form the 3-D model.

7.1.1 Automatic Rotation

The ideal solution would be to be able to extract the rotation of the target object directly from data in the input image, and two methods were attempted to do this. The first and much more successful method was to return to the detected object image and treat the target object as a blob, and then use the MATLAB `regionprops()` function to collect statistics about the blob, including its orientation. The orientation statistic collected by this function is the angle of rotation (with respect to the horizontal axis) of the major axis of the blob, the same axis that was used as the distance criteria for combining multiple blobs into a single detected target object.

One problem with using this method for blob extraction is that it does not account for target objects that are not symmetric. The target shown in Figure 7.1 shows an example of a target shape that is asymmetric, and the ellipse that is used to determine the major axis of the object of the target blob will not be of the desired rotation. A 2-D view of the

slice image *after* its rotation has been adjusted is also shown, and the target object is not parallel with the horizontal axis as is desired.

The result of this problem is that some of the input slice images that will be used to create a 3-D model are rotated slightly with respect to each other, so cell locations do not match; this discrepancy is especially noticeable around the edges of the slices (since targets in separate images might not overlap).

Figure 7.1: Rotation of Asymmetric Target Example Images

Another solution that was tested very briefly was to use edge detection on the detected target object image and pass the edge detected image through the Hough Transform. The orientation of the most prevalent lines found would be the extracted rotation. With input images like that of Figure 7.1, this method worked fairly well, since the edge of the object is near linear, but with many other input images, this method failed completely and was replaced with the axis-based measure of orientation mentioned previously.

An underlying problem with both (and in fact, all) automatic rotation methods is that they cannot account for the direction of the target object. Since all of the targets in MSTAR

are vehicle targets, each target has a front end and a back end, and even if the heights slice can be successfully rotated to be parallel with the horizontal axis, there is no way of knowing if the target vehicle is pointed towards the left of the image or the right. Essentially, even when automatic rotation detection works, a target object in a given slice image might still be rotated 180 degrees with respect to the target objects in other slice images of the input set. The 3-D model created would have completely invalid data.

7.1.2 Embedded MSTAR Rotation

One aspect of MSTAR images is that the raw image format used by MSTAR contains a plain text header with various SAR properties and information. This information includes depression angles, conditions, radar frequency and bandwidth, and most notably in the context of rotation: the orientation of the target object. Since this information is embedded after-the-fact and is not readily available in a real-time application, using the rotation embedded in the header is not ideal, but it does provide for a very good testing environment for 3-D model forming.

An example excerpt from the header of an MSTAR image is shown in Figure 7.2. A lot of information is given in the headers, including information about the physical orientation of the target with respect to the ground plane and the radar platform.

```
...
TargetType= btr60_transport
TargetSerNum= k10yt7532
TargetAz= 67.483337
TargetRoll= 359.943665
TargetPitch= 359.608215
TargetYaw= 6.784937
...
```

Figure 7.2: MSTAR Header Excerpt

The rotation number found in the MSTAR headers (the variable `TargetAz` in particular) was extracted from each of the MSTAR images in the 3-D shadow technique test sets and used to correct the rotation of the target objects. All of the target objects in the heights slice matrices became oriented parallel to the horizontal axis, so that they could be fused together by the 3-D model forming algorithm.

7.2 Fusion Technique and Model Forming

The technique used for taking multiple rotation-adjusted heights slice matrices and combining them to form a single 3-D heights matrix involves comparing the levels (heights) of comparable cells across the set of input images. For each cell location (a given pixel), the lowest value found amidst all of the input slices is the value used at that location in the output matrix because each pixel in the individual slice images contains the level of the largest height across its entire column (before rotation adjustment). This is simply a factor of the shadow technique itself, since all heights are computed per column. An effect of this phenomenon is that many of the pixels in a given column have overestimated heights, and the target object at that location is not as tall as the height level implies. It follows, then, that for any given physical location (cell location) the lowest value found among all input slices is the most valid, since the lowest value will come from the input slice where the height of that location was *least* overestimated, or not overestimated at all.

One stipulation of the lowest-value method for slice fusion is that for each pixel location, the lowest *nonzero* value was chosen, and any zeroes in the input slice images at that location were ignored because of how the slices themselves are created (details are

discussed in Chapter 6). For each slice, only the pixels that were set as white in the detected object image were given height values in the output slice matrix. Since the computation for each height (shown in Figure 6.2) relies only on the upper most detected object pixel, this only affects the pixel size of the individual output slice images is affected. When multiple slices are fused together, however, a slice in which a portion of the detected object was not detected means a good portion of height information not available in a certain region of the target. The remaining input slices can be used to form the model at those pixel locations and the incomplete slice can be ignored.

To be sure about this technique, other point-by-point fusion techniques were tested, including using the mean, median, and mode of the height levels of the input slices. These other metrics, however, tended to decrease the consistency of the output matrices, and created very noisy edge regions.

7.3 Results

7.3.1 Model Output

Figure 7.3 shows standard photographs of the six MSTAR target classes found in the twenty-two test sets. These target vehicles vary in size, material and shape, and produced varying results when tested.

Figures 7.4 through 7.6 show three examples of the fused models created, along with the input MSTAR target images and the individual 2-D views of the slice matrices. Figures 7.4 and 7.5 show models that were created using the target pose rotations found in the MSTAR headers, while Figure 7.6 was created using automatic target rotation extraction.

BRDM_2: 4x4 Amphibious Scout Car	T62: Battle Tank

2S1: Self-propelled Artillery	BTR_60: 8x8 Armored Personnel Carrier

ZSU_23_4: Self-propelled Anti-Aircraft Gun	D7: Armored Bulldozer

Figure 7.3: MSTAR Target Photographs

Figure 7.4: Shadow Technique 3-D Model Output Example #1: 2S1

Figure 7.5: Shadow Technique 3-D Model Output Example #2: BTR_60

Figure 7.6: Shadow Technique 3-D Model Output Example #3: BTR_60

7.3.2 Measurement and Accuracy

The target regions are measured in each test set's output matrix, and these measurements in pixels are then converted to an equivalent physical measurement using the known MSTAR target resolution and pixel spacing (in meters). These measurements are then compared to the true physical measurements of each target vehicle ^[11]. The twenty-two test sets were all taken at varying depression angles and with a varying number of input images, so that comparison to the true physical measurements might be able to indicate the modeling method's sensitivity to these input parameters.

Figure 7.7 shows a large table with all twenty-two test sets, along with the true physical measurements of each target and the physical measurements computed from the model matrices. Input characteristics displayed in the table include: method of rotation extraction, number of input images, target vehicle type, and depression angle.

Because they are calculated from the pixel measurements, the computed physical measurements are quantized, yielding some immediate inaccuracies in the process and measurements. Other inaccuracies are often due to features in the output matrices such as edge noise or outlier pixels.

Figures 7.9 through 7.12 show the results seen amidst all twenty-two sets against the four variables already mentioned. The accuracies are assessed by comparing the computed measurements to the true measurements with four simple metrics: percent difference between the two volumes and percent difference between the two lengths, widths and heights. The number of sets that matched each assessment is also shown.

Figure 7.8 lists the individual sets by number and shows the accuracies with these same metrics for each one. The additional column shown in this table displays the average percent difference of the three physical measurements for each set.

					True Dimensions			Computed Dimensions		
Set	Rotation	Angle	Input Images	Target Vehicle	Length (m)	Width (m)	Height (m)	Length (m)	Width (m)	Height (m)
1	Header	30°	4	2S1	7.62	2.85	2.73	6.40	2.93	3.07
2	Header	45°	3	BRDM_2	5.75	2.35	2.31	5.38	2.33	2.58
3	Auto	15°	4	D7	6.93	3.66	3.35	6.20	3.13	3.78
4	Auto	15°	5	2S1	7.62	2.85	2.73	6.80	2.93	3.19
5	Auto	15°	4	BRDM_2	5.75	2.35	2.31	5.59	2.73	2.62
6	Auto	30°	5	BRDM_2	5.75	2.35	2.31	5.59	2.73	2.10
7	Auto	45°	5	ZSU_23_4	6.54	2.95	2.25	5.38	2.53	2.94
8	Header	45°	6	2S1	7.62	2.85	2.73	5.79	2.73	3.22
9	Header	17°	3	BRDM_2	5.75	2.35	2.31	4.47	2.12	2.91
10	Header	17°	3	2S1	7.62	2.85	2.73	6.80	2.93	2.99
11	Header	15°	3	2S1	7.62	2.85	2.73	6.80	2.33	3.06
12	Header	17°	3	ZSU_23_4	6.54	2.95	2.25	5.69	3.13	2.59
13	Header	15°	3	ZSU_23_4	6.54	2.95	2.25	5.79	3.34	2.69
14	Auto	17°	5	D7	6.93	3.66	3.35	5.99	2.73	3.36
15	Header	17°	3	T62	6.63	3.52	2.40	7.21	2.33	2.68
16	Header	17°	4	T62	6.63	3.52	2.40	6.60	2.73	2.90
17	Auto	15°	3	BTR_60	7.54	2.80	2.32	6.20	2.33	2.86
18	Header	17°	4	BTR_60	7.54	2.80	2.32	7.01	2.93	2.86
19	Header	15°	4	BTR_60	7.54	2.80	2.32	6.40	2.53	2.79
20	Auto	15°	5	T62	6.63	3.52	2.40	7.21	2.93	2.98
21	Header	17°	3	D7	6.93	3.66	3.35	5.79	2.53	3.29
22	Auto	30°	4	ZSU_23_4	6.54	2.95	2.25	4.77	2.53	3.37

Figure 7.7: Shadow Technique 3-D Model Test Set Measurement Results

Set Number	Volume Percent Difference	Length Percent Difference	Width Percent Difference	Height Percent Difference	Average Percent Difference
1	2.899	16.010	2.807	12.454	10.424
2	3.612	6.435	0.851	11.688	6.325
3	13.669	10.534	14.481	12.836	12.617
4	7.202	10.761	2.807	16.850	10.139
5	28.094	2.783	16.170	13.420	10.791
6	2.671	2.783	16.170	9.091	9.348
7	7.813	17.737	14.237	30.667	20.880
8	14.151	24.016	4.211	17.949	15.392
9	11.654	22.261	9.787	25.974	19.341
10	0.481	10.761	2.807	9.524	7.697
11	18.224	10.761	18.246	12.088	13.698
12	6.261	12.997	6.102	15.111	11.403
13	19.838	11.468	13.220	19.556	14.748
14	35.335	13.564	25.410	0.299	13.091
15	19.618	8.748	33.807	11.667	18.074
16	6.710	0.452	22.443	20.833	14.576
17	15.648	17.772	16.786	23.276	19.278
18	19.932	7.029	4.643	23.276	11.649
19	7.767	15.119	9.643	20.259	15.007
20	12.396	8.748	16.761	24.167	16.559
21	43.280	16.450	30.874	1.791	16.372
22	6.311	27.064	14.237	49.778	30.360

Figure 7.8: Accuracy of Shadow Technique 3-D Model Test Sets

Rotation Extraction Method	Number of Sets	Average Volume Percent Difference	Average Length Percent Difference	Average Width Percent Difference	Average Height Percent Difference
Header	13	13.417	12.501	12.265	15.551
Auto	9	14.349	12.416	15.229	20.042

Figure 7.9: Accuracy of Shadow Technique 3-D Model Test Sets By Rotation Extraction Method

Depression Angle	Number of Sets	Average Volume Percent Difference	Average Length Percent Difference	Average Width Percent Difference	Average Height Percent Difference
15°	8	15.355	10.993	13.514	17.806
17°	8	17.909	11.533	16.984	13.559
30°	3	3.960	15.286	11.072	23.774
45°	3	8.526	16.063	6.433	20.101

Figure 7.10: Accuracy of Shadow Technique 3-D Model Test Sets By Depression Angle

Number of Input Images	Number of Sets	Average Volume Percent Difference	Average Length Percent Difference	Average Width Percent Difference	Average Height Percent Difference
3	9	15.402	13.073	14.720	14.519
4	7	12.197	11.285	12.061	21.837
5-6	6	13.261	12.935	13.266	16.504

Figure 7.11: Accuracy of Shadow Technique 3-D Model Test Sets By Number of Input Images

Target Vehicle	Number of Sets	Average Volume Percent Difference	Average Length Percent Difference	Average Width Percent Difference	Average Height Percent Difference
2S1	5	8.592	14.462	6.175	13.773
BRDM_2	4	11.508	8.565	10.745	15.043
BTR_60	3	14.449	13.307	10.357	22.270
D7	3	30.761	13.516	23.588	4.975
T62	3	12.908	5.983	24.337	18.889
ZSU_23_4	4	10.056	17.317	11.949	28.778

Figure 7.12: Accuracy of Shadow Technique 3-D Model Test Sets By Target Vehicle

It can be seen from the results in Figures 7.7 through 7.12 that some of the input parameters have a greater effect on the quality of the output model than others. Some of the results indicated that certain criteria lead to a greater amount of accuracy, but usually only with slight improvement.

Figure 7.9 indicates what was already expected, that the rotation extraction method which used the rotation data in MSTAR image headers was more accurate in model creation than the automatic method of extraction. The differences in accuracy, however, were still mostly minimal, and with one metric the automatic method actually performed very slightly better.

The data in Figure 7.10 were more extreme, with the results at the 30° and 45° depression angles showing a large amount of inconsistency between the different metrics. Not as much data was available at these two angles compared to the lower angles, and consequently there were fewer tests performed, which could explain the greater amount

of inconsistency in the results. Nevertheless, it seems clear from the data in Figure 7.10 that the lower depression angles performed better on average. The models from the 15° depression angle outperformed those of the 17° depression angle in all but one metric, indicating that lower depression angles in general will yield better results.

Figure 7.11 shows that using a greater number of input images do not necessarily lead to a greater amount of accuracy, as the results with 4 input images performed best in all but one metric. The differences between all three (3, 4 or 5 to 6) were marginal in most categories, but the results did seem to show that using only 3 input images was not quite enough, and then 4 or more inputs yielded the best results.

From Figure 7.12 it can be observed that the shape and type of the target vehicle being modeled did make a significant difference in the average quality of the models. The target vehicles 2S1 and BRDM_2 were modeled much more accurately on average than the other targets. Although it performed much better than the others with one metric, the D7 target vehicle performed significantly worse than all others overall. Figure 7.3 shows that this target vehicle is perhaps the one with the most complex shape, signifying it would be more difficult to accurately model.

The overall accuracies for all of the test models were 12.466%, 13.477% and 17.389% in terms of length, width and height, and 13.798% in terms of volume. These results indicate that the method is satisfactory in modeling and extracting the measurements of an arbitrary radar target with varying input parameters.

7.3.3 Practical Limits to Comparison with Spectral Estimation

Unfortunately, no useful comparison can be made between the models created with the shadow technique and those of the decoupled least-squares method for spectral estimation. As described in Section 4.2.1, the maximum altitude that was resolvable with the decoupled least-squares technique was too low for MSTAR inputs to incorporate the entire volume of the target. As such, the least-squares models created with MSTAR were not nearly tall enough to be compared to the models of the shadow technique, which has no practical height limit.

8 Additional Study with Non-SAR Modeling

A short side study was conducted related to the prospect of 3-D SAR imaging, involving the use of standard imagery to model 2-D objects in a 3-D space. This chapter discusses the details and the results of the study.

In experimenting with non-SAR grayscale images of a simple geometric shape (a box), the test set of images that was created was of a higher quality than available SAR images would be (from MSTAR sets). The box shapes were clearly visible to the human eye in the test images, and a short algorithm was written to automatically extract and characterize the shape in software.

8.1 Process Description

The program written consisted of a 10-step algorithm that used several of the tools mentioned in Chapter 5. It relies on finding prevalent lines in the image and using the information from these lines to find and categorize the vertices of the object. These vertices are then ordered and used to redefine (or redraw) the object mathematically, first in a 2-D space, and then in a 3-D space. These concepts could be later expanded to incorporate other basic figures, perhaps beginning with an expansion of the Hough Transform to seek out shapes other than lines.

The 10 steps of the algorithm for extracting a 3-D description of the box-shaped object from a 2-D grayscale image are outlined in the following subsections. It should be noted that the algorithm incorporates two passes of edge detection followed by the Hough

Transform. The first collects the outer border edges of the figure, and the second collects the edges that appear inside the object.

8.1.1 Step One: First-Pass Edge Detection

The first pass of Canny edge detection uses relaxed threshold parameters and a slightly enlarged standard deviation for the Gaussian filter (extra smoothing). This step is to highlight the edges in the image that correspond to where the background ends and the box figure itself begins.

To account for extra edges that are extracted from parts of the background (not on the edge of the object), the edge pixels are filtered through blobbing. The pixels are all blobbed and only the pixels that are members of the *largest* blob are left in the edge output image because all of the edges on the border of the object will be connected to one another at the vertices, but other edges that were detected are simply due to noise.

8.1.2 Step Two: First-Pass Hough Transform

The edge detected binary image output from step one is used as the input to the first pass of the Hough Transform, and the endpoints of the prevalent lines in the image are collected in an array. Because the edges previously found are linear and represent the border of the object, the endpoints returned by the transform (and MATLAB helper routines) also represent the pixel locations of the outer vertices of the object, with some noise. These collected points are used in later steps to describe the object mathematically.

Figure 8.1 is an example image showing the line segments returned by the Hough transform overlaid onto the binary edge detected image. The red pixels in the image show the Hough lines while the white pixels show edge pixels. This image shows that some pixels that were not true edge pixels were detected in step one. Some of these pixels did not end up in a Hough line, but others did. In addition, some white edge pixels did not yield a line output from Hough, meaning fewer lines were extracted than were actually present in the image.

While not all of the lines are actually necessary to collect all of the vertices (as the endpoints of the last edge will both be repeated vertices), both of these discrepancies create a small amount of error in the collected endpoint locations. To some degree these errors are accounted for in later steps of the algorithm.

Figure 8.1: Non-SAR Algorithm First-Pass Hough Transform Example Image

8.1.3 Step Three: Border Mask Creation

The next step in the algorithm requires finding a mask region (as discussed in Section 5.6) that highlights the pixels in the input image that comprise the object. The first pass of the Hough Transform returned the lines and endpoints that describe the borders and these endpoints are used to forge a binary mask.

As mentioned in Chapter 5, a quick and simple way to create a binary mask given a set of endpoints is to use MATLAB's `roipoly()`, and step three in the process was accomplished using this function. Its use is preceded by a short routine that takes the endpoints returned by Hough and places them in a circular order, ensuring that the mask created consists of one continuous region.

The routine simply selects one of the given endpoints from Hough at random to begin the output array and calculates the physical distance to all the other available endpoints. The endpoint nearest to the selected endpoint is placed next in the output array, and the process repeats using the second point to compute the distances to all the other points. As each point is placed in the output array, it is discarded from the set of available endpoints.

This short algorithm relies on the fact that adjacent endpoints on the outer border of the image object are likely to be much nearer together than endpoints on opposite sides of the object. If the set of endpoints returned by the Hough Transform is relatively accurate to the object, then the endpoints are collected in the correct order.

Because it involves nested for-loops, this algorithm is not especially efficient, but this limitation does not significantly affect the speed of the process because the set of endpoints is always only a handful.

8.1.4 Step Four: Second-Pass Edge Detection

The underlying purpose of extracting a mask of the object region in step three is to allow a second pass at edge detection in which the Canny edge detector parameters can be different from the first pass and more suited for detection of the edges inside the object region. The edges inside the object are not edges between pixels of different materials (as were the edges between the object and the background). These inner edges are more subtle because they are due to a change in the contours and angles of a single type of material. As such, the parameters of the Canny edge detector must be made less strict to account for more subtle changes in gray levels.

The original input image is mask-filtered using the object mask created in step three and the filtering function used is a call to Canny edge detection. The returned resulting image contains the inner edges of the image object, but also may contain stray edge pixels at the edge of the mask. Because the mask used is placed directly on top of the outer border of the object, and some pixels that represent the outer borders may still be underneath the mask. To compensate, the mask is eroded slightly (a thin layer of pixels are removed from the edges) to ensure that no outer borders show up in the second pass of edge detection.

8.1.5 Step Five: Second Pass Hough Transform

The edge-detected image of the inner portion of the object created in step four is passed through the Hough Transform to extract the lines and endpoints of edges on the inner view of the object. Just as before, some of the lines can be missed, but later steps in the algorithm account for this problem. In this step, at most three lines can be on the inner view of any object and from any viewpoint. This stipulation leads to only three cases of returns from the Hough Transform: one, two, or three lines. These three cases are handled individually later on.

8.1.6 Step Six: Edge Image Creation

Step six is not a true computation in the scope of extracting a 3-D representation of the object, because it is only for display and convenience purposes, but it is included here and in the algorithm anyway. This step is simply to perform a logical OR operation on all of the pixels in the edge detected images from steps one and four (the outer border edges and the inner object edges). This process creates a binary output image that contains all of the edges that describe the view of the object, making it easy to see what information is available and what is not.

8.1.7 Step Seven: Finding an Inner Vertex

Steps seven and eight are implemented as a single function in the application but are described as two relatively independent steps. The goal of the two steps is to obtain a complete array of row and column locations that correspond to all of the visible vertices in the image object. Step seven in particular, finds the one vertex that is logically distinct

from all the others: the inner vertex in the object region. The reason that steps seven and eight were implemented as a single function is that a single situation exists in which the returns from step seven rely partially on the returns from step eight, as described later.

The inner vertex is almost always completely surrounded in the image by pixels of the object itself (rather than background pixels), and almost always contains three visible edge lines protruding from it, one for each dimension. This inner vertex, then, is a good candidate for a base point for conversion into a 3-D space later on.

The first step in extracting the inner vertex location is to take the set of inner lines returned from the second pass of the Hough Transform and filter out lines that are extremely close together in terms of orientation angle using the Hough theta parameter. This process effectively weeds out any lines that are repeated versions of already-detected lines but with slightly different orientations (an occasional side effect of the Hough Transform). In most circumstances, the three possibly visible lines connected to the inner vertex would have very different angles of orientation in the image, usually somewhere between 60 and 120 degrees apart. In the few circumstances where two separate lines of separate dimensions in the image were extracted by Hough at orientations of only a few degrees apart, the information about their intersection would probably be much less useful than the information about their intersection with the other dimension, so filtering out one would not be too detrimental.

After the Hough lines are filtered, the algorithm branches based on the number of lines that were left in the set, which is kept to three or fewer. The three different situations are described as follows:

1) One inner line in the set:

There is not enough information present from a single line and only two endpoints, so both endpoints are added to the output array for the time being until the set of border vertices is collected in step eight. This situation is the only one in which a dependency exists between steps seven and eight—and the reason they are implemented as a common function.

2) Two inner lines in the set:

Both lines are described using their slope-intercept formula computed from their endpoints, and the intersection of the two is computed and placed in the output array.

3) Three inner lines in the set:

In this situation “extra” information is available from which to compute the most likely location of the inner vertex. The slope-intercept form of all three lines is computed from their endpoints and the three intersections of those lines are calculated as well. If two of those intersections are exactly the same, then it is a likely candidate for the inner vertex and it is used as the output object. If all three points are different, then their row and column locations are averaged to create the output array.

In situation 1, after step eight has completed and a good set of border vertices is available, the two inner vertices that remain from step seven are compared to all of the border vertices. The inner vertex point that has the closest distance to any of the border vertices is *not* selected, because it is the point that is more likely to be a repeat version of a border vertex. The other point is selected for output as the inner vertex.

The formulas used to compute the intersection of any two lines are shown in Figure 8.2.

$$\begin{aligned}y &= m_1x + b_1 \text{ and } y = m_2x + b_2 \\x &= \frac{b_2 - b_1}{m_2 - m_1} \\y &= m_1x + b_1\end{aligned}$$

Figure 8.2: Non-SAR Algorithm Arithmetic Solution of Two Lines

8.1.8 Step Eight: Border Vertex Filtering

To find the vertices that are on the outside border of the image object, all of the distances between all of the endpoints returned by the first pass of the Hough Transform are computed. Then the minimum distance is found and if the two points represented by that distance are members of different line segments the arithmetic solution for an intersection between those lines is found in the same manner as in step seven. The endpoints that were of a minimum distance from each other are then discarded and the process begins again. This process continues for the same number of iterations as there were available line segments from Hough. If there were six lines, there would be six iterations to find all six border vertices.

To make sure two lines that were not adjacent to each other did not create a border vertex with their intersection, an iteration of the loop is ignored if the minimum distance found was greater than 15% of the maximum distance between all points in the set. This threshold is relatively arbitrary, but since the distances between endpoints of adjacent lines are usually very small (less than 5 pixels) and the maximum distance is usually quite large, the threshold works quite well.

After the loop has completed all points that were not discarded are added to the output set of border vertices to account for all situations where fewer than six outer border lines were found by Hough. Effectively the endpoints of the lines that have only one adjacent neighbor line are added to the output set.

Figure 8.3 shows an example image where the inner vertex is highlighted by a green X and the outer border vertices are each highlighted by a blue X. The lower images in the figure highlight a couple of the factors that step seven and eight successfully account for. The two steps successfully found a good representation of an inner vertex with only two inner lines and successfully calculated a border vertex based on information from the mathematical description of the outer line segments rather than just the intersections of the pixels of the lines themselves.

Figure 8.3: Non-SAR Algorithm Vertex Finding Example Images

8.1.9 Step Nine: Object Recreation in 2-D Space

At this point in the algorithm, the only representation of the image object available is in the form of a set of vertices. In order to fully represent the viewpoint the image has of the object in a 2-D space, a set of lines is needed to describe every visible edge in the image. The Hough Transform was helpful in finding sets of pixels that indicated a line, but lines could easily be missed, blended together, or incomplete in its returns, so a new step is needed to forge a better set of object lines using the set of extracted visible vertices.

Step nine approaches this problem by first determining the set of border lines using trigonometry. Since from a software perspective, it is unknown which vertices are connected to each other through lines, some assumptions must be made in order to determine the connections.

Observation of the object reveals that for every vertex on the outer border of the object, the two other outer vertices to which it is connected are the two points that are at the most extreme angles of orientation with respect to it. For example, in the example figure shown in Figure 8.4, the software would draw a reference line DG between an outer vertex and the inner vertex. It would then begin collecting a set of all of the angles between that line and the lines drawn between point D and all other outer vertices (AD, BD, CD, DE, and DF). As can be seen, the two vertices to which point D is actually connected through an edge in the image are points C and E. The angle between lines DG (the reference line) and CD will be larger (more positive) than all other angles in the collected set, and the angle between lines DG and DE will be smaller (more negative)

than all other angles in the collected set. It is these two lines, therefore, that are connected to point D and are added to the output set of 2-D lines.

Figure 8.4: Non-SAR Algorithm Outer Vertex Angles Example Image

When these steps are completed for each border vertex, the output set should have six lines representing all the lines between the outer border vertices. The last three lines to be collected are the inner lines going through a point like point G in Figure 8.4. An inner vertex will have three object edges protruding from it, and these lines (AG, CG and GE in the example) will be connected to every other outer vertex. In other words, there are only two possible sets of inner vertices: $\{AG, CG, GE\}$ or $\{BG, DG, FG\}$. The algorithm completes step nine by considering both of these sets and using the fact that each of the three inner lines should be relatively parallel to two of the border lines.

For each of the two sets of possible inner lines, the square of the error is computed for the angle of each line with respect to the outer lines it should be parallel to. For example, in Figure 8.4, the square of the difference in orientation angles between lines GE and CD is computed and placed into an array. Then the square of the difference in orientation angles between GE and AF is computed and placed into the same array. The same process is performed for the two outer lines which should be parallel to line AG, and so on. Two arrays are created, one for each set of possible inner lines, and the set selected for output is the set that has the smallest average value in the array, or the smallest mean-squared error. In this case, the set with lines AG, CG and GE would have a much smaller mean-squared error than the other set, so it is selected.

8.1.10 Step Ten: Object Recreation in 3-D Space

The final step in the algorithm takes the set of known 2-D line segments described by their endpoints and attempts to convert these lines to lines in 3-D space with (x,y,z) endpoints. The z direction is taken to be positive looking into the image with the inner vertex of the object used as an orientation point, set at $(x,y,0)$, where x and y were the column and row of the inner vertex in 2-D space. It is assumed that the input image itself is a given perspective of the 3-D space matrix, so the x and y values of each vertex in the 2-D line set should remain the same in 3-D space, with only the z parameter being unknown going into step ten. It is also assumed that the box is right-angled, meaning that the three lines connected to the inner vertex are all perpendicular to one another.

Based on the assumptions, an arithmetic solution can be obtained for all eight 3-D vertices of the object (even the invisible ones) using the known constants. This process

begins with a solution for the three endpoints for the three lines that are connected to the inner vertex. Assuming that the inner vertex is at the front of the 3-D image matrix ($z = 0$), the depths relative to this point can be found for the other endpoints to each of the three lines it is connected to. Three constants are computed first, followed by the calculations for the relative depths for each of these three other points.

The arithmetic showing these formulas and the solution to obtain them is shown in Figure 8.5. It begins by stating that since all three lines in question are perpendicular, the dot products between them will equal 0. All line labels follow the example from Figure 8.4.

$$\begin{aligned}
 &AG \perp CG \perp GE \Rightarrow AG \bullet CG = AG \bullet GE = CG \bullet GE = 0 \\
 \\
 &\text{Define :} \\
 &C_{12} = \Delta x_{AG} \cdot \Delta x_{CG} + \Delta y_{AG} \cdot \Delta y_{CG} \\
 &C_{13} = \Delta x_{AG} \cdot \Delta x_{GE} + \Delta y_{AG} \cdot \Delta y_{GE} \\
 &C_{23} = \Delta x_{CG} \cdot \Delta x_{GE} + \Delta y_{CG} \cdot \Delta y_{GE} \\
 \\
 &AG \bullet CG = C_{12} + \Delta z_{AG} \cdot \Delta z_{CG} = C_{12} + z_A \cdot z_C = 0 \\
 &AG \bullet GE = C_{13} + \Delta z_{AG} \cdot \Delta z_{GE} = C_{13} + z_A \cdot z_E = 0 \\
 &CG \bullet GE = C_{23} + \Delta z_{CG} \cdot \Delta z_{GE} = C_{23} + z_C \cdot z_E = 0 \\
 \\
 &z_A = \sqrt{\frac{C_{12} \cdot C_{13}}{C_{23}}} \text{ and } z_E = \frac{C_{13}}{z_A} \text{ and } z_C = \frac{C_{23}}{z_E}
 \end{aligned}$$

Figure 8.5: Non-SAR Algorithm 3-D Space Transformation Arithmetic

The three constants are referred to as C_{12} , C_{13} and C_{23} , and the outputs z_1 , z_2 and z_3 are the z parameters for the endpoints of the lines connected to the inner vertex (in Figure 8.4 they would be points A, C, and E). The x and y parameters for these three points remain unchanged from the 2-D model from step nine.

It should be noted that the formulas shown in Figure 8.5 for the three z parameters are used exactly as shown. The values for some of the constants should be negated, but this negation was ignored in order to make the resulting z values all positive and make matrix indexing much easier in software. The z values in the matrices used in software are also shifted by +1 to account for the fact that MATLAB begins matrix and array indexing at 1 rather than 0.

The information and dimensional measures obtained from the first three lines are used to create the rest of the 3-D line output set. Since one line was collected for each dimension around a common point (the inner vertex), the collection of the remaining 3-D endpoints and lines is merely a process of summing vectors together. For example, in the example of Figure 8.4, the 3-D vector described by line CG is added to point A to obtain the 3-D location of point B, and then 3-D vector EG is added to point B to obtain the missing (hidden) vertex, and so on until all eight vertices and all twelve 3-D lines are described.

Since both visual output and a 3-D image matrix are desired, Bresenham's Line Algorithm is used in a 3-D context to highlight all of the pixels in the 3-D output matrix that are on one of the lines that describe the object. The algorithm is commonly used to draw lines in images because it simply traverses along image columns and determines which pixel centers in each column are nearest to the true mathematical description of the line.

Plotting is done with a 3-D view of the lines of the recreated box object, and the view is continually rotated 360 degrees around the object (at approximately 25 frames per second) so that all viewpoints are obtained. In addition, the three primary perspectives—

front, top, and side—are all shown below the 3-D view. An example output screenshot is shown in Figure 8.6. Some vertices are highlighted with colored Xs to show their location throughout all four images.

Figure 8.6: Non-SAR Algorithm Example Output Image

8.2 Results

In all, eight grayscale input images were created of the same box-shaped object. The input images varied in perspective, angle, lighting, distance from the object, and visibility of the dimensions of the object. The underlying goal was to be able to account for all factors and variations of the input and be able to extract the dimensions and orientation of the object given any reasonable input image. The input test images and all examples of outputs from various stages of the non-SAR algorithm are available in Appendix F.

With the algorithm described in the previous section only six of the eight input images were processed correctly. The algorithm failed when given a purely front-side perspective, where no inner vertex was visible on the object.

The application in its current form is quite sensitive to certain types of noise, particularly because of the edge detection processes near the beginning. Edge detection is not an exact science, and as such is not particularly robust or flexible for a wide range of input types. This limitation was the reason for using two separate stages of edge detection and Hough Transform, each stage suited for a specific task.

The non-SAR 3-D extraction algorithm served as a decent model for expansion into 3-D extraction from SAR imagery. The SAR images available for testing (MSTAR) differ in numerous aspects, including the presence of radar shadows that were used in the SAR 3-D algorithm. The SAR images are also of a lower resolution and contain a greater amount of noise.

9 Conclusions and Future Work

9.1 High-Resolution Spectral Estimation

Two methods for high-resolution spectral estimation were presented in Chapter 3, and a number of simulations were executed in Chapter 4 to test the accuracy, usability, and sensitivity of one of these methods. The decoupled least-squares technique for finding the Maximum Likelihood estimates of target reflector locations and amplitudes proved reliable enough to resolve targets in the range-altitude dimension, but required a sizeable amount of input data with stringent data collection requirements.

9.1.1 Technique Comparison

While several parameters and aspects of the technique were investigated in Chapter 4, there are still some which deserve additional attention. To begin with, a more in-depth comparison should be made between the decoupled least-squares technique for spectral estimation and the RELAX algorithm. While both methods operate on similar principles and process data similarly, they do use very different methods to account for individual scatterer signatures in the range-cross-range dimensions, and further study should be done to compare the accuracy of results from both techniques, especially with respect to the parameters explored in Chapter 4.

This comparison would be particularly important with respect to the MSTAR simulations performed. The decoupled least-squares technique proved very sensitive to the amount of input data and the spread of the depression angles between the minimum and maximum angle. While the RELAX algorithm uses a similar method of data collection

and interpolation and would require a similar amount, it could prove to be less sensitive to the spread of depression angles. If this were the case, it could make the RELAX algorithm a more practical solution in a real-world application where data collection at extremely precise depression angles may not be possible ^{[3] [8] [9]}.

Unfortunately, all mentioned methods for high-resolution spectral estimation require the collection of target SAR data to be at the same viewpoint of the target, which may be a complicated limiting factor in a practical application.

9.1.2 Dynamic Reflector Quantity Estimation

The decoupled least-squares technique for estimating scatterer locations and amplitudes differed from the RELAX algorithm in one other way: it required a static and predetermined estimate for K , the number of scatterers in z for a typical range-cross-range location. One method for dynamically estimating K , based on a secondary minimization of the mean-squared error of current reflector estimates, was briefly studied and simulated in Chapter 4. This technique proved to be more resistant to inaccuracies when overestimating K , but expectedly did not yield as accurate of results as when K was statically estimated correctly.

Other methods for dynamically estimating K could be researched, including methods based on those of the RELAX and CLEAN algorithms. In those techniques, the number of reflectors in z is effectively estimated by a set of arbitrary noise floor thresholds. Due to the nature of the decoupled least-squares estimation procedure this cannot be directly applied to that method, but the same principles could be applied in other ways to

individually account for reflector energy, and the estimate for K could be increased until the output energy passes some threshold.

Additionally, the current method being used to dynamically estimate K could be reworked to limit the range of K which is allowable. Currently, the primary reason for inaccuracy is the number of false reflectors which are plotted due to an overestimation of K . If the number of grid points at which K is overestimated is reduced by using a lower maximum for K , then the accuracy might improve considerably. In the case of the simulations of Chapter 4, if K were limited to be between 2 and 3 instead of 2 and 4, better results may occur.

9.2 Shadow Technique

A working algorithm for 3-D feature extraction that uses information innate to the SAR platform (the depression angle) and information discernable from individual and arbitrary SAR images (the shadow) was created. While it was a decent proof-of-concept, the obvious applications for individual heights slices created using the shadow technique are thus far limited. One possible application, model creation, was explored in this thesis with some results, but other applications or expansions and improvements could easily be researched.

The shadow technique itself relies heavily on two factors: the presence of a radar shadow in the input image and the ability of the algorithm to discern the shadow region. The vast majority of tested images that were processed had a successful shadow extraction, and the

images that yielded partial or complete failures were visibly predictable (usually very dark images).

Two or more slice images can be used with the current fusion algorithm for creating a rough 3-D model of an input target. The validity or accuracy of the model is directly related to the accuracy of the shadow and object extraction steps in each of the input slices. The model creation technique itself also is dependent upon a method for determining the rotation of the target object in each of the input slices, and while two methods for this step were explored, only one is useable in a practical or real-time application: automatic extraction based on the image. More accurate models, however, are created using the rotations embedded into the MSTAR file headers, and these models are useful for testing and analysis.

The 3-D target model creation process utilizes no inputs other than the SAR images and the depression angle, meaning it can be used for any arbitrary SAR input or scenario, not just the eight target classes available with MSTAR. It only requires the visibility of a SAR target and a SAR shadow.

This is a distinct set of advantages when compared to the methods presented in Chapter 3 for high-resolution spectral estimation. Those methods required a considerable amount of additional data at very precise depression angles and target poses, while the shadow technique was far less strict.

9.2.1 Incorporation with ATR Algorithms

Since the majority of prior SAR research at Cal Poly was aimed at accurately identifying target vehicles in arbitrary MSTAR images, incorporation of any new techniques or data into the current Automatic Target Recognition algorithms is worth studying.

The data extracted by the shadow technique could be used as an additional feature input to the current ATR algorithms, or the output of the ATR algorithms could be used as inputs to the target modeling process. The current model-forming algorithm is for any arbitrary SAR target input, but the models could be refined to use built-in constraints for model heights and shapes if a good estimate of the target type was given as an input from some ATR algorithm. This would make the technique less robust and only capable of dealing with the eight MSTAR classes, but would probably yield a greater accuracy in the models and reduce errors in the process since additional information is available about the target.

9.2.2 Process Improvements

One variable of the model-forming process that was not explored was the use of multiple depression angles in forming a single 3-D model, similar to the spectral estimation techniques. The test sets created for the shadow technique covered all depression angles between 15° and 45° , but each set had input images from only one angle. This approach was taken to simulate a situation in which a SAR platform would be monitoring a target at a single elevation and ground distance. If multiple passes could be made and the target could be illuminated for the creation of SAR images at different depression angles—or if a second SAR radar could create images from the perspective of a separate platform at a

different time—then input image slices created from multiple depression angles could be practically obtained for the purpose of creating a 3-D model.

Furthermore, another possible improvement to the shadow technique for 3-D target model creation is the prospect of shadow-based target rotation extraction. In a given MSTAR image, since the location of the radar is always below the image, the shape of the detected shadow region itself carries some information as to the rotation of the target object. If a “training set” of pre-defined shadow shapes (dependent on target type and depression angle) were collected, then the shape and size of the shadow could lead to an accurate calculation of the rotation and orientation of the target object. This fact was briefly mentioned in a previous Cal Poly paper by Dr. Saghri and Andrea DeKeleita ^[15].

A third portion of the shadow technique process which could use some additional study is in the process of data fusion for creating models. Since the method of fusing multiple slice images is currently somewhat ad-hoc, certain improvements could be examined. To eliminate edge noise, filtering of the output slice could be attempted, or the method of fusion itself could be altered. One such adjustment could be to require that for a given cell location, two or more slice images must have nonzero heights. Currently, the lowest nonzero height is selected for a given location in the output matrix, but occasionally there was data at a certain cell in only one input image out of three or more. Requiring that two images have data at each location might mean less noise and inconsistency around the edges of the target model.

9.2.3 Slice Quality and Failure Detection

No viable measure of the quality of individual slices was developed, and no method for the detection of failures was researched. A failure can be detected in many simple ways, beginning with a measure of the quality of detected shadows, but none was immediately needed for any of the tests performed in this project. A good method for detecting a failure would be useful in determining the certainty of an ATR algorithm that might use shadow technique features, or determining if individual slice images should be ignored in the creation of a 3-D model from several slices.

9.3 Additional Study with Non-SAR 3-D Modeling

Although the images were not created from radar data, and the amount of noise present is minimal compared to the noise in a SAR image, the non-SAR algorithm demonstrates what could be done with higher-quality grayscale images. The direct extraction of 3-D shape, size, and orientation would be extremely useful in any target-recognition or target modeling application.

9.4 Summary and Final Thoughts

This thesis project explored several existing methods for the creation of 3-D target functions, data, and models from Synthetic Aperture Radar data. Three-dimensional SAR data could have countless applications for reconnaissance, target recognition, target modeling, terrain mapping, and even target tracking.

Some simulations were executed to investigate the feasibility, accuracy, sensitivity, and practicality of one of the better existing methods for 3-D SAR. The results showed promising results with regards to 3-D target modeling, but also highlighted the limiting factors that prevent these techniques from being practically utilized in a real-world target recognition or target modeling application.

In an attempt to find a solution to the existing obstacles, this thesis project also experimented with the extraction and use of 3-D features from a single SAR image or multiple SAR images. The proposed shadow technique employed visible and quantifiable image features coupled with information from basic geometric characteristics of the SAR data collection process.

Future studies on 3-D SAR will undoubtedly include research towards finding direct methods to overcome the practical limitations of high-resolution spectral estimation, but may also include additional examinations into observable and measurable image features from existing 2-D SAR imagery.

Bibliography

- [1] Blackwell, Dustin; Hupton, John; Hursig, Robert; Kiefer, Jessica; Schlutz, Matt; Seims, Scott. "BalloonSat 2008." 2008.
- [2] Bryant, Michael L.; Gostin, Lamar L.; Soumekh, Mehrdad. 3-D E-CSAR Imaging of a T-72 Tank and Synthesis of its SAR Reconstructions. Pages 211- 227. IEEE Transactions on Aerospace and Electronic Systems, January 2003.
- [3] Castelloe, Michael W. and Munson, David C., Jr. 3-D SAR Imaging via High-Resolution Spectral Estimation Methods: Experiments with XPATCH. San Diego, CA: Hughes Network Systems; Urbana, IL: Coordinated Science Laboratory, University of Illinois, 1997.
- [4] Cumming, Ian G and Frank H Wong. Digital Processing of Synthetic Aperture Radar Data. Norwood, MA: Artech House, Inc., 2005.
- [5] Curlander, J. and R. McDonough. Synthetic Aperture Radar: Systems and Signal Processing. New York, NY: John Wiley & Sons, 1991.
- [6] Currie, A.; Baker, C.J.; Bullock, R.; Edwards, R.; Griffiths, H.D. High Resolution 3-D Radar Imaging. London, U.K.: Electrical and Electronic Engineering Dept., University College, 1995.
- [7] Guilas, Chessa F. "Hausdorff Probabilistic Feature Analysis in SAR Image Recognition." 2006.

- [8] Kuklinski, Walter S. and Andrea L. Kraay. 3D SAR Imaging Using a Hybrid Decomposition Super-Resolution Technique. Bedford, MA: The MITRE Corporation, 2000.
- [9] Li, Jian. Efficient Mixed-Spectrum Estimation with Applications to Target Feature Extraction. Pages 281 – 295. IEEE Transactions on Signal Processing, February 1996.
- [10] Mason, Paul Ryan. "MATLAB Simulation of Two-Dimensional SAR Imaging By Range Doppler Algorithm." 2007.
- [11] MilitaryFactory.com. Military Tanks, Vehicles, and Artillery. 2003.
<<http://www.militaryfactory.com/armor/>>
- [12] Mossing, J. and T. Ross. "An Evaluation of SAR ATR Algorithm Performance Sensitivity to MSTAR Extended Operating Conditions." 1998.
- [13] "Caterpillar D7 Bulldozer." 2008. Olive-Drab.com Military Information HQ.
<http://www.olive-drab.com/idphoto/id_photos_d7_dozer.php>
- [14] Pasquali, P.; Prati, C.; Rocca, F.; Seymour, M.; Fortuny, J.; Ohlmer, E.; Sieber, A.J. A 3-D SAR Experiment with EMSL Data. Vancouver, Canada: University of British Columbia, 1995.
- [15] Saghri, John and Andrew DeKeleita. "Exploitation of Target Shadows in Synthetic Aperture Radar Imagery for Automatic Target Recognition." San Diego, CA, 2006.

- [16] Saghri, John and Dan A. Cary. "A Rectangular-Fit Classifier for Synthetic Aperture Radar Automatic Target Recognition." San Diego, CA, 2007.
- [17] Sandia Corporation. Synthetic Aperture Radar. 2005.
<<http://www.sandia.gov/RADAR/sar.html>>.
- [18] Sansosti, Eugenio. A Simple and Exact Solution for the Interferometric and Stereo SAR Geolocation Problem. Pages 1625 – 1624. IEEE Transactions on Geoscience and Remote Sensing, August 2004.
- [19] Soumekh, Mehrdad. Synthetic Aperture Radar Signal Processing. New York, NY: John Wiley & Sons, Inc., 1999.
- [20] "Canny Edge Detector." 15 April 2008. Wikipedia: The Free Encyclopedia.
<http://en.wikipedia.org/wiki/Canny_edge_detector>.
- [21] "The Hough Transform." 15 April 2008. Wikipedia: The Free Encyclopedia.
<http://en.wikipedia.org/wiki/Hough_transform>.
- [22] "Synthetic Aperture Radar." 20 May 2008. Wikipedia: The Free Encyclopedia.
<http://en.wikipedia.org/wiki/Synthetic_aperture_radar>.
- [23] Zaharris, Brian. "Two-Dimensional Synthetic Aperture Radar Imaging and Moving Target Tracking Using the Range Doppler Algorithm Simulated in MATLAB." 2006.

Appendix A: Spectral Estimation Simulation Code

Driver File: dls3D.m

```
function [] = dls3D(sim_num,M,K,out_filename)
% Implement a 3-D simulation based on Kuklinski's Decoupled-Least Squares
% technique for high-resolution spectral estimation for the Maximum
% Likelihood estimates of point scatterer amplitudes and locations in the
% range-altitude axis.
%
% The "output" (not actually outputted) is a 4-D matrix, where each (x,y)
% location is a set of Kx2 points with the K locations of scatterers and
% their amplitudes for the given (x,y).
%
% sim_num ==> The simulation type to run: (1) is a single pixel with M
% frequency samples and 4 scatterers in the z axis, (2) is a
% 16x16 region of M frequency samples and a varying number of
% scatterers (0 to 4) in the z axis, 3) is MSTAR data of
% the 2S1 target excluding the phase (as it isn't available),
% and (4) is MSTAR data of the SLICY target. All simulations
% use S(x,y,Wz) frequency domain data as input.
% M ==> The number of frequency samples for each (x,y) data point.
% Equivalently the number of passes of the radar over the target
% region. This input is ignored for MSTAR simulations.
% K ==> The number of estimated scatterers to use in the DLS technique for
% each (x,y) location. Valid inputs are 2, 3, 4, and [], where []
% represents dynamic estimation of K for each (x,y) (based on the
% mean-squared errors of the estimations at that point)
% out_filename ==> String containing the filename of a CSV file to print
% the results in.

% Turn off warnings
warning off;

% get matrix S and radar parameters here
if (sim_num == 1)
    [S,fc,BW,phi_0,phi_min,phi_max] = getPixelData(M);
elseif (sim_num == 2)
    [S,fc,BW,phi_0,phi_min,phi_max] = getImageData(M);
elseif (sim_num == 3)
    [S,fc,BW,phi_0,phi_min,phi_max] = get2S1Data();
    M = 3; % overwrite the input
else
    [S,fc,BW,phi_0,phi_min,phi_max,Wz_S] = getSLICYData();
    M = 9; % overwrite the input
end

% Compute basic parameters and initialize K
Wc = 2*pi*((fc*2)/(3*10^8));

% Prepare the "output" matrices
if (isempty(K))
    dynamic_K = 1;
    img_dlsEst3D = zeros(size(S,1),size(S,2),4,2);
else
    dynamic_K = 0;
    img_dlsEst3D = zeros(size(S,1),size(S,2),K,2);
end
```

```

energy = zeros(size(S,1),size(S,2));

fprintf('\nComputing estimated reflector locations');

for (x = 1:size(S,1))
    fprintf('.');

    % Update the various parameters that depend upon the range dimension
    Wz = zeros(M,1);
    if (sim_num < 4)
        % Standard simulation and computations
        d_Wz = 2*pi*(2/(3*10^8))*(fc+(BW/size(S,1))* ...
            (x-((size(S,1)-1)/2)))* ((tan(phi_0-phi_min)- ...
            tan(phi_0-phi_max))/M);
        Lz = (2*pi)/d_Wz;
        d_z = (Lz/M) ^ 4; % min reflector spacing = 1/4*Fourier resolution

        % Create frequency values for this range
        for (m = 1:M)
            Wz(m) = Wc - ((M-1)*d_Wz)/2 + (m-1)*d_Wz;
        end
    else
        % SLICY simulation with inconsistent frequency sampling
        d_Wz = 2*pi*(2/(3*10^8))*(fc+(BW/size(S,1))* ...
            (x-((size(S,1)-1)/2)))* ((tan(phi_0-phi_min)- ...
            tan(phi_0-phi_max))/31); % pseudo M = 31
        Lz = (2*pi)/d_Wz;
        d_z = (Lz/M) ^ 4; % min reflector spacing = 1/4*Fourier resolution

        % Create frequency values for this range
        for (m = 1:M)
            Wz(m) = Wz_S(x,m);
        end
    end

    for (y = 1:size(S,2))
        % Get signal matrix for current (x,y)
        Y = squeeze(S(x,y,:)); % M x 1

        % Prepare gradient-search parameters
        fun = @(k)estimate(k,Y,Wz);
        ops = optimset('MaxFunEvals',50000,'MaxIter',500, ...
            'LargeScale','off','Display','off', ...
            'TolFun',1*10^(-9),'TolX',1*10^(-9), ...
            'TolCon',1*10^(-9));

        % Create starting estimates from the IDFT of the signal in z
        z_f = [0:d_z*4:Lz-d_z*4];
        Ys = zeros(M,1);
        for (j = 1:M)
            for (m = 1:M)
                Ys(j) = Ys(j) + Y(m)*exp(i*Wz(m)*z_f(j));
            end
            Ys(j) = abs(Ys(j))/M;
        end

        if (dynamic_K)
            mse = zeros(1,3);
            z_loc_loop = zeros(4,3);

            % Loop against all three possible Ks (2, 3, and 4)
            for (K = 2:4)

```

```

        % Get linear constraints matrices for the gradient search
        [A,b] = getConstraints(K,d_z,Lz);

        x0 = zeros(K,1);
        for (k = 1:K)
            [val,idx] = max(Ys);
            x0(k) = (idx-1)*d_z*4;
            Ys(idx) = 0;
        end
        x0 = sort(x0);

        [z_loc_loop(1:K,K-1),mse(K-1)] = ...
            fmincon(fun,x0,A,b,[],[],[],[],[],ops);
    end

    % Now use the estimates of the best K
    [discard,idx] = min(mse);
    K = idx + 1;
    z_loc = z_loc_loop(1:K,K-1);
else
    % Just use the K that was an input

    % Get linear constraints matrices for the gradient-based search
    [A,b] = getConstraints(K,d_z,Lz);

    x0 = zeros(K,1);
    for (k = 1:K)
        [val,idx] = max(Ys);
        x0(k) = (idx-1)*d_z*4;
        Ys(idx) = 0;
    end
    x0 = sort(x0);

    [z_loc] = fmincon(fun,x0,A,b,[],[],[],[],[],ops);
end

% Prepare the output matrix for this (x,y) location
V = transpose(conj(exp(-i*Wz*z_loc')));
a_ml = abs(inv(V*transpose(conj(V))*V*Y));
img_dlsEst3D(x,y,1:K,:) = [z_loc,a_ml];
energy(x,y) = sum(abs(Y).*abs(Y)); % for limiting plotted points
end
end

% Create output plots and show the "image" and elevation map
fprintf('\nCreating plots and outputs');
figure(1);
clf('reset');
maxEnergy = max(max(energy));
elev_map = zeros(size(S,1),size(S,2));
for (x = 1:size(S,1))
    fprintf('.');
    for (y = 1:size(S,2))
        Y = squeeze(S(x,y,:));
        if (sum(abs(Y).*abs(Y)) >= 0.25 * maxEnergy)
            maxAmplitude = max(img_dlsEst3D(x,y,:,2));
            if (dynamic_K)
                for (k = 1:4)
                    if (img_dlsEst3D(x,y,k,2) ~= 0) % amplitude != 0
                        if (img_dlsEst3D(x,y,k,2) == maxAmplitude)
                            plot3(x,y,img_dlsEst3D(x,y,k,1),'r. ');
                            elev_map(x,y) = img_dlsEst3D(x,y,k,1);
                        end
                    end
                end
            end
        end
    end
end

```

```

        hold all;
        elseif (img_dlsEst3D(x,y,k,2) >= 0.5*maxAmplitude)
            plot3(x,y,img_dlsEst3D(x,y,k,1),'b. ');
            if (img_dlsEst3D(x,y,k,1) > elev_map(x,y))
                elev_map(x,y) = img_dlsEst3D(x,y,k,1);
            end
            hold all;
        elseif (img_dlsEst3D(x,y,k,2) >= 0.25*maxAmplitude)
            plot3(x,y,img_dlsEst3D(x,y,k,1),'g. ');
            hold all;
        end
    end
end
else
    for (k = 1:K)
        if (img_dlsEst3D(x,y,k,2) == maxAmplitude)
            plot3(x,y,img_dlsEst3D(x,y,k,1),'r. ');
            elev_map(x,y) = img_dlsEst3D(x,y,k,1);
            hold all;
        elseif (img_dlsEst3D(x,y,k,2) >= 0.5 * maxAmplitude)
            plot3(x,y,img_dlsEst3D(x,y,k,1),'b. ');
            if (img_dlsEst3D(x,y,k,1) > elev_map(x,y))
                elev_map(x,y) = img_dlsEst3D(x,y,k,1);
            end
            hold all;
        elseif (img_dlsEst3D(x,y,k,2) >= 0.25 * maxAmplitude)
            plot3(x,y,img_dlsEst3D(x,y,k,1),'g. ');
            hold all;
        end
    end
end
end
end
end
grid on;
xlabel('Range');
ylabel('Cross-Range');
zlabel('Altitude (m)');
title('3-D View of Target Reflector Points');
view([-36,18]);
copyobj(gcf,0);
title('Side View of Target Reflector Points');
view([0,0]);

if (sim_num > 1)
    h = figure(3);
    surfc(elev_map);
    grid on;
    set(gca,'XDir','reverse');
    xlabel('Cross-Range');
    ylabel('Range');
    zlabel('Altitude');
    title('3-D View of Target Elevation Map');
    view([-38,62]);
    copyobj(h,0);
    view([0,90]);
    title('Top View of Target Elevation Map');
end

% Save to output CSV file for computation of MSE/MedSE and reference
csv_out = zeros(size(S,1)*size(S,2),11);
for (x = 1:size(S,1))

```

```

for (y = 1:size(S,2))
    csv_out((x-1)*size(S,1)+y,1) = x;
    csv_out((x-1)*size(S,1)+y,2) = y;
    csv_out((x-1)*size(S,1)+y,3) = img_dlsEst3D(x,y,1,1);
    csv_out((x-1)*size(S,1)+y,4) = img_dlsEst3D(x,y,2,1);
    if (K > 2 || dynamic_K) ...
        csv_out((x-1)*size(S,1)+y,5) = img_dlsEst3D(x,y,3,1);
    end
    if (K > 3 || dynamic_K) ...
        csv_out((x-1)*size(S,1)+y,6) = img_dlsEst3D(x,y,4,1);
    end
    csv_out((x-1)*size(S,1)+y,7) = img_dlsEst3D(x,y,1,2);
    csv_out((x-1)*size(S,1)+y,8) = img_dlsEst3D(x,y,2,2);
    if (K > 2 || dynamic_K) ...
        csv_out((x-1)*size(S,1)+y,9) = img_dlsEst3D(x,y,3,2);
    end
    if (K > 3 || dynamic_K) ...
        csv_out((x-1)*size(S,1)+y,10) = img_dlsEst3D(x,y,4,2);
    end
    csv_out((x-1)*size(S,1)+y,11) = elev_map(x,y);
end
end
csvwrite(out_filename, csv_out);

fprintf('\nDone.\n\n');

```

File: getPixelData.m

```

function [S,fc,BW,phi_0,phi_min,phi_max] = getPixelData(M)
% Returns a single pixel simulation with K = 4 scatterers in z and M
% frequency samples. Gaussian white noise is added to the signal at all
% samples.

% Radar data parameters
fc = 5*10^9;
BW = 0.5*10^9;

phi_0 = deg2rad(30);
phi_min = deg2rad(28);
phi_max = deg2rad(31.12425);

% Create empty signal matrix
S = zeros(1,1,M);

Wc = 2*pi*((fc*2)/(3*10^8));
d_Wz = 2*pi*(2/(3*10^8))*(fc+(BW/size(S,1))*(1-((size(S,1)-1)/2)))* ...
    ((tan(phi_0-phi_min)-tan(phi_0-phi_max))/M);

fprintf('\nGenerating input frequency samples');

% Create noise
n_r = randn(1,M); % real
n_q = randn(1,M); % quadrature
n = n_r + i*n_q;
n = n - mean(n);
n = n/std(n);

```



```

Wz = zeros(M,1);
Y = zeros(M,1);
for (m = 1:M)
    fprintf('.');
    Wz(m) = Wc - ((M-1)*d_Wz)/2 + (m-1)*d_Wz;
    Y(m) = 10*exp(-i*Wz(m)*2.0) + 10*exp(-i*Wz(m)*2.4) + ...
        0.5*exp(-i*Wz(m)*3.5) + 7*exp(-i*Wz(m)*4.25) + n(m);
end
S(1,1,:) = Y;

```

File: getImageData.m

```

function [S,fc,BW,phi_0,phi_min,phi_max] = getImageData(M)
% Returns a 16x16 image simulation with K = [0,4] scatterers in z and
% M frequency samples. Gaussian white noise is added to the signal at
% all samples.

% Radar data parameters
fc = 5*10^9;
BW = 0.5*10^9;

phi_0 = deg2rad(30);
phi_min = deg2rad(28);
phi_max = deg2rad(32);

Wc = 2*pi*((fc*2)/(3*10^8));

% Create empty signal matrix
S = zeros(16,16,M);

fprintf('\nGenerating input frequency samples');

% Import scatterer locations and amplitudes from CSV file
scene = csvread('image.csv');

% Create signal matrix
for (x = 1:size(S,1))
    fprintf('.');

    % Update parameters for this range x
    d_Wz = 2*pi*(2/(3*10^8))*(fc+(BW/size(S,1))*(x-((size(S,1)-1)/2)))* ...
        ((tan(phi_0-phi_min)-tan(phi_0-phi_max))/M);

    % Create the matrix containing all frequency values Wz for this range x
    Wz = zeros(M,1);
    for (m = 1:M)
        Wz(m) = Wc - ((M-1)*d_Wz)/2 + (m-1)*d_Wz;
    end

    for (y = 1:16)
        Y = zeros(M,1);

        % Create noise
        n_r = randn(1,M); % real

```

```

n_q = randn(1,M); % quadrature
n = n_r + i*n_q;
n = n - mean(n);
n = n/std(n);

% Get scene information (for up to K = 4 scatterers)
z1 = scene((x-1)*size(S,1)+y,3);
z2 = scene((x-1)*size(S,1)+y,4);
z3 = scene((x-1)*size(S,1)+y,5);
z4 = scene((x-1)*size(S,1)+y,6);
a1 = scene((x-1)*size(S,1)+y,7);
a2 = scene((x-1)*size(S,1)+y,8);
a3 = scene((x-1)*size(S,1)+y,9);
a4 = scene((x-1)*size(S,1)+y,10);

% Generate frequency samples (with noise)
for (m = 1:M)
    Y(m) = a1*exp(-i*Wz(m)*z1) + a2*exp(-i*Wz(m)*z2) + ...
           a3*exp(-i*Wz(m)*z3) + a4*exp(-i*Wz(m)*z4) + n(m);
end

S(x,y,:) = Y;
end
end

```

File: get2S1Data.m

```

function [S,fc,BW,phi_0,phi_min,phi_max] = get2S1Data()
% Returns a set of M = 3 frequency samples for each (x,y) location in the
% set of input SAR images of the 2S1 target. Limited to M = 3 due to the
% lack of available MSTAR data

% Radar data parameters (from MSTAR headers and platform information)
fc = 9.599*10^9;
BW = 0.5991*10^9;

phi_0 = deg2rad(30);
phi_min = deg2rad(15);
phi_max = deg2rad(45);

Wc = 2*pi*((fc*2)/(3*10^8));

% Create output matrix
M = 3;
S = zeros(158,158,M);

% Collect all three input matrices
fid = fopen('..\..\SAR Data\DLS MSTAR Input\2S1\HB15140.000.all');
img1_mag = fread(fid,size(S,1)*size(S,2),'float32');
img1_mag = flipdim(transpose(reshape(img1_mag,size(S,1),size(S,2))),1);
img1_phase = fread(fid,size(S,1)*size(S,2),'float32');
img1_phase = flipdim(transpose(reshape(img1_phase,size(S,1),size(S,2))),1);
fclose(fid);
fid2 = fopen('..\..\SAR Data\DLS MSTAR Input\2S1\HB15335.000.all');
img2_mag = fread(fid2,size(S,1)*size(S,2),'float32');
img2_mag = flipdim(transpose(reshape(img2_mag,size(S,1),size(S,2))),1);

```

```

img2_phase = fread(fid2,size(S,1)*size(S,2),'float32');
img2_phase = flipdim(transpose(reshape(img2_phase,size(S,1),size(S,2))),1);
fclose(fid2);
fid3 = fopen('..\..\SAR Data\DLS MSTAR Input\2S1\HB17054.000.all');
img3_mag = fread(fid3,size(S,1)*size(S,2),'float32');
img3_mag = flipdim(transpose(reshape(img3_mag,size(S,1),size(S,2))),1);
img3_phase = fread(fid3,size(S,1)*size(S,2),'float32');
img3_phase = flipdim(transpose(reshape(img3_phase,size(S,1),size(S,2))),1);
fclose(fid3);

% Overwrite with TIFF data (higher magnitude, helps in viewing/scaling)
img1_mag = flipdim(imread('..\..\SAR Data\DLS MSTAR Input\2S1\HB15140.tif'),1);
img2_mag = flipdim(imread('..\..\SAR Data\DLS MSTAR Input\2S1\HB15335.tif'),1);
img3_mag = flipdim(imread('..\..\SAR Data\DLS MSTAR Input\2S1\HB17054.tif'),1);

fprintf('\nGenerating input frequency samples');

% Fill in the output matrix
for (x = 1:size(S,1))
    fprintf('.');
    for (y = 1:size(S,2))
        S(x,y,1) = double(img1_mag(x,y))*exp(i*img1_phase(x,y));
        S(x,y,2) = double(img2_mag(x,y))*exp(i*img2_phase(x,y));
        S(x,y,3) = double(img3_mag(x,y))*exp(i*img3_phase(x,y));
    end
end
end

```

File: getSLICYData.m

```

function [S,fc,BW,phi_0,phi_min,phi_max,Wz_S] = getSLICYData()
% Returns a set of M = 9 frequency samples for each (x,y) location in the
% set of input SAR images of the SLICY target. Limited to M = 9 due to the
% lack of available MSTAR data

% Radar data parameters (from MSTAR headers and platform information)
fc = 9.599*10^9;
BW = 0.5991*10^9;

phi_0 = deg2rad(30);
phi_min = deg2rad(15);
phi_max = deg2rad(45);

Wc = 2*pi*((fc*2)/(3*10^8));

% Create output matrix
M = 9;
Wz_S = zeros(54,M);
S = zeros(54,54,M);

% Collect all nine input matrices
fid = fopen('..\..\SAR Data\DLS MSTAR Input\SLICY\HB15192_15.015.all');
img1_mag = fread(fid,size(S,1)*size(S,2),'float32');
img1_mag = flipdim(transpose(reshape(img1_mag,size(S,1),size(S,2))),1);
img1_phase = fread(fid,size(S,1)*size(S,2),'float32');
img1_phase = flipdim(transpose(reshape(img1_phase,size(S,1),size(S,2))),1);
fclose(fid);

```

```

fid2 = fopen('..\..\SAR Data\DLS MSTAR Input\SLICY\HB17695_16.015.all');
img2_mag = fread(fid2,size(S,1)*size(S,2),'float32');
img2_mag = flipdim(transpose(reshape(img2_mag,size(S,1),size(S,2))),1);
img2_phase = fread(fid2,size(S,1)*size(S,2),'float32');
img2_phase = flipdim(transpose(reshape(img2_phase,size(S,1),size(S,2))),1);
fclose(fid2);
fid3 = fopen('..\..\SAR Data\DLS MSTAR Input\SLICY\HB19434_17.015.all');
img3_mag = fread(fid3,size(S,1)*size(S,2),'float32');
img3_mag = flipdim(transpose(reshape(img3_mag,size(S,1),size(S,2))),1);
img3_phase = fread(fid3,size(S,1)*size(S,2),'float32');
img3_phase = flipdim(transpose(reshape(img3_phase,size(S,1),size(S,2))),1);
fclose(fid3);
fid4 = fopen('..\..\SAR Data\DLS MSTAR Input\SLICY\HB17956_29.015.all');
img4_mag = fread(fid4,size(S,1)*size(S,2),'float32');
img4_mag = flipdim(transpose(reshape(img4_mag,size(S,1),size(S,2))),1);
img4_phase = fread(fid4,size(S,1)*size(S,2),'float32');
img4_phase = flipdim(transpose(reshape(img4_phase,size(S,1),size(S,2))),1);
fclose(fid4);
fid5 = fopen('..\..\SAR Data\DLS MSTAR Input\SLICY\HB15377_30.015.all');
img5_mag = fread(fid5,size(S,1)*size(S,2),'float32');
img5_mag = flipdim(transpose(reshape(img5_mag,size(S,1),size(S,2))),1);
img5_phase = fread(fid5,size(S,1)*size(S,2),'float32');
img5_phase = flipdim(transpose(reshape(img5_phase,size(S,1),size(S,2))),1);
fclose(fid5);
fid6 = fopen('..\..\SAR Data\DLS MSTAR Input\SLICY\HB19596_31.015.all');
img6_mag = fread(fid6,size(S,1)*size(S,2),'float32');
img6_mag = flipdim(transpose(reshape(img6_mag,size(S,1),size(S,2))),1);
img6_phase = fread(fid6,size(S,1)*size(S,2),'float32');
img6_phase = flipdim(transpose(reshape(img6_phase,size(S,1),size(S,2))),1);
fclose(fid6);
fid7 = fopen('..\..\SAR Data\DLS MSTAR Input\SLICY\HB18265_43.015.all');
img7_mag = fread(fid7,size(S,1)*size(S,2),'float32');
img7_mag = flipdim(transpose(reshape(img7_mag,size(S,1),size(S,2))),1);
img7_phase = fread(fid7,size(S,1)*size(S,2),'float32');
img7_phase = flipdim(transpose(reshape(img7_phase,size(S,1),size(S,2))),1);
fclose(fid7);
fid8 = fopen('..\..\SAR Data\DLS MSTAR Input\SLICY\HB18705_44.015.all');
img8_mag = fread(fid8,size(S,1)*size(S,2),'float32');
img8_mag = flipdim(transpose(reshape(img8_mag,size(S,1),size(S,2))),1);
img8_phase = fread(fid8,size(S,1)*size(S,2),'float32');
img8_phase = flipdim(transpose(reshape(img8_phase,size(S,1),size(S,2))),1);
fclose(fid8);
fid9 = fopen('..\..\SAR Data\DLS MSTAR Input\SLICY\HB16984_45.015.all');
img9_mag = fread(fid9,size(S,1)*size(S,2),'float32');
img9_mag = flipdim(transpose(reshape(img9_mag,size(S,1),size(S,2))),1);
img9_phase = fread(fid9,size(S,1)*size(S,2),'float32');
img9_phase = flipdim(transpose(reshape(img9_phase,size(S,1),size(S,2))),1);
fclose(fid9);

% Overwrite mag with TIFF data (higher magnitude, helps in viewing/scaling)
img1_mag = flipdim( ...
    imread('..\..\SAR Data\DLS MSTAR Input\SLICY\HB15192_15.tif'),1);
img2_mag = flipdim( ...
    imread('..\..\SAR Data\DLS MSTAR Input\SLICY\HB17695_16.tif'),1);
img3_mag = flipdim( ...
    imread('..\..\SAR Data\DLS MSTAR Input\SLICY\HB19434_17.tif'),1);
img4_mag = flipdim( ...
    imread('..\..\SAR Data\DLS MSTAR Input\SLICY\HB17956_29.tif'),1);
img5_mag = flipdim( ...
    imread('..\..\SAR Data\DLS MSTAR Input\SLICY\HB15377_30.tif'),1);
img6_mag = flipdim( ...
    imread('..\..\SAR Data\DLS MSTAR Input\SLICY\HB19596_31.tif'),1);

```

```

img7_mag = flipdim( ...
    imread('..\..\SAR Data\DLS MSTAR Input\SLICY\HB18265_43.tif'),1);
img8_mag = flipdim( ...
    imread('..\..\SAR Data\DLS MSTAR Input\SLICY\HB18705_44.tif'),1);
img9_mag = flipdim( ...
    imread('..\..\SAR Data\DLS MSTAR Input\SLICY\HB16984_45.tif'),1);

fprintf('\nGenerating input frequency samples');

% Fill in the output matrix
for (x = 1:size(S,1))
    fprintf('.');

    % Update the frequency parameters that depend upon the range dimension
    % M = 31 is used in the computation, since between 15 and 45 degrees
    % there would be 31 samples (we happen to only have 9 of them)
    d_Wz = 2*pi*(2/(3*10^8))*(fc+(BW/size(S,1))*(x-((size(S,1)-1)/2)))* ...
        ((tan(phi_0-phi_min)-tan(phi_0-phi_max))/31);

    % Now fill in the actual frequencies available for this range location
    for (m = 1:M)
        % Wz(m) = Wc - ((M-1)*d_Wz)/2 + (m-1)*d_Wz;
        if (m <= 3)
            Wz_S(x,m) = Wc - ((31-1)*d_Wz)/2 + (m-1)*d_Wz;
        elseif (m <= 6)
            Wz_S(x,m) = Wc - ((31-1)*d_Wz)/2 + (m+11-1)*d_Wz;
        else
            Wz_S(x,m) = Wc - ((31-1)*d_Wz)/2 + (m+22-1)*d_Wz;
        end
    end

    for (y = 1:size(S,2))
        S(x,y,1) = double(img1_mag(x,y))*exp(i*img1_phase(x,y));
        S(x,y,2) = double(img2_mag(x,y))*exp(i*img2_phase(x,y));
        S(x,y,3) = double(img3_mag(x,y))*exp(i*img3_phase(x,y));
        S(x,y,4) = double(img4_mag(x,y))*exp(i*img4_phase(x,y));
        S(x,y,5) = double(img5_mag(x,y))*exp(i*img5_phase(x,y));
        S(x,y,6) = double(img6_mag(x,y))*exp(i*img6_phase(x,y));
        S(x,y,7) = double(img7_mag(x,y))*exp(i*img7_phase(x,y));
        S(x,y,8) = double(img8_mag(x,y))*exp(i*img8_phase(x,y));
        S(x,y,9) = double(img9_mag(x,y))*exp(i*img9_phase(x,y));
    end
end
end

```

File: getConstraints.m

```

function [A,b] = getConstraints(K,d_z,Lz)
% Creates linear constraints matrices A & b for the gradient-based
% minimization search such that A*k=b where k is a given estimate of
% reflector locations in the z dimension

A = zeros(3*K-1,K);
b = zeros(3*K-1,1);

for (k = 1:K-1)
    % A matrix

```

```

    A(k,k) = 1;
    A(k,k+1) = -1;

    % b matrix
    b(k) = -1*d_z;
end

k = K;
j = 1;
while (k < 3*K-1)
    % A matrix
    A(k,j) = -1;
    A(k+1,j) = 1;

    % b matrix
    b(k) = -1*(j-1)*d_z;
    b(k+1) = Lz-(K-j)*d_z;
    k = k + 2;
    j = j + 1;
end

```

File: estimate.m

```

function [mse] = estimate(k,Y,Wz)
% Takes in a set of K estimated reflector locations and the current
% frequency samples Y, makes a least-squares estimate of the scatterer
% amplitudes and computes the mean-squared error cost function of the
% current estimate compared to the samples in Y

% Create V matrix based on current scatterer locations
V = transpose(conj(exp(-i*Wz*k')));

% Estimate scatterer amplitudes
a_ml = inv(V*transpose(conj(V)))*V*Y;

% Compute the mean-squared error of the estimate
mse = transpose(conj((Y - transpose(conj(V))*a_ml)))* ...
    (Y - transpose(conj(V))*a_ml);

```

Appendix B: Spectral Estimation Simulation Test Matrix

The following table is the matrix of fabricated target reflector positions (in the z axis) and amplitudes, indexed by range-cross-range (x,y) location. The 3-D shape of the fabricated target is a tank-like vehicle, complete with turret and cannon.

The columns which contain the z locations of the reflectors are highlighted in light gray to help discern between positions and amplitudes in the lower rows of the table.

All (x,y) points in the image matrix contain between 1 and 4 reflectors with amplitude greater than 0. The majority of the points in which the target is located have 3 reflectors, but a few have 4 reflectors.

The target reflector locations and amplitudes shown here are pre-noise, as the complex Additive Gaussian White Noise samples are added during the creation of the $S(x, y, \omega_z)$ matrix in MATLAB.

x	y	z_1	z_2	z_3	z_4	a_1	a_2	a_3	a_4
1	1	0	0.05	0	0.15	0	0.075	0.1	0.25
1	2	0	0	0	0	0	0	0	0
1	3	0	0	0	0	0	0	0	0
1	4	0	0	0	0	0	0	0	0
1	5	0	0	0	0	0	0	0	0
1	6	0	0	0	0	0	0	0	0
1	7	0	0	0	0	0	0	0	0
1	8	0.01	0.05	0.1	0.65	0.01	0.2	0.15	1
1	9	0	0	0	0	0	0	0	0
1	10	0	0	0	0	0	0	0	0
1	11	0	0	0	0	0	0	0	0
1	12	0	0	0	0	0	0	0	0
1	13	0	0	0.7	1.15	0	0.05	1.5	0.15
1	14	0	0	0	0	0	0	0	0
1	15	0	0	0	0	0	0	0	0
1	16	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0
2	2	0	0	0	0	0	0	0	0
2	3	0	0	0	0	0	0	0	0

2	4	0	0	0	0	0	0	0	0
2	5	0	0	0	0	0	0	0	0
2	6	0	0	0	0	0	0	0	0
2	7	0	0	0	0	0	0	0	0
2	8	0	0	0	0.5	0.1	0	0.1	0.3
2	9	0	0	0	0	0	0	0	0
2	10	0	0	0	0	0	0	0	0
2	11	0.25	0	0	0	0.1	0	0	0.15
2	12	0	0	0	0	0	0	0	0
2	13	0	0	0	0	0	0	0	0
2	14	0	0	0	0	0	0	0	0
2	15	0	0	0	0	0	0	0	0
2	16	0	0	0	0	0	0	0	0
3	1	0	0.15	0	0	0	0.08	0	0
3	2	0	0	0	0	0	0	0	0
3	3	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0
3	5	0	0	0	0	0	0	0	0
3	6	0	0	0	0	0	0	0	0
3	7	0.55	0.8	0	0	0.9	0.15	0	0.1
3	8	0	0	0	0	0	0	0	0
3	9	0	0	0	0	0	0	0	0
3	10	0	0	0	0	0	0	0	0
3	11	0	0	0	0	0	0	0	0
3	12	0	0	0	0	0	0	0	0
3	13	0	0	0	0	0	0	0	0
3	14	0	0	0	0	0	0	0	0
3	15	0	0	0	0.05	0.19	0.06	0	0.1
3	16	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0
4	2	0	0	0	0	0	0	0	0
4	3	0	0.15	0	0	0	0.08	0	0
4	4	0	0	0	0	0	0	0	0
4	5	0.1	1.96	2	0	7.2	12.5	0.4	0
4	6	0.11	1.95	2	0	7.1	12.5	0.4	0
4	7	0.1	1.96	2	0	7.2	12.5	0.4	0
4	8	0.1	1.96	2	0	7.2	12.5	0.35	0
4	9	0.1	1.93	2.02	0	7.2	12.5	0.4	0
4	10	0.11	1.95	2	0	7.1	12.5	0.4	0
4	11	0.09	1.96	1.99	0	6.45	12.45	0.4	0
4	12	0	0	0	0	0	0	0	0
4	13	0	0	0	0	0	0	0	0
4	14	0	0.15	0	0	0	0.08	0	0
4	15	0	0	0	0	0	0	0	0
4	16	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0
5	2	0	0	0	0	0	0	0	0
5	3	0	0	0	0	0	0	0	0
5	4	0.1	1.93	2	0	7.3	12.6	0.405	0
5	5	0.1	1.96	2	0	5.3	10.5	0.35	0
5	6	0.11	1.95	2	0	5.2	10.5	0.35	0
5	7	0.1	1.96	2	0	5.1	10.5	0.345	0
5	8	0.1	1.96	2	0	5.2	10.5	0.35	0

5	9	0.1	1.93	2.02	0	5.2	10.5	0.39	0
5	10	0.11	1.95	2	0	5.25	10.45	0.4	0
5	11	0.09	1.96	1.99	0	5.04	10.55	0.35	0
5	12	0.1	1.93	2	2.1	7.3	10.6	0.36	4.25
5	13	0	0	0	0	0	0	0	0
5	14	0	0	0	0	0	0	0	0
5	15	0	0	0	0	0	0	0	0
5	16	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0
6	2	0	0	0	0	0	0	0	0
6	3	0	0	0	0	0	0	0	0
6	4	0.1	1.93	2	0	7.3	12.6	0.405	0
6	5	0.1	1.96	2	0	5.3	10.5	0.35	0
6	6	0.11	1.95	2	0	5.2	10.5	0.35	0
6	7	0.1	1.96	2	0	5.1	10.5	0.345	0
6	8	0.1	1.96	2	0	5.2	10.5	0.35	0
6	9	0.1	1.93	2.02	0	5.2	10.5	0.39	0
6	10	0.11	1.95	2	0	5.25	10.45	0.4	0
6	11	0.09	1.96	1.99	0	5.04	10.55	0.35	0
6	12	0.1	1.93	2	2.1	7.3	10.6	0.36	4.25
6	13	0	0	0	0	0	0	0	0
6	14	0	0	0	0	0	0	0	0
6	15	0	0	0	0	0	0	0	0
6	16	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0
7	2	0	0	0	0	0	0	0	0
7	3	0	0	0	0	0	0	0	0
7	4	0.1	1.93	2	0	7.3	12.6	0.405	0
7	5	0.1	1.96	2	0	5.3	10.5	0.35	0
7	6	0.11	1.95	2	0	5.2	10.5	0.35	0
7	7	0.2	2	2.75	0	3.5	16.1	10.5	0
7	8	0.18	2	2.75	0	3.4	16.4	10.5	0
7	9	0.2	2	2.75	0	3.5	16.1	10.5	0
7	10	0.11	1.95	2	0	5.25	10.45	0.4	0
7	11	0.09	1.96	1.99	0	5.04	10.55	0.35	0
7	12	0.1	1.93	2	2.1	7.3	10.6	0.36	4.25
7	13	0	0	0	0	0	0	0	0
7	14	0	0	0	0	0	0	0	0
7	15	0	0	0	0	0	0	0	0
7	16	0	0	0	0	0	0	0	0
8	1	0	0	0	0	0	0	0	0
8	2	0	0	0	0	0	0	0	0
8	3	0	0	0	0	0	0	0	0
8	4	0.1	1.93	2	0	7.3	12.6	0.405	0
8	5	0.1	1.96	2	0	5.3	10.5	0.35	0
8	6	0.11	1.95	2	0	5.2	10.5	0.35	0
8	7	0.2	2	2.75	0	3.5	16.1	10.5	0
8	8	0.18	2	3.25	0	3.45	16.35	12.75	0
8	9	0.2	2	2.75	0	3.5	16.1	10.5	0
8	10	0.11	1.95	2	0	5.25	10.45	0.4	0
8	11	0.09	1.96	1.99	0	5.04	10.55	0.35	0
8	12	0.1	1.93	2	2.1	7.3	10.6	0.36	4.25
8	13	0	0	0	0	0	0	0	0

8	14	0	0	0	0	0	0	0	0
8	15	0	0	0	0	0	0	0	0
8	16	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0
9	2	0	0	0	0	0	0	0	0
9	3	0	0	0	0	0	0	0	0
9	4	0.1	1.93	2	0	7.3	12.6	0.405	0
9	5	0.1	1.96	2	0	5.3	10.5	0.35	0
9	6	0.11	1.95	2	0	5.2	10.5	0.35	0
9	7	0.2	2	2.75	0	3.5	16.1	10.5	0
9	8	0.18	2	3.25	0	3.45	16.35	12.75	0
9	9	0.2	2	2.75	0	3.5	16.1	10.5	0
9	10	0.11	1.95	2	0	5.25	10.45	0.4	0
9	11	0.09	1.96	1.99	0	5.04	10.55	0.35	0
9	12	0.1	1.93	2	2.1	7.3	10.6	0.36	4.25
9	13	0	0	0	0	0	0	0	0
9	14	0	0	0	0	0	0	0	0
9	15	0	0	0	0	0	0	0	0
9	16	0	0	0	0	0	0	0	0
10	1	0	0	0	0	0	0	0	0
10	2	0	0	0	0	0	0	0	0
10	3	0	0	0	0	0	0	0	0
10	4	0.1	1.93	2	0	7.3	12.6	0.405	0
10	5	0.1	1.96	2	0	5.3	10.5	0.35	0
10	6	0.11	1.95	2	0	5.2	10.5	0.35	0
10	7	0.1	1.96	2	0	5.1	10.5	0.345	0
10	8	0.18	2	3.25	3.3	5	10	12.75	12
10	9	0.1	1.93	2.02	0	5.2	10.5	0.39	0
10	10	0.11	1.95	2	0	5.25	10.45	0.4	0
10	11	0.09	1.96	1.99	0	5.04	10.55	0.35	0
10	12	0.1	1.93	2	2.1	7.3	10.6	0.36	4.25
10	13	0	0	0	0	0	0	0	0
10	14	0	0	0	0	0	0	0	0
10	15	0	0	0	0	0	0	0	0
10	16	0	0	0	0	0	0	0	0
11	1	0	0	0	0	0	0	0	0
11	2	0	0	0	0	0	0	0	0
11	3	0	0	0	0	0	0	0	0
11	4	0.1	1.93	2	0	7.3	12.6	0.405	0
11	5	0.1	1.96	2	0	5.3	10.5	0.35	0
11	6	0.11	1.95	2	0	5.2	10.5	0.35	0
11	7	0.1	1.96	2	0	5.1	10.5	0.345	0
11	8	0.18	2	3.25	3.3	5	10	12.75	12
11	9	0.1	1.93	2.02	0	5.2	10.5	0.39	0
11	10	0.11	1.95	2	0	5.25	10.45	0.4	0
11	11	0.09	1.96	1.99	0	5.04	10.55	0.35	0
11	12	0.1	1.93	2	0	7.3	10.6	0.36	0
11	13	0	0	0	0	0	0	0	0
11	14	0	0	0	0	0	0	0	0
11	15	0	0	0	0	0	0	0	0
11	16	0	0	0	0	0	0	0	0
12	1	0	0	0	0	0	0	0	0
12	2	0	0	0	0	0	0	0	0

12	3	0	0.15	0	0	0	0.08	0	0
12	4	0	0	0	0	0	0	0	0
12	5	0.1	1.96	2	0	7.2	12.5	0.4	0
12	6	0.11	1.95	2	0	7.1	12.5	0.4	0
12	7	0.1	1.96	2	0	7.2	12.5	0.4	0
12	8	0.18	2	3.25	3.3	5	10	12.75	12
12	9	0.1	1.93	2.02	0	7.2	12.5	0.4	0
12	10	0.11	1.95	2	0	7.1	12.5	0.4	0
12	11	0.09	1.96	1.99	0	6.45	12.45	0.4	0
12	12	0	0	0	0	0	0	0	0
12	13	0	0	0	0	0	0	0	0
12	14	0	0.15	0	0	0	0.08	0	0
12	15	0	0	0	0	0	0	0	0
12	16	0	0	0	0	0	0	0	0
13	1	0	0	0	0	0	0	0	0
13	2	0	0.15	0	0	0	0.08	0	0
13	3	0	0	0	0	0	0	0	0
13	4	0	0	0	0	0	0	0	0
13	5	0	0	0	0	0	0	0	0
13	6	0	0	0	0	0	0	0	0
13	7	0	0	0	0	0	0	0	0
13	8	3.25	3.3	0	0	12.75	12	0	0
13	9	0	0	0	0	0	0	0	0
13	10	0	0	0	0	0	0	0	0
13	11	0	0	0	0	0	0	0	0
13	12	0	0	0	0	0	0	0	0
13	13	0	0	0	0	0	0	0	0
13	14	0	0	0	0	0	0	0	0
13	15	0	0	0	0	0	0	0	0
13	16	0	0	0	0	0	0	0	0
14	1	0	0	0	0	0	0	0	0
14	2	0	0	0	0	0	0	0	0
14	3	0	0	0	0	0	0	0	0
14	4	0	0	0	0	0	0	0	0
14	5	0	0	0	0	0	0	0	0
14	6	0	0	0	0	0	0	0	0
14	7	0	0	0	0	0	0	0	0
14	8	3.25	3.3	0.11	0	12.75	12	1.15	0
14	9	0	0	0	0	0	0	0	0
14	10	0	0	0	0	0	0	0	0
14	11	0	0	0	0	0	0	0	0
14	12	0	0.15	0	0	0	0.08	0	0
14	13	0	0	0	0	0	0	0	0
14	14	0	0	0	0	0	0	0	0
14	15	0	0	0	0	0	0	0	0
14	16	0	0	0	0	0	0	0	0
15	1	0	0	0	0	0	0	0	0
15	2	0	0	0	0	0	0	0	0
15	3	0	0	0	0	0	0	0	0
15	4	0	0	0	0	0	0	0	0
15	5	0	0	0	0	0	0	0	0
15	6	0	0	0	0	0	0	0	0
15	7	0	0	0	0	0	0	0	0

15	8	3.25	3.3	0	0	12.75	12	0	0
15	9	0	0	0	0	0	0	0	0
15	10	0	0	0	0	0	0	0	0
15	11	0	0.15	0	0	0	0.08	0	0
15	12	0	0	0	0	0	0	0	0
15	13	0	0	0	0	0	0	0	0
15	14	0	0	0	0	0	0	0	0
15	15	0	0	0	0	0	0	0	0
15	16	0	0	0	0	0	0	0	0
16	1	0	0	0	0	0	0	0	0
16	2	0	0	0	0	0	0	0	0
16	3	0	0.15	0	0	0	0.08	0	0
16	4	0	0	0	0	0	0	0	0
16	5	0	0	0	0	0	0	0	0
16	6	0	0	0	0	0	0	0	0
16	7	0	0	0	0	0	0	0	0
16	8	0	0	0	0	0	0	0	0
16	9	0	0	0	0	0	0	0	0
16	10	0	0	0	0	0	0	0	0
16	11	0	0	0	0	0	0	0	0
16	12	0	0	0	0	0	0	0	0
16	13	0	0	0	0	0	0	0	0
16	14	0	0	0	0	0	0	0	0
16	15	0	0	0	0	0	0	0	0
16	16	0	0	0	0	0	0	0	0

Appendix C: Shadow Technique Test Sets

Set 1: 2S1 at 30° with 4 Inputs and MSTAR Rotation

Set 2: BRDM_2 at 45° with 3 Inputs and Automatic Rotation

Set 3: D7 at 15° with 4 Inputs and Automatic Rotation

Set 4: 2S1 at 15° with 5 Inputs and Automatic

Set 5: BRDM_2 at 15° with 4 Inputs and Automatic Rotation

Set 6: BRDM_2 at 30° with 5 Inputs and Automatic Rotation

Set 7: ZSU_23_4 at 45° with 5 Inputs and Automatic Rotation

Set 8: 2S1 at 45° with 6 Inputs and MSTAR Rotation

Set 9: BRDM_2 at 17° with 3 Inputs and MSTAR Rotation

Set 10: 2S1 at 17° with 3 Inputs and MSTAR Rotation

Set 11: 2S1 at 15° with 3 Inputs and MSTAR Rotation

Set 12: ZSU_23_4 at 17° with 3 Inputs and MSTAR Rotation

Set 13: ZSU_23_4 at 15° with 3 Inputs and MSTAR Rotation

Set 14: D7 at 17° with 5 Inputs and Automatic Rotation

Set 15: T62 at 17° with 3 Inputs and MSTAR Rotation

Set 16: T62 at 17° with 4 Inputs and MSTAR Rotation

Set 17: BTR_60 at 17° with 3 Inputs and Automatic Rotation

Set 18: BTR_60 at 17° with 4 Inputs and MSTAR Rotation

Set 19: BTR_60 at 15° with 4 Inputs and MSTAR Rotation

Set 20: T62 at 15° with 5 Inputs and Automatic Rotation

Set 21: D7 at 17° with 3 Inputs and MSTAR Rotation

Set 22: ZSU_23_4 at 30° with 4 Inputs and Automatic Rotation

Appendix D: Shadow Technique Code

Driver File: sar3D.m

```
function [] = sar3D(set_num,to_process)
% Attempt to extract 3D information from a single or multiple SAR images
% using a shadow-extraction technique
%
% set_num ==> Set number in the input data folder to look for the images in
% to_process ==> Set of image number indeces to process. If this is equal
%                 to the empty set, then all images in the folder will be
%                 processed.

close all;

dir_list = dir('..\..\SAR Data\3D Sets\');

for i = 3:numel(dir_list)
    if (strcmp(int2str(set_num),strtok(dir_list(i).name,'-')))
        sub_dir = dir_list(i).name;
        angle = regexpi(sub_dir,'\-', 'split');
        angle = sscanf(char(angle(2)), '%d');
    end
end

image_names = dir(strcat('..\..\SAR Data\3D Sets\',sub_dir,'*.tif'));
sample_imgs = dir(strcat('..\..\SAR Data\3D Sets\',sub_dir,'*.jpg'));

for i = 1:numel(image_names)
    if (isempty(to_process) || ~isempty(find(to_process == i,1)))
        img_in = imread(strcat('..\..\SAR Data\3D Sets\',sub_dir, ...
                                '\',image_names(i).name));

        figure(i);
        subplot(2,2,1);
        imshow(img_in);
        title('Input Image');

        if (i == 1)
            % Pre-allocate heights matrix slice set
            object_slices = zeros(size(img_in,1),size(img_in,2), ...
                                    size(image_names,1));
            rotations = zeros(length(image_names));
        end

        % Extract object mask and object orientation
        [img_object,rotations(i)] = getObject(img_in);
        figure(i);
        subplot(2,2,2);
        imshow(img_object);
        title('Object');

        % Extract shadow mask
        img_shadow = getShadow(img_in,img_object);
        figure(i);
        subplot(2,2,3);
        imshow(img_shadow);
        title('Shadow');
```

```

% Iterate horizontally along the object and compute an object
% height for each column
img_heights = getSlice(img_object,img_shadow,angle);

% TEST VOLUME
vols(i) = sum(sum(img_heights));
avgs(i) = mean(mean(nonzeros(img_heights)));
vehicle = dir(strcat('..\..\SAR Data\3D Sets\',sub_dir,'\*.txt'));

% Hard-Coded From the Image File Headers
if (strcmp(image_names(i).name,'hb15269.tif') &&
strcmp(sample_imgs(1).name,'2S1.jpg')) % Set 1
    rotations(i) = -338.224854 + 90;
elseif (strcmp(image_names(i).name,'hb15277.tif') &&
strcmp(sample_imgs(1).name,'2S1.jpg'))
    rotations(i) = -26.224838 + 90;
elseif (strcmp(image_names(i).name,'hb15282.tif') &&
strcmp(sample_imgs(1).name,'2S1.jpg'))
    rotations(i) = -62.224838 + 90;
elseif (strcmp(image_names(i).name,'hb15298.tif') &&
strcmp(sample_imgs(1).name,'2S1.jpg'))
    rotations(i) = -158.224838 + 90;
elseif (strcmp(image_names(i).name,'hb17007.tif') &&
strcmp(sample_imgs(1).name,'BRDM_2.jpg')) % Set 2
    rotations(i) = -89.325272 + 90;
elseif (strcmp(image_names(i).name,'hb16962.tif') &&
strcmp(sample_imgs(1).name,'BRDM_2.jpg'))
    rotations(i) = -127.325272 + 90;
elseif (strcmp(image_names(i).name,'hb17057.tif') &&
strcmp(sample_imgs(1).name,'BRDM_2.jpg'))
    rotations(i) = -37.325272 + 90;
elseif (strcmp(image_names(i).name,'hb16948.tif') &&
strcmp(sample_imgs(1).name,'2S1.jpg')) % Set 8
    rotations(i) = -27.224838 + 90;
elseif (strcmp(image_names(i).name,'hb16985.tif') &&
strcmp(sample_imgs(1).name,'2S1.jpg'))
    rotations(i) = -293.224854 + 90;
elseif (strcmp(image_names(i).name,'hb17121.tif') &&
strcmp(sample_imgs(1).name,'2S1.jpg'))
    rotations(i) = -343.224854 + 90;
elseif (strcmp(image_names(i).name,'hb17299.tif') &&
strcmp(sample_imgs(1).name,'2S1.jpg'))
    rotations(i) = -47.224838 + 90;
elseif (strcmp(image_names(i).name,'hb17472.tif') &&
strcmp(sample_imgs(1).name,'2S1.jpg'))
    rotations(i) = -198.224838 + 90;
elseif (strcmp(image_names(i).name,'hb17512.tif') &&
strcmp(sample_imgs(1).name,'2S1.jpg'))
    rotations(i) = -310.224854 + 90;
elseif (strcmp(image_names(i).name,'hb19389.tif') &&
strcmp(sample_imgs(1).name,'BRDM_2.jpg')) % Set 9
    rotations(i) = -168.325 + 90;
elseif (strcmp(image_names(i).name,'hb19951.tif') &&
strcmp(sample_imgs(1).name,'BRDM_2.jpg'))
    rotations(i) = -147.325272 + 90;
elseif (strcmp(image_names(i).name,'hb19474.tif') &&
strcmp(sample_imgs(1).name,'BRDM_2.jpg'))
    rotations(i) = -29.325 + 90;
elseif (strcmp(image_names(i).name,'hb19379.tif') &&
strcmp(sample_imgs(1).name,'2S1.jpg')) % Set 10
    rotations(i) = -141.225 + 90;

```

```

elseif (strcmp(image_names(i).name, 'hb19520.tif') &&
strcmp(sample_imgs(1).name, '2S1.jpg'))
    rotations(i) = -147.225 + 90;
elseif (strcmp(image_names(i).name, 'hb19575.tif') &&
strcmp(sample_imgs(1).name, '2S1.jpg'))
    rotations(i) = -292.225 + 90;
elseif (strcmp(image_names(i).name, 'hb14951.tif') &&
strcmp(sample_imgs(1).name, '2S1.jpg')) % Set 11
    rotations(i) = -78.224838 + 90;
elseif (strcmp(image_names(i).name, 'hb14986.tif') &&
strcmp(sample_imgs(1).name, '2S1.jpg'))
    rotations(i) = -253.224838 + 90;
elseif (strcmp(image_names(i).name, 'hb14939.tif') &&
strcmp(sample_imgs(1).name, '2S1.jpg'))
    rotations(i) = -18.224838 + 90;
elseif (strcmp(image_names(i).name, 'hb20010.tif') &&
strcmp(sample_imgs(1).name, 'ZSU_23_4.jpg')) % Set 12
    rotations(i) = -107.993332 + 90;
elseif (strcmp(image_names(i).name, 'hb19528.tif') &&
strcmp(sample_imgs(1).name, 'ZSU_23_4.jpg'))
    rotations(i) = -168.993332 + 90;
elseif (strcmp(image_names(i).name, 'hb19933.tif') &&
strcmp(sample_imgs(1).name, 'ZSU_23_4.jpg'))
    rotations(i) = -56.993332 + 90;
elseif (strcmp(image_names(i).name, 'hb14965.tif') &&
strcmp(sample_imgs(1).name, 'ZSU_23_4.jpg')) % Set 13
    rotations(i) = -149.993332 + 90;
elseif (strcmp(image_names(i).name, 'hb15015.tif') &&
strcmp(sample_imgs(1).name, 'ZSU_23_4.jpg'))
    rotations(i) = -40.993332 + 90;
elseif (strcmp(image_names(i).name, 'hb15221.tif') &&
strcmp(sample_imgs(1).name, 'ZSU_23_4.jpg'))
    rotations(i) = -78.993332 + 90;
elseif (strcmp(image_names(i).name, 'hb19418.tif') &&
strcmp(sample_imgs(1).name, 'T62.jpg')) % Set 15
    rotations(i) = -244.515503 + 90;
elseif (strcmp(image_names(i).name, 'hb19790.tif') &&
strcmp(sample_imgs(1).name, 'T62.jpg'))
    rotations(i) = -31.515511 + 90;
elseif (strcmp(image_names(i).name, 'hb19931.tif') &&
strcmp(sample_imgs(1).name, 'T62.jpg'))
    rotations(i) = -48.515511 + 90;
elseif (strcmp(image_names(i).name, 'hb19390.tif') &&
strcmp(sample_imgs(1).name, 'T62.jpg')) % Set 16
    rotations(i) = -174.515503 + 90;
elseif (strcmp(image_names(i).name, 'hb19557.tif') &&
strcmp(sample_imgs(1).name, 'T62.jpg'))
    rotations(i) = -250.515503 + 90;
elseif (strcmp(image_names(i).name, 'hb19880.tif') &&
strcmp(sample_imgs(1).name, 'T62.jpg'))
    rotations(i) = -142.515503 + 90;
elseif (strcmp(image_names(i).name, 'hb19896.tif') &&
strcmp(sample_imgs(1).name, 'T62.jpg'))
    rotations(i) = -227.515503 + 90;
elseif (strcmp(image_names(i).name, 'hb03788.tif') &&
strcmp(sample_imgs(1).name, 'BTR_60.jpg')) % Set 18
    rotations(i) = -322.483337 + 90;
elseif (strcmp(image_names(i).name, 'hb03797.tif') &&
strcmp(sample_imgs(1).name, 'BTR_60.jpg'))
    rotations(i) = -52.483341 + 90;
elseif (strcmp(image_names(i).name, 'hb03991.tif') &&
strcmp(sample_imgs(1).name, 'BTR_60.jpg'))

```

```

        rotations(i) = -125.483337 + 90;
        elseif (strcmp(image_names(i).name, 'hb04030.tif') &&
strcmp(sample_imgs(1).name, 'BTR_60.jpg'))
            rotations(i) = -331.483337 + 90;
        elseif (strcmp(image_names(i).name, 'hb03395.tif') &&
strcmp(sample_imgs(1).name, 'BTR_60.jpg')) % Set 19
            rotations(i) = -186.483337 + 90;
        elseif (strcmp(image_names(i).name, 'hb04952.tif') &&
strcmp(sample_imgs(1).name, 'BTR_60.jpg'))
            rotations(i) = -48.483341 + 90;
        elseif (strcmp(image_names(i).name, 'hb05658.tif') &&
strcmp(sample_imgs(1).name, 'BTR_60.jpg'))
            rotations(i) = -107.483337 + 90;
        elseif (strcmp(image_names(i).name, 'hb05683.tif') &&
strcmp(sample_imgs(1).name, 'BTR_60.jpg'))
            rotations(i) = -253.483337 + 90;
        elseif (strcmp(image_names(i).name, 'hb19479.tif') &&
strcmp(sample_imgs(1).name, 'D7.jpg')) % Set 21
            rotations(i) = -161.307434 + 90;
        elseif (strcmp(image_names(i).name, 'hb19853.tif') &&
strcmp(sample_imgs(1).name, 'D7.jpg'))
            rotations(i) = -118.307442 + 90;
        elseif (strcmp(image_names(i).name, 'hb20001.tif') &&
strcmp(sample_imgs(1).name, 'D7.jpg'))
            rotations(i) = -170.307434 + 90;
    end

    % Rotate and resize the heights image and place into the matrix
    % of heights image slices
    object_slices(:, :, i) = imrotate(img_heights, -1*rotations(i), ...
        'nearest', 'crop');
    figure(i);
    subplot(2, 2, 4);
    imshow(object_slices(:, :, i));
    title('Heights Matrix');
end
end

% Take each 2D height image in the 3D objects matrix and combine them to
% form a final output 2D height image, which can be displayed in 3D

% Fuse the separate height slices into one 2D matrix of 3D heights
img_object3D = fuseSlices(object_slices);
% img_object3D = imerode(fuseSlices(object_slices), strel('square', 2));

% Length
fprintf('Length');
lengths = zeros(size(img_object3D, 1));
for r = 1:size(img_object3D, 1)
    % lengths(r) = numel(nonzeros(img_object3D(r, :)));
    mincol = 0;
    maxcol = 0;
    for c = 1:size(img_object3D, 2)
        if (mincol == 0 && img_object3D(r, c) > 0)
            mincol = c;
        end
        if (img_object3D(r, c) > 0)
            maxcol = c;
        end
    end
    lengths(r) = maxcol - mincol;
end
end

```

```

(max(max(lengths)) - 1) * 0.203125 + 0.3047;
(mean(mean(nonzeros(lengths))) - 1) * 0.203125 + 0.3047;
(median(median(nonzeros(lengths))) - 1) * 0.203125 + 0.3047

% Width
fprintf('Width');
widths = zeros(size(img_object3D,2));
for c = 1:size(img_object3D,2)
    % widths(c) = numel(nonzeros(img_object3D(:,c)));
    minrow = 0;
    maxrow = 0;
    for r = 1:size(img_object3D,1)
        if (minrow == 0 && img_object3D(r,c) > 0)
            minrow = r;
        end
        if (img_object3D(r,c) > 0)
            maxrow = r;
        end
    end
    widths(c) = maxrow - minrow;
end
(max(max(widths)) - 1) * 0.202148 + 0.3047;
(mean(mean(nonzeros(widths))) - 1) * 0.202148 + 0.3047;
(median(median(nonzeros(widths))) - 1) * 0.202148 + 0.3047

% Height
fprintf('Height');
max(max(img_object3D)) * 0.3047;
mean(mean(nonzeros(img_object3D))) * 0.3047
(mean(mean(nonzeros(img_object3D))) - 1) * 0.203125 + 0.3047;
(median(median(nonzeros(img_object3D))) - 1) * 0.203125 + 0.3047;
median(median(nonzeros(img_object3D))) * 0.3047;

% Show 3D plot of the fused object
figure(numel(image_names) + 1);
clf('reset');
surfc(img_object3D);
set(gca,'XDir','reverse');
xlabel('Column');
ylabel('Row');
zlabel('Height');
title('3-D Representation');
view([171,82]);
% panView(gca,25,2);

% Show sample image
figure(numel(image_names) + 2);
clf('reset');
imshow(imread(strcat('..\..\SAR Data\3D Sets\',sub_dir, ...
    '\',sample_imgs(1).name)));

```

File: getObject.m

```

function [img_out,rotation] = getObject(img_in)
% Extracts a SAR target object from the input image
%
% img_in ==> Input SAR image
%

```



```

% img_out ==> Output binary mask image of the object

% Median filtering
img_filtered = double(medfilt2((img_in)) / 255);

% Threshold
img_bw = im2bw(img_filtered);

% Filter out noisy islands
labelmat = bwlabeln(img_bw);
stats = regionprops(labelmat, 'Area');
areas = sort([stats.Area], 2, 'ascend');
if (size(areas, 2) > 1)
    img_object = bwareaopen(img_bw, areas(size(areas, 2)-1));
else
    img_object = bwareaopen(img_bw, max([stats.Area]));
end

% Clean up a little bit
img_out = imfill(bwmorph(img_object, 'fill'), 'holes');

% Filter out and combine multiple blobs if necessary
labelmat = bwlabeln(img_out);
stats = regionprops(labelmat, 'Area', 'Orientation', 'Centroid', ...
    'MajorAxisLength', 'MinorAxisLength');

if (numel(stats) > 1)
    [maxArea, idx] = max([stats.Area]);
    keep = zeros(numel(stats), 1);
    keep(1) = idx;
    idx2 = 2;
    for i = 1:numel(stats)
        if (i ~= idx)
            if (blobsClose(labelmat, stats(idx).MajorAxisLength, ...
                [idx, i]))
                p1 = fliplr(stats(i).Centroid);
                p2 = fliplr(stats(idx).Centroid);
                if (abs((p1(1) - p2(1)) / (p1(2) - p2(2))) < 1)
                    keep(idx2) = i;
                    idx2 = idx2 + 1;
                end
            end
        end
    end

    for r = 1:size(img_out, 1)
        for c = 1:size(img_out, 2)
            if (isempty(find(keep(:) == labelmat(r, c), 1)))
                img_out(r, c) = 0;
            end
        end
    end
end

[maxArea, idx] = max([stats.Area]);
rotation = stats(idx).Orientation;

img_out = medfilt2(img_out);

```


File: getShadow.m

```
function [img_out] = getShadow(img_in,img_object)
% Extracts a SAR shadow from the input image
%
% img_in ==> Input SAR image
%
% img_out ==> Output binary mask image of the shadow

img_in = double(img_in) / 255;

img_mask = zeros(size(img_in,1),size(img_in,2));
for c = 1:size(img_object,2)
    use_col = 0;
    for r = 1:size(img_object,1)
        if (img_object(r,c) == 1)
            use_col = 1;
        end
    end

    if (use_col == 1)
        img_mask(:,c) = 1;
    end
end

% Threshold
call = @ (img,lev) im2bw(img,mean(mean(img)) * 0.45);
img_bw = not(roifilt2(img_in,img_mask,call));

% Filter out noisy islands
labelmat = bwlabeln(img_bw);
stats = regionprops(labelmat,'Area');
areas = sort([stats.Area],2,'ascend');
if (size(areas,2) > 1)
    img_shadow = bwareaopen(img_bw,areas(size(areas,2)-1));
else
    img_shadow = bwareaopen(img_bw,max([stats.Area]));
end

% Clean up a little bit
img_out = imfill(img_shadow,'holes');

% Filter out and combine multiple blobs if necessary
labelmat = bwlabeln(img_out);
stats = regionprops(labelmat,'Area','Centroid','MinorAxisLength');

if (numel(stats) > 1)
    [maxArea,idx] = max([stats.Area]);
    keep = zeros(numel(stats),1);
    keep(1) = idx;
    idx2 = 2;
    for i = 1:numel(stats)
        if (i ~= idx)
            if (blobsClose(labelmat,stats(idx).MinorAxisLength * 2, ...
                [idx,i]))
                p1 = fliplr(stats(i).Centroid);
                p2 = fliplr(stats(idx).Centroid);
                if (abs((p1(1) - p2(1)) / (p1(2) - p2(2))) < 1)
```

```

        keep(idx2) = i;
        idx2 = idx2 + 1;
    end
end
end
end

for r = 1:size(img_out,1)
    for c = 1:size(img_out,2)
        if (isempty(find(keep(:) == labelmat(r,c),1)))
            img_out(r,c) = 0;
        end
    end
end
end
end

img_out = medfilt2(img_out);

```

File: getSlice.m

```

function [out] = getSlice(img_object,img_shadow,angle)
% Computes, based on the viewing angle of the plane, the height of the
% target object in every column
%
% img_object ==> Input object mask image.
% img_shadow ==> Input shadow mask image.
% angle ==> Angle of the radar view with respect to the ground plane.
%
% out ==> Output heights matrix.

out = zeros(size(img_object,1),size(img_object,2));

for c = 1:size(img_object,2)
    object_edge = 0;
    for r = 1:size(img_object,1)
        if (img_object(r,c) == 1 && object_edge == 0)
            object_edge = r;
        end
    end
    if (object_edge ~= 0)
        shadow_edge = 0;
        for r = 1:size(img_shadow,1)
            if (img_shadow(r,c) == 1 && shadow_edge == 0)
                shadow_edge = r;
            end
        end
        if (shadow_edge ~= 0)
            out(:,c) = (object_edge - shadow_edge)
                ... * tan(deg2rad(angle));
        end
    end
end
end

out = immultiply(out,img_object);

```

File: fuseSlices.m

```
function [img_out] = fuseSlices(slices)
% Fuse multiple height slices of an object into a single 3D height
% representation of the object
%
% slices ==> Semi-3D object height slices.
%
% out ==> 2D output matrix containing the 3D heights of the target
%         object.

% Pre-allocate the output matrix
img_out = zeros(size(slices,1),size(slices,2));

for r = 1:size(slices,1)
    for c = 1:size(slices,2)
        if (isempty(nonzeros(slices(r,c,:))))
            img_out(r,c) = 0;
        else
            img_out(r,c) = min(min(nonzeros(slices(r,c,:))));
        end
    end
end
end
```

File: doHough.m

```
function [lines,theta,rho,peaks] = doHough(img,num_lines,figure_num)
% Returns the lines vector after doing the Hough transform.
% Also plots the Hough lines in red.
%
% Inputs:
% img ==> The image to find lines in.
% num_lines ==> The maximum number of lines to return.
% figure_num ==> What figure number to plot the lines on.  WILL NOT
%                 PLOT if this number is less than 0.
%
% Outputs:
% lines ==> Is a struct vector containing the endpoints.

[H,theta,rho] = hough(img);
peaks = houghpeaks(H,num_lines,'Threshold',ceil(0.25*max(H(:))));
lines = houghlines(img,theta,rho,peaks,'MinLength',10);

if (figure_num >= 0)
    % Plot the figure
    figure(figure_num);
    imshow(img);
    hold all;
    for i = 1:numel(lines)
        x1 = lines(i).point1(1);
        y1 = lines(i).point1(2);
        x2 = lines(i).point2(1);
        y2 = lines(i).point2(2);
        plot([x1 x2],[y1 y2],'Color','r','LineWidth', 2);
    end
end
```

```
        hold off;  
    end
```

File: blobsClose.m

```
function [close] = blobsClose(labelmat,dist,label_idx)  
% Determines if two of the blobs in the label matrix are within a certain  
% distance of one another  
%  
% labelmat ==> The input label matrix.  
% dist ==> The distance criteria.  
% label_idx ==> 2-element vector defining which two blobs to look at.  
%  
% close ==> 1 if the blobs are close, 0 otherwise.  
  
[r1,c1] = find(labelmat == label_idx(1));  
[r2,c2] = find(labelmat == label_idx(2));  
  
for i = 1:size(r1,1)  
    for j = 1:size(r2,1)  
        if (sqrt((r1(i) - r2(j))^2 + (c1(i) - c2(j))^2) < dist)  
            close = 1;  
            return;  
        end  
    end  
end  
close = 0;
```

Appendix E: Non-SAR Modeling Images

Test Image 1

Input and intermediate images:

Input Image	Step Two Output: First-Pass Hough Transform

Step Eight Output: Object Vertices	Step Nine Output: 2-D Object Recreation

Output image:

Test Image 2

Input and intermediate images:

Input Image	Step Two Output: First-Pass Hough Transform

Step Eight Output: Object Vertices	Step Nine Output: 2-D Object Recreation

Output image:

Test Image 3

Input and intermediate images:

Input Image	Step Two Output: First-Pass Hough Transform

Step Eight Output: Object Vertices	Step Nine Output: 2-D Object Recreation

Output image:

Test Image 4

Input and intermediate images:

Input Image	Step Two Output: First-Pass Hough Transform

Step Eight Output: Object Vertices	Step Nine Output: 2-D Object Recreation

Output image:

Test Image 5

Input and intermediate images:

Input Image	Step Two Output: First-Pass Hough Transform

Step Eight Output: Object Vertices	Step Nine Output: 2-D Object Recreation

Output image:

Test Image 6

Input and intermediate images:

Input Image	Step Two Output: First-Pass Hough Transform

Step Eight Output: Object Vertices	Step Nine Output: 2-D Object Recreation

Output image:

Test Image 7 (Failure)

This test image failed during Step Eight, due to a lack of visibility of the inner vertex.

Input and intermediate images:

Test Image 8 (Failure)

This test image failed during Step Nine, due to the lack of visibility of the inner vertex which caused an incorrect value to be extracted in Step Eight.

Input and intermediate images:

Appendix F: Non-SAR Modeling Code

Driver File: reg3D.m

```
function [] = reg3D(img_num)
% Attempt to extract 3D data from a single-object non-SAR image.  Reads
% in the image and passes it to the ordered helper methods which
% perform the algorithm's steps.

% Read in the image
img_in = imread(strcat('..\..\Non-SAR Data\',int2str(img_num),'.jpg'));
figure(1);
imshow(img_in);

% STEP 1
% Perform first-pass edge detection (get the border lines) and filter
% out unwanted objects
img_outeredges = edge(img_in,'canny',[0.01 0.55],1.2);
img_outeredges = filterEdges(img_outeredges);
figure(2);
imshow(img_outeredges);

% STEP 2
% Perform the first-pass Hough Transform
outerlines = doHough(img_outeredges,'matlab',6,1,3);

% STEP 3
% Extract the mask of the inner region and the unfiltered
% outer vertices of the object
img_mask = getMask(img_in,outerlines);
figure(4);
imshow(img_mask);

% STEP 4
% Perform second-pass edge detection (get the inner lines)
call = @ (img) edge(img,'canny',[0.01 0.2]);
img_inneredges = immultiply(roifilt2(img_in,img_mask,call), ...
    bwmorph(img_mask,'erode'));
figure(5);
imshow(img_inneredges);

% STEP 5
% Second-pass of Hough transform to find the inner edge lines
innerlines = doHough(img_inneredges,'matlab',3,0.5,6);

% STEP 6
% Combine both edge detection passes for a final edge image
img_edges = img_outeredges | img_inneredges;
figure(7);
imshow(img_edges)

% STEPS 7 & 8
% Find and filter all object vertices
[vertices,outervertices,innervertex] = getObjVertices(img_edges, ...
    outerlines,innerlines,8);

% STEP 9
% Use the vertices to redraw the image and collect a set of lines that
```

```

% can be used to describe the object
[img_object2D,objectlines2D] = recreateObj2D(...
    vertices,outervertices, ...
    innervertex,size(img_edges,1),size(img_edges,2),9);

% Print out the lines
objectlines2D

% STEP 10
% Find a 3D form of the image
[img_object3D,objectlines3D,pixelvol] = ...
    recreateObj3D(objectlines2D, ...
    size(img_edges,1),size(img_edges,2),10);

% Print out the lines
objectlines3D

```

File: filterEdges.m

```

function [img_out] = filterEdges(img_in)
% Filters out unwanted objects that aren't part of the main object in
% question.
%
% img_in ==> Input edge image
%
% img_out ==> Output edge image with only the main object remaining

[labelmat,num] = bwlabeln(img_in);
if (num == 1)
    img_out = img_in;
else
    stats = regionprops(labelmat,'Area');
    img_out = bwareaopen(img_in,max([stats.Area]));
end

```

File: doHough.m

```

function [lines,theta,rho,peaks] =
doHough(img,method,num_lines,theta_res,figure_num)
% Returns the lines vector after doing the Hough transform one of two
% ways. Also plots the Hough lines in red.
%
% img ==> The image to find lines in
% method ==> String describing which Hough method to use
% num_lines ==> The maximum number of lines to return
% theta_res ==> Theta resolution for Hough
% thresh_mult ==> The percentage of the maximum in the Hough matrix to
%                 multiply by in order to get the threshold for a peak
% figure_num ==> What figure number to plot the lines on. WILL NOT
%                 PLOT if this number is less than 0.

```

```

%
% lines ==> If MATLAB's Hough, is a struct vector containing the
%           endpoints. If my Hough, is a matrix containing theta and
%           rho information.

if (strcmp(method,'matlab'))
    % BUILT-IN HOUGH
    if (numel(nonzeros(img)) ~= 0)
        [H,theta,rho] = hough(img,'ThetaResolution',theta_res);
        peaks = ...
            houghpeaks(H,num_lines,'Threshold',ceil(0.25*max(H(:))));
        lines = houghlines(img,theta,rho,peaks);

        if (figure_num >= 0 && isfield(lines,'point1'))
            % Plot the figure
            figure(figure_num);
            imshow(img);
            hold all;
            for i = 1:numel(lines)
                x1 = lines(i).point1(1);
                y1 = lines(i).point1(2);
                x2 = lines(i).point2(1);
                y2 = lines(i).point2(2);
                plot([x1 x2],[y1 y2],'Color','r','LineWidth', 2);
            end
            hold off;
        end
    else
        lines = [];
    end
elseif (strcmp(method,'hupton'))
    % MY VERSION OF HOUGH
    [Y,X] = size(img);
    votes = hough2(img,Y,X/2);

    lines = getLines(votes,num_lines);

    if (figure_num >= 0)
        % Plot the figure
        figure(figure_num);
        imshow(img);
        hold all;
        xval = [1 : 1 : X];
        for i = 1:length(lines)
            rho = (lines(i,1) - Y/2);
            theta = lines(i,2) * (pi/(X/2));
            yval = (rho - sin(theta) * (xval - X/2))/cos(theta) + Y/2;
            % function is not the normal Hough function because MATLAB
            % uses rows,cols rather than X,Y
            plot(xval,yval,'r-','LineWidth',1);
        end
        hold off;
        theta = [];
        rho = [];
        peaks = [];
    end
end
end

```

File: getMask.m

```
function [mask] = getMask(img,lines_in)
% Returns a binary region-of-interest image the same size as the input
% where region defined by the endpoints in the lines vector are all 1s
% and all other pixels are 0s. Also returns the vertices that
% correspond to the outer vertices of the object.
%
% img ==> The input image, used for sizing the mask
% lines ==> The struct vector created by MATLAB's Hough routine
%
% mask ==> The binary region-of-interest image
% vertices ==> A matrix of the [r,c] values of the vertices used to
%             form the mask.

% preserve
lines = lines_in;

% preallocate for speed
c = zeros(1,numel(lines)*2);
r = zeros(1,numel(lines)*2);

% start somewhere
c(1) = lines(1).point1(1);
r(1) = lines(1).point1(2);
c(2) = lines(1).point2(1);
r(2) = lines(1).point2(2);

% reset to very far away so they won't be used again
[Y,X] = size(img);
lines(1).point1(1) = -1*X;
lines(1).point1(2) = -1*Y;
lines(1).point2(1) = -1*X;
lines(1).point2(2) = -1*Y;

for i = 3:numel(lines)*2
    distances = zeros(numel(lines)*2,3); % preallocate for speed

    for j = 1:numel(lines)
        distances(j,1) = sqrt((c(i-1)-lines(j).point1(1))^2 + ...
            (r(i-1)-lines(j).point1(2))^2);
        distances(j,2) = j;
        distances(j,3) = 1;

        distances(j+numel(lines),1) = ...
            sqrt((c(i-1)-lines(j).point2(1))^2 + ...
            (r(i-1)-lines(j).point2(2))^2);
        distances(j+numel(lines),2) = j;
        distances(j+numel(lines),3) = 2;
    end

    [dist,idx] = min(distances(:,1));
    if (distances(idx,3) == 1)
        c(i) = lines(distances(idx,2)).point1(1);
        r(i) = lines(distances(idx,2)).point1(2);
        lines(distances(idx,2)).point1(1) = -1*X;
        lines(distances(idx,2)).point1(2) = -1*Y;
    else
        c(i) = lines(distances(idx,2)).point2(1);
        r(i) = lines(distances(idx,2)).point2(2);
        lines(distances(idx,2)).point2(1) = -1*X;
        lines(distances(idx,2)).point2(2) = -1*Y;
    end
end
```

```

        lines(distances(idx,2)).point2(2) = -1*Y;
    end
end
mask = roipoly(img,c,r);

```

File: getObjVertices.m

```

function [vertices,outervertices,innervertex] = getObjVertices(img, ...
    outerlines,innerlines,figurenum)
% Finds the innervertex and filters the outervertices (by finding an
% actual arithmetic solution to the line interceptions). Combines all
% vertices into appropriate output matrices.
%
% img ==> The original edge image, to be used for plotting and sizing.
% mask ==> Mask image, originally used to find the outervertices and
%         used just in case only one inner line is found.
% outerlines ==> The outer Hough lines of the object.
% innerlines ==> The inner Hough lines of the object.
% figurenum ==> the figure number to plot the image and vertices on.
%
% vertices ==> a vector containing all unique vertices of the object
% outervertices ==> The outer vertices of the object after filtering.
% innervertex ==> The inner vertex of the object.

% Find the innervertex first
if (~isfield(innerlines,'point1'))
    innervertex = [];
    fprintf('No inner vertex found!\n');
    return;
end

% Check for lines too close together and remove the shorter one
numlines = numel(innerlines);
for i = 1:numel(innerlines)
    for j = i:numel(innerlines)
        theta1 = innerlines(i).theta;
        theta2 = innerlines(j).theta;

        filter = 0;
        if (i ~= j)
            if (abs(theta1 - theta2) <= 5)
                filter = 1;
            elseif ((theta1 > 85 && theta2 < -85) && ...
                (abs((90 - theta1) - (-90 - theta2)) <= 5))
                filter = 1;
            elseif ((theta2 > 85 && theta1 < -85) && ...
                (abs((90 - theta2) - (-90 - theta1)) <= 5))
                filter = 1;
            end
        end

        if (filter == 1)
            numlines = numlines - 1;

            r1 = innerlines(i).point1(2);
            c1 = innerlines(i).point1(1);

```

```

        r2 = innerlines(i).point2(2);
        c2 = innerlines(i).point2(1);
        disti = sqrt((r1 - r2)^2 + (c1 - c2)^2);

        r1 = innerlines(j).point1(2);
        c1 = innerlines(j).point1(1);
        r2 = innerlines(j).point2(2);
        c2 = innerlines(j).point2(1);
        distj = sqrt((r1 - r2)^2 + (c1 - c2)^2);

        % Move the bad line way out of the image
        if (disti > distj)
            innerlines(j).point1(1) = -1*size(img,2);
            innerlines(j).point1(2) = -1*size(img,1);
            innerlines(j).point2(1) = -2*size(img,2);
            innerlines(j).point2(2) = -2*size(img,1);
        else
            innerlines(i).point1(1) = -1*size(img,2);
            innerlines(i).point1(2) = -1*size(img,1);
            innerlines(i).point2(1) = -2*size(img,2);
            innerlines(i).point2(2) = -2*size(img,1);
        end
    end
end

lines = zeros(numlines,4);
for i = 1:numel(innerlines)
    if (innerlines(i).point1(1) > 0)
        lines(i,1) = innerlines(i).point1(2);
        lines(i,2) = innerlines(i).point1(1);
        lines(i,3) = innerlines(i).point2(2);
        lines(i,4) = innerlines(i).point2(1);
    end
end

if (numlines == 1)
    % Only one line, add both points to the innervertex vector and sort
    % out which one is correct later
    innervertex(1,1) = lines(1,1);
    innervertex(1,2) = lines(1,2);
    innervertex(2,1) = lines(1,3);
    innervertex(2,2) = lines(1,4);
elseif (numlines == 2)
    % Two lines found, just find their intersection
    m1 = (lines(1,3) - lines(1,1)) / (lines(1,4) - lines(1,2));
    b1 = lines(1,1) - m1*lines(1,2);
    m2 = (lines(2,3) - lines(2,1)) / (lines(2,4) - lines(2,2));
    b2 = lines(2,1) - m2*lines(2,2);
    innervertex(2) = (b2 - b1) / (m1 - m2);
    innervertex(1) = m1*innervertex(2) + b1;
else
    % Three lines found, find all three possible intersections, if two
    % or more agree, use that point; if none agree, average them
    solutions = zeros(3,2);

    m1 = (lines(1,3) - lines(1,1)) / (lines(1,4) - lines(1,2));
    b1 = lines(1,1) - m1*lines(1,2);
    m2 = (lines(2,3) - lines(2,1)) / (lines(2,4) - lines(2,2));
    b2 = lines(2,1) - m2*lines(2,2);
    m3 = (lines(3,3) - lines(3,1)) / (lines(3,4) - lines(3,2));
    b3 = lines(3,1) - m3*lines(3,2);

```



```

solutions(1,2) = (b2 - b1) / (m1 - m2);
solutions(1,1) = m1*solutions(1,2) + b1;
solutions(2,2) = (b3 - b1) / (m1 - m3);
solutions(2,1) = m1*solutions(2,2) + b1;
solutions(3,2) = (b3 - b2) / (m2 - m3);
solutions(3,1) = m2*solutions(3,2) + b2;

if (solutions(1,1) == solutions(2,1) ...
    && solutions(1,2) == solutions(2,2))
    innervertex(1) = solutions(1,1);
    innervertex(2) = solutions(1,2);
elseif (solutions(1,1) == solutions(3,1) ...
    && solutions(1,2) == solutions(3,2))
    innervertex(1) = solutions(1,1);
    innervertex(2) = solutions(1,2);
elseif (solutions(2,1) == solutions(3,1) ...
    && solutions(2,2) == solutions(3,2))
    innervertex(1) = solutions(2,1);
    innervertex(2) = solutions(2,2);
else
    innervertex(1) = (solutions(1,1) + solutions(2,1) + ...
        solutions(3,1)) / 3;
    innervertex(2) = (solutions(1,2) + solutions(2,2) + ...
        solutions(3,2)) / 3;
end
end

% Now find the rest of the vertices
points = zeros(numel(outerlines)*2,4);
lengths = zeros(numel(outerlines),1);
for i = 1:numel(outerlines)
    % Place all lines' points in the points vector
    points(i,1) = outerlines(i).point1(2);
    points(i,2) = outerlines(i).point1(1);
    points(i+numel(outerlines),1) = outerlines(i).point2(2);
    points(i+numel(outerlines),2) = outerlines(i).point2(1);

    % Place the lines' other endpoints in the vector, for use later
    points(i,3) = outerlines(i).point2(2);
    points(i,4) = outerlines(i).point2(1);
    points(i+numel(outerlines),3) = outerlines(i).point1(2);
    points(i+numel(outerlines),4) = outerlines(i).point1(1);

    % Get the length of the line (for use in determining if a solution
    % is valid later on)
    lengths(i) = sqrt((points(i,1) - points(i,3))^2 + ...
        (points(i,2) - points(i,4))^2);
end

% Find all distances for vertex detection
distances = zeros(size(points,1),3);
idx = 1;
for i = 1:size(points,1)
    r1 = points(i,1);
    c1 = points(i,2);
    for j = i+1:size(points,1)
        r2 = points(j,1);
        c2 = points(j,2);
        distances(idx,1) = sqrt((c1 - c2)^2 + (r1 - r2)^2);
        distances(idx,2) = i;
        distances(idx,3) = j;
    end
end

```

```

        idx = idx + 1;
    end
end

% Find the vertices
outervertices = zeros(6,2);
idx = 1;

i = 1;
while (i <= numel(outerlines))
    [dist,idx_min] = min(distances(:,1));

    % If the two lines closest together are actually very far apart,
    % then their algebraic solution is most likely NOT an actual vertex
    % (they're not adjacent lines in the image)
    if (dist > 0.15 * max(lengths))
        break;
    end

    idx_i = distances(idx_min,2);
    idx_j = distances(idx_min,3);

    m1 = (points(idx_i,3) - points(idx_i,1)) ...
        / (points(idx_i,4) - points(idx_i,2));
    b1 = points(idx_i,1) - m1*points(idx_i,2);
    m2 = (points(idx_j,3) - points(idx_j,1)) ...
        / (points(idx_j,4) - points(idx_j,2));
    b2 = points(idx_j,1) - m2*points(idx_j,2);

    % Make sure we don't find the same set again
    distances(idx_min,1) = max(distances(:,1));

    if (m1 == m2 && b1 == b2)
        % Lines are the same, points must have been, too
        i = i - 1;
    else
        % Lines are different, find solution
        outervertices(idx,2) = (b2 - b1) / (m1 - m2);
        outervertices(idx,1) = m1*outervertices(idx,2) + b1;
        idx = idx + 1;

        % Make the used points negative so we can tell later which
        % points were NOT used
        points(idx_i,1) = -1 * points(idx_i,1);
        points(idx_i,2) = -1 * points(idx_i,2);
        points(idx_j,1) = -1 * points(idx_j,1);
        points(idx_j,2) = -1 * points(idx_j,2);
    end

    i = i + 1;
end

% Go back and find all points not used and use them
for i = 1:size(points,1)
    if (points(i,1) > 0 && points(i,2) > 0)
        % Point unused, is an outervertex
        outervertices(idx,1) = points(i,1);
        outervertices(idx,2) = points(i,2);
        idx = idx + 1;
    end
end
end

```

```

% If the innervertex had two points, figure out which one is correct now
if (size(innervertex,1) == 2)
    distances = zeros(size(outervertices,1),2);
    for i = 1:size(outervertices,1)
        distances(i,1) = sqrt((innervertex(1,1) - ...
            outervertices(i,1))^2 ...
            + (innervertex(1,2) - outervertices(i,2))^2);
        distances(i,2) = sqrt((innervertex(2,1) - ...
            outervertices(i,1))^2 ...
            + (innervertex(2,2) - outervertices(i,2))^2);
    end

    dist1 = min(distances(:,1));
    dist2 = min(distances(:,2));

    % Select the point that is NOT closest to one of the outer vertices
    if (dist1 < dist2)
        innervertex(1,:) = [];
    else
        innervertex(2,:) = [];
    end
end

vertices = outervertices;
vertices(size(vertices,1)+1,1) = innervertex(1);
vertices(size(vertices,1)+2,2) = innervertex(2);

% Plot the vertices
figure(figurenum);
imshow(img);
hold all;
for i = 1:size(outervertices,1)
    plot(outervertices(i,2),outervertices(i,1),'bx','MarkerSize', ...
        16,'LineWidth',2);
end

plot(innervertex(2),innervertex(1),'gx','MarkerSize',16,'LineWidth',2);
hold off;

```

File: recreateObj2D.m

```

function [img,lines] = recreateObj2D(vertices,outervertices, ...
    innervertex,Y,X,figurenum)
% Recreates the object in a new image with new lines to define the
% object in a 2D environment
%
% vertices ==> Complete vertex set.
% outervertices ==> Vertices on the outer edge of the object.
% innervertex ==> Inner vertex of the object.
% Y ==> Row size of the object.
% X ==> Col size of the object.
% figurenum ==> Figure number to plot the image on.
%
% img ==> Output image with the object redrawn.
% lines ==> 4-column matrix containing the endpoints of all the defined
%           lines in the object. IMPORTANT: the last 3 rows of this
%           matrix must be the three lines protruding from the

```

```

%           innervertex, with the first 2 columns in each of those rows
%           being the [r,c] values of the innervertex itself.

% Prepare line arrays
lines = zeros(size(outervertices,1)+3,4);
iv_oddset = zeros(3,2);
iv_evenset = zeros(3,2);

idx = 1;
for i = 1:size(outervertices,1)
    lineset = zeros(size(outervertices,1)-1,3);
    idx2 = 1;

    d_y = -1*(innervertex(1) - outervertices(idx,1));
    d_x = innervertex(2) - outervertices(idx,2);
    base_angle = rad2deg(atan2(d_y,d_x));
    for j = 1:size(outervertices,1)
        if (j ~= idx)
            lineset(idx2,1) = outervertices(j,1);
            lineset(idx2,2) = outervertices(j,2);

            % Now get the angle with respect to the line towards the
            % innervertex (basically get the angle towards the other
            % point as if the object were rotated so that the line
            % towards the innervertex was directly to the right of the
            % current vertex, creating a new coordinate system)
            d_y = -1*(lineset(idx2,1) - outervertices(idx,1));
            d_x = lineset(idx2,2) - outervertices(idx,2);

            angle = rad2deg(atan2(d_y,d_x)) - base_angle;
            if (angle > 180)
                angle = angle - 360;
            elseif (angle < -180)
                angle = angle + 360;
            end
            lineset(idx2,3) = angle;

            idx2 = idx2 + 1;
        end
    end

    % Find and add the line with the maximum angle (could use min) to
    % the outerlines output array
    lines(i,1) = outervertices(idx,1);
    lines(i,2) = outervertices(idx,2);
    [max_theta,idx2] = max(lineset(:,3));

    lines(i,3) = lineset(idx2,1);
    lines(i,4) = lineset(idx2,2);

    % Add a line to one of the innervertex (iv) arrays
    if (rem(i,2) == 0)
        iv_evenset(i/2,1) = lines(i,3);
        iv_evenset(i/2,2) = lines(i,4);
    else
        iv_oddset(ceil(i/2),1) = lines(i,3);
        iv_oddset(ceil(i/2),2) = lines(i,4);
    end

    % Now find where we went to, and that's the idx of the next
    % iteration
    idx = find(outervertices(:,1) == lines(i,3) & ...

```

```

        outervertices(:,2) == lines(i,4));
end

% Determine which set of innervertex (iv) line sets is the correct one

% Create squared-error vectors
iv_odd_err = zeros(size(outervertices,1),1);
iv_even_err = zeros(size(outervertices,1),1);

% Start looping through the two iv line sets and calculate squared
% errors
for i = 1:3
    % Even set
    d_y = -1*(innervertex(1) - iv_evenset(i,1));
    d_x = innervertex(2) - iv_evenset(i,2);
    base_angle = rad2deg(atan2(d_y,d_x));

    % Find the line that starts at the current outer point, and we can
    % reference the other three lines-of-interest from there (one up,
    % one down, and two down)
    idx = find(lines(:,1) == iv_evenset(i,1) & ...
        lines(:,2) == iv_evenset(i,2));

    % Calculate squared errors
    len = size(outervertices,1);
    d_y = -1*(lines(rem(idx,len)+1,3) - lines(rem(idx,len)+1,1));
    d_x = lines(rem(idx,len)+1,4) - lines(rem(idx,len)+1,2);
    iv_even_err(i) = (rad2deg(atan2(d_y,d_x)) - base_angle)^2;

    d_y = -1*(lines(rem(rem(idx,len)+3,len)+1,1) - ...
        lines(rem(rem(idx,len)+3,len)+1,3));
    d_x = lines(rem(rem(idx,len)+3,len)+1,2) - ...
        lines(rem(rem(idx,len)+3,len)+1,4);
    iv_even_err(i*2) = (rad2deg(atan2(d_y,d_x)) - base_angle)^2;

    % Repeat for Odd set
    d_y = -1*(innervertex(1) - iv_oddset(i,1));
    d_x = innervertex(2) - iv_oddset(i,2);
    base_angle = rad2deg(atan2(d_y,d_x));

    idx = find(lines(:,1) == iv_oddset(i,1) & ...
        lines(:,2) == iv_oddset(i,2));

    d_y = -1*(lines(rem(idx,len)+1,3) - lines(rem(idx,len)+1,1));
    d_x = lines(rem(idx,len)+1,4) - lines(rem(idx,len)+1,2);
    iv_odd_err(i) = (rad2deg(atan2(d_y,d_x)) - base_angle)^2;

    d_y = -1*(lines(rem(rem(idx,len)+3,len)+1,1) - ...
        lines(rem(rem(idx,len)+3,len)+1,3));
    d_x = lines(rem(rem(idx,len)+3,len)+1,2) - ...
        lines(rem(rem(idx,len)+3,len)+1,4);
    iv_odd_err(i*2) = (rad2deg(atan2(d_y,d_x)) - base_angle)^2;
end

len = size(lines,1);
lines(len,1) = innervertex(1);
lines(len,2) = innervertex(2);
lines(len-1,1) = innervertex(1);
lines(len-1,2) = innervertex(2);
lines(len-2,1) = innervertex(1);

```

```

lines(len-2,2) = innervertex(2);
if (mean(iv_odd_err) < mean(iv_even_err))
    lines(len,3) = iv_oddset(1,1);
    lines(len,4) = iv_oddset(1,2);
    lines(len-1,3) = iv_oddset(2,1);
    lines(len-1,4) = iv_oddset(2,2);
    lines(len-2,3) = iv_oddset(3,1);
    lines(len-2,4) = iv_oddset(3,2);
else
    lines(len,3) = iv_evenset(1,1);
    lines(len,4) = iv_evenset(1,2);
    lines(len-1,3) = iv_evenset(2,1);
    lines(len-1,4) = iv_evenset(2,2);
    lines(len-2,3) = iv_evenset(3,1);
    lines(len-2,4) = iv_evenset(3,2);
end

img = zeros(Y,X);
[idx,labelmat] = drawline(lines(:,1:2),lines(:,3:4),[Y X]);
% Drawline ==> 3rd Party app
% Credits:
% Daniel Simoes Lopes
% ICIST
% Instituto Superior Tecnico - Universidade Tecnica de Lisboa
% danlopes (at) civil ist utl pt
% http://www.civil.ist.utl.pt/~danlopes
%
% June 2007 original version.

img(idx) = labelmat;

figure(figurenum);
imshow(img);

% Replot the lines for the sake of viewing/rotating in MATLAB
hold all;
for i=1:size(lines,1)
    plot([lines(i,2) lines(i,4)],[lines(i,1) lines(i,3)], ...
        'Color','r','LineWidth', 2);
end
hold off;

```

File: recreateObj3D.m

```

function [img,lines,vol] = recreateObj3D(objectlines2D,Y,X,figurenum)
% Recreates the object in a new image with new lines to define the
% object in a 3D environment
%
% objectlines2D ==> Complete line set describing the object in 2D. The
%                     last three elements in this vector are the three
%                     inner lines of the object
% innervertex ==> Inner vertex of the object.
% Y ==> Row size of the object.
% X ==> Col size of the object.
% figurenum ==> Figure number to plot the image on.
%
% img ==> Output image with the object drawn in 3D
% lines ==> 6-column matrix containing the endpoints of all the defined

```

```

%           lines in the object.

% Calculate formula constants used in solving for the depths of each
% endpoint of the three lines protruding from the innervertex
len = size(objectlines2D,1);
const12 = abs((objectlines2D(len,3) - objectlines2D(len,1)) ...
    * (objectlines2D(len-1,3) - objectlines2D(len-1,1)) ...
    + (objectlines2D(len,4) - objectlines2D(len,2)) ...
    * (objectlines2D(len-1,4) - objectlines2D(len-1,2)));

const13 = abs((objectlines2D(len,3) - objectlines2D(len,1)) ...
    * (objectlines2D(len-2,3) - objectlines2D(len-2,1)) ...
    + (objectlines2D(len,4) - objectlines2D(len,2)) ...
    * (objectlines2D(len-2,4) - objectlines2D(len-2,2)));

const23 = abs((objectlines2D(len-1,3) - objectlines2D(len-1,1)) ...
    * (objectlines2D(len-2,3) - objectlines2D(len-2,1)) ...
    + (objectlines2D(len-1,4) - objectlines2D(len-1,2)) ...
    * (objectlines2D(len-2,4) - objectlines2D(len-2,2)));

% Compute the depths (adjusted to move into the matrix by 1 pixel)
d1 = sqrt(const12*const13/const23) + 1;
d3 = const13/(d1 - 1) + 1;
d2 = const23/(d3 - 1) + 1;

% Compute the required size of the third dimension
Z = round(d1 + d2 + d3) + 1;

% Form the lines vector for output and object recreation
lines = zeros(12,6);
for i = 1:len
    % The z-value for (i,3) could be set to anything here, but setting
    % the forward-most point in the 3D image to 1 ensures minimal size
    % for the 3D image matrix
    if (i == len)
        lines(12,1:2) = objectlines2D(i,1:2);
        lines(12,4:5) = objectlines2D(i,3:4);
        lines(12,3) = 1;
        lines(12,6) = d1;
    elseif (i == len-1)
        lines(11,1:2) = objectlines2D(i,1:2);
        lines(11,4:5) = objectlines2D(i,3:4);
        lines(11,3) = 1;
        lines(11,6) = d2;
    elseif (i == len-2)
        lines(10,1:2) = objectlines2D(i,1:2);
        lines(10,4:5) = objectlines2D(i,3:4);
        lines(10,3) = 1;
        lines(10,6) = d3;
    end
end

len = 12;
idx = 1;
vecs = zeros(3,3);
for i = 0:2
    vecs(i+1,1:3) = lines(len-i,4:6) - lines(len-i,1:3);

    lines(idx,1:3) = lines(len-rem(i+1,3),4:6);
    lines(idx,4:6) = lines(len-rem(i+1,3),4:6) + vecs(i+1,1:3);
    lines(idx+1,1:3) = lines(len-rem(i+2,3),4:6);
    lines(idx+1,4:6) = lines(len-rem(i+2,3),4:6) + vecs(i+1,1:3);

```

```

    idx = idx + 2;
end

for i = 1:3
    pt = lines(i,4:6);
    vec1 = pt(1:3) - lines(i,1:3);
    vec2 = pt(1:3) - lines(i+3,1:3);
    % Take advantage of the periodic feature of the lines array as
    % formed above

    for j = 1:3
        if (vecs(j,1:3) ~= vec1 & vecs(j,1:3) ~= vec2)
            break;
        end
    end

    lines(idx,1:3) = pt;
    lines(idx,4:6) = pt + vecs(j,1:3);
    idx = idx + 1;
end

% Compute the length of each dimension of the 3D object
dim1 = sqrt((lines(len,1) - lines(len,4))^2 + (lines(len,2) - ...
    lines(len,5))^2 + (lines(len,3) - lines(len,6))^2);
dim2 = sqrt((lines(len-1,1) - lines(len-1,4))^2 + (lines(len-1,2) - ...
    lines(len-1,5))^2 + (lines(len-1,3) - lines(len-1,6))^2);
dim3 = sqrt((lines(len-2,1) - lines(len-2,4))^2 + (lines(len-2,2) - ...
    lines(len-2,5))^2 + (lines(len-2,3) - lines(len-2,6))^2);
vol = dim1 * dim2 * dim3;

% Create the image matrix
% NOTE: cannot create helper function for this since it would require
% creating an instantaneous temporary variable of the 3D image matrix -->
% very big waste of running memory!
% ==> Bresenham's Line Algorithm for 3-D Lines
img = zeros(Y,X,Z);

for i = 1:size(lines,1)
    pixel(1) = round(lines(i,1));
    pixel(2) = round(lines(i,2));
    pixel(3) = round(lines(i,3));
    dx = round(lines(i,4)) - round(lines(i,1));
    dy = round(lines(i,5)) - round(lines(i,2));
    dz = round(lines(i,6)) - round(lines(i,3));

    if (dx < 0)
        x_inc = -1;
    else
        x_inc = 1;
    end
    l = abs(dx);

    if (dy < 0)
        y_inc = -1 ;
    else
        y_inc = 1;
    end
    m = abs(dy);

    if (dz < 0)
        z_inc = -1 ;

```



```

else
    z_inc = 1;
end
n = abs(dz);

dx2 = 2^l;
dy2 = 2^m;
dz2 = 2^n;

if ((l >= m) && (l >= n))
    err_1 = dy2 - 1;
    err_2 = dz2 - 1;
    for i = 1:l
        img(pixel(1),pixel(2),pixel(3)) = 1;
        if (err_1 > 0)
            pixel(2)= pixel(2) + y_inc;
            err_1 = err_1 - dx2;
        end
        if (err_2 > 0)
            pixel(3) = pixel(3) + z_inc;
            err_2 = err_2 - dx2;
        end
        err_1 = err_1 + dy2;
        err_2 = err_2 + dz2;
        pixel(1) = pixel(1) + x_inc;
    end
elseif ((m >= l) && (m >= n))
    err_1 = dx2 - m;
    err_2 = dz2 - m;
    for i = 1:m
        img(pixel(1),pixel(2),pixel(3)) = 1;
        if (err_1 > 0)
            pixel(1) = pixel(1) + x_inc;
            err_1 = err_1 - dy2;
        end
        if (err_2 > 0)
            pixel(3) = pixel(3) + z_inc;
            err_2 = err_2 - dy2;
        end
        err_1 = err_1 + dx2;
        err_2 = err_2 + dz2;
        pixel(2) = pixel(2) + y_inc;
    end
else
    err_1 = dy2 - n;
    err_2 = dx2 - n;
    for i = 1:n
        img(pixel(1),pixel(2),pixel(3))=1;
        if (err_1 > 0)
            pixel(2) = pixel(2) + y_inc;
            err_1 = err_1 - dz2;
        end
        if (err_2 > 0)
            pixel(1) = pixel(1) + x_inc;
            err_2 = err_2 - dz2;
        end
        err_1 = err_1 + dy2;
        err_2 = err_2 + dx2;
        pixel(3) = pixel(3) + z_inc;
    end
    img(pixel(1),pixel(2),pixel(3))=1;
end
end

```

```

end

% Draw the lines in a new window
innervertex(1) = lines(size(lines,1),1);
innervertex(2) = lines(size(lines,1),2);
pt1(1) = lines(size(lines,1),4);
pt1(2) = lines(size(lines,1),5);
pt1(3) = lines(size(lines,1),6);
pt2(1) = lines(size(lines,1)-1,4);
pt2(2) = lines(size(lines,1)-1,5);
pt2(3) = lines(size(lines,1)-1,6);
pt3(1) = lines(size(lines,1)-2,4);
pt3(2) = lines(size(lines,1)-2,5);
pt3(3) = lines(size(lines,1)-2,6);

figure(figurenum);
clf('reset');
whitebg(figurenum,[0 0 0]);

dist = max([dim1 dim2 dim3]);

width = 0.4;
height = width * (Y/X);
depth = width * (Z/X);

% Front perspective
subplot('Position',[0.5,0.1,width,height]);
for i = 1:size(lines,1)
    line([lines(i,2) lines(i,5)], [lines(i,1) lines(i,4)], ...
        [lines(i,3) lines(i,6)], 'Color', 'w');
end
hold all;
plot(innervertex(2),innervertex(1), 'gx', 'MarkerSize',10, 'LineWidth',2);
plot3(pt1(2),pt1(1),pt1(3), 'bx', 'MarkerSize',10, 'LineWidth',2);
plot3(pt2(2),pt2(1),pt2(3), 'rx', 'MarkerSize',10, 'LineWidth',2);
plot3(pt3(2),pt3(1),pt3(3), 'yx', 'MarkerSize',10, 'LineWidth',2);
axis([0 X 0 Y 0 Z]);
set(gca, 'DataAspectRatio',[1 1 1]);
set(gca, 'CameraTarget',[X/2,Y/2,0]);
set(gca, 'CameraPosition',[X/2,Y/2,-dist]);
set(gca, 'CameraUpVector',[0 -1 0]);
set(gca, 'TickDir','out');

% Left Perspective
subplot('Position',[0.3,0.1,depth,height]);
for i = 1:size(lines,1)
    line([lines(i,2) lines(i,5)], [lines(i,1) lines(i,4)], ...
        [lines(i,3) lines(i,6)], 'Color', 'w');
end
hold all;
plot(innervertex(2),innervertex(1), 'gx', 'MarkerSize',10, 'LineWidth',2);
plot3(pt1(2),pt1(1),pt1(3), 'bx', 'MarkerSize',10, 'LineWidth',2);
plot3(pt2(2),pt2(1),pt2(3), 'rx', 'MarkerSize',10, 'LineWidth',2);
plot3(pt3(2),pt3(1),pt3(3), 'yx', 'MarkerSize',10, 'LineWidth',2);
axis([0 X 0 Y 0 Z]);
set(gca, 'DataAspectRatio',[1 1 1]);
set(gca, 'CameraTarget',[0,Y/2,Z/2]);
set(gca, 'CameraPosition',[-dist,Y/2,Z/2]);
set(gca, 'CameraUpVector',[0 -1 0]);
set(gca, 'xtick',[], 'ytick',[]);

% Top Perspective

```

```

subplot('Position',[0.5,0.6,width,depth]);
for i = 1:size(lines,1)
    line([lines(i,2) lines(i,5)], [lines(i,1) lines(i,4)], ...
        [lines(i,3) lines(i,6)], 'Color', 'w');
end
hold all;
plot(innervertex(2),innervertex(1), 'gx', 'MarkerSize',10, 'LineWidth',2);
plot3(pt1(2),pt1(1),pt1(3), 'bx', 'MarkerSize',10, 'LineWidth',2);
plot3(pt2(2),pt2(1),pt2(3), 'rx', 'MarkerSize',10, 'LineWidth',2);
plot3(pt3(2),pt3(1),pt3(3), 'yx', 'MarkerSize',10, 'LineWidth',2);
axis([0 X 0 Y 0 Z]);
set(gca, 'DataAspectRatio',[1 1 1]);
set(gca, 'CameraTarget', [X/2,0,Z/2]);
set(gca, 'CameraPosition', [X/2,-dist,Z/2]);
set(gca, 'CameraUpVector', [0 0 1]);
set(gca, 'xtick', [], 'ytick', [], 'ztick', []);

% 3D Perspective
subplot('Position',[0.05,0.55,width,height]);
for i = 1:size(lines,1)
    line([lines(i,2) lines(i,5)], [lines(i,1) lines(i,4)], ...
        [lines(i,3) lines(i,6)], 'Color', 'w');
end
hold all;
plot(innervertex(2),innervertex(1), 'gx', 'MarkerSize',10, 'LineWidth',2);
plot3(pt1(2),pt1(1),pt1(3), 'bx', 'MarkerSize',10, 'LineWidth',2);
plot3(pt2(2),pt2(1),pt2(3), 'rx', 'MarkerSize',10, 'LineWidth',2);
plot3(pt3(2),pt3(1),pt3(3), 'yx', 'MarkerSize',10, 'LineWidth',2);
axis([0 X 0 Y 0 Z]);
set(gca, 'DataAspectRatio',[1 1 1]);
set(gca, 'CameraTarget', [X/2,Y/2,Z/2]);
set(gca, 'CameraPosition', [0,0,-dist]);
set(gca, 'CameraUpVector', [0 -1 0]);
set(gca, 'xtick', [], 'ytick', [], 'ztick', []);
panView(gca,25,2,[0,0,-dist],2);

```

File: panView.m

```

function [] = panView(varargin)
% Pans around a 3D image.
%
% Possible Inputs (as part of varargin):
% figurenum ==> The figure number to pan around.
% numRotations ==> The number of rotations to do (default is 5).
% panSpeed ==> The relative speed of the pan.
% camStartPos ==> Vector [x,y,z] position to start/end the camera at.
% constDim ==> The dimension (1,2, or 3) to keep constant during pan.
%
% The input defaults will cause 5 rotations around the current figure
% using the current camera position as the start and end point, and
% keeping the z dimension constant.

% Set defaults and retrieve inputs
figurenum = gca;
numRotations = 5;
panSpeed = 1;
camStartPos = get(gca, 'CameraPosition');

```

```

constDim = 3;

if (numel(varargin) == 1)
    figurenum = varargin{1};
elseif (numel(varargin) == 2)
    figurenum = varargin{1};
    numRotations = varargin{2};
elseif (numel(varargin) == 3)
    figurenum = varargin{1};
    numRotations = varargin{2};
    panSpeed = varargin{3};
elseif (numel(varargin) == 4)
    figurenum = varargin{1};
    numRotations = varargin{2};
    panSpeed = varargin{3};
    camStartPos = varargin{4};
elseif (numel(varargin) == 5)
    figurenum = varargin{1};
    numRotations = varargin{2};
    panSpeed = varargin{3};
    camStartPos = varargin{4};
    constDim = varargin{5};
end

% Enable user termination
command_new = 'set(gcf, 'UserData', true)';
set(gcf, 'UserData', false);
set(gca, 'ButtonDownFcn', command_new);
set(gcf, 'ButtonDownFcn', command_new);
set(gcf, 'KeyPressFcn', command_new);

% Disable zooming during rotation
oldstatus = get(figurenum, 'CameraViewAngleMode');
set(figurenum, 'CameraViewAngleMode', 'manual');

% Set up the rotation
target = get(figurenum, 'CameraTarget');

origin = zeros(1,3);
if (constDim == 1)
    amp = sqrt((camStartPos(1,2) - target(1,2))^2 + ...
        (camStartPos(1,3) - target(1,3))^2);
    origin(1,1) = camStartPos(1,1);
    origin(1,2) = target(1,2);
    origin(1,3) = target(1,3);

    start_t1 = asin((camStartPos(1,2) - origin(1,2)) / amp);
    start_t2 = acos((camStartPos(1,3) - origin(1,3)) / amp);
elseif (constDim == 2)
    amp = sqrt((camStartPos(1,1) - target(1,1))^2 + ...
        (camStartPos(1,3) - target(1,3))^2);
    origin(1,1) = target(1,1);
    origin(1,2) = camStartPos(1,2);
    origin(1,3) = target(1,3);

    start_t1 = asin((camStartPos(1,1) - origin(1,1)) / amp);
    start_t2 = acos((camStartPos(1,3) - origin(1,3)) / amp);
else
    amp = sqrt((camStartPos(1,1) - target(1,1))^2 + ...
        (camStartPos(1,2) - target(1,2))^2);
    origin(1,1) = target(1,1);
    origin(1,2) = target(1,2);

```

```

origin(1,3) = camStartPos(1,3);

start_t1 = asin((camStartPos(1,1) - origin(1,1)) / amp);
start_t2 = acos((camStartPos(1,2) - origin(1,2)) / amp);
end

t1 = start_t1:0.002*panSpeed:numRotations*2*pi + start_t1;
t2 = start_t2:0.002*panSpeed:numRotations*2*pi + start_t2;

for j = 1:size(t1,2)
    if (get(gcf, 'UserData') == true)
        % Reset and re-enable zooming in case the cam position is ever
        % changed
        set(figurenum, 'CameraPosition', camStartPos);
        set(figurenum, 'CameraViewAngleMode', oldstatus);
        break;
    end

    newCamPos = zeros(1,3);
    if (constDim == 1)
        newCamPos(1,1) = camStartPos(1,1);
        newCamPos(1,2) = origin(1,2) + amp * sin(t1(j));
        newCamPos(1,3) = origin(1,3) + amp * cos(t2(j));
    elseif (constDim == 2)
        newCamPos(1,1) = origin(1,1) + amp * sin(t1(j));
        newCamPos(1,2) = camStartPos(1,2);
        newCamPos(1,3) = origin(1,3) + amp * cos(t2(j));
    else
        newCamPos(1,1) = origin(1,1) + amp * sin(t1(j));
        newCamPos(1,2) = origin(1,2) + amp * cos(t2(j));
        newCamPos(1,3) = camStartPos(1,3);
    end

    set(figurenum, 'CameraPosition', newCamPos);
    drawnow;
end

% Reset and re-enable zooming in case the cam position is ever changed
set(figurenum, 'CameraPosition', camStartPos);
set(figurenum, 'CameraViewAngleMode', oldstatus);

```