Automated Control of Thermal Ablation Using 3D Echo Decorrelation Imaging

A Method of Improving Treatment of Soft Tissue Cancers Using Ultrasound Imaging

A Baccalaureate Capstone Report submitted to the
Department of Electrical Engineering and Computer Science in the College of Engineering and Applied Science at the University of Cincinnati

In partial fulfillment of the requirements for the degree of

Bachelor of Science
in Electrical Engineering
by

Peter Grimm

Advised by:
Dr. Douglas Mast

Depeartment of Electrical Engineering and Computer Science
University of Cincinnati
United States
May 10, 2019

UNIVERSITY OF CINCINNATI COLLEGE OF ENGINEERING AND APPLIED SCIENCES

Automated Control of Thermal Ablation Using Three-Dimensional Echo Decorrelation Imaging

Peter Grimm
May 10, 2019

CONTENTS

1. Abstract **5**

2. Introduction **6**
2.1. Problem . . . 6
2.2. Solution . . . 6
2.3. Credibility . . . 7
2.4. Project Goals and Scope . . . 7

3. Discussion **8**
3.1. Project Concept . . . 8
3.1.1. Overview of Echo Decorrelation . . . 8
3.1.2. Overview of Devices and Control . . . 10
3.2. Design Objectives . . . 12

4. Methodology **15**
4.1. MATLAB Echo Decorrelation Implementation . . . 15
4.1.1. Data Access . . . 15
4.1.2. Scan Conversion . . . 15
4.1.3. Decorrelation . . . 17
4.1.4. Verification . . . 18
4.1.5. Program Structure . . . 21
4.2. C++ Echo Decorrelation Imeplentation . . . 22
4.2.1. Data Access . . . 22

4.2.2. Scan Conversion . . . 24
4.2.3. Decorrelation . . . 24
4.2.4. Program Structure . . . 24
4.3. RF Generator Interface . . . 26
4.3.1. RF Generator Data Output . . . 26
4.3.2. RF Generator Control . . . 27
4.4. Ultrasound Machine Automation . . . 31
4.5. MATLAB Control Algorithm Implementation . . . 31
4.5.1. Budget . . . 32
4.6. Timeline . . . 33
4.7. Problems and Limitations . . . 34
4.8. Future Tasks . . . 35

5. Conclusion **36**

A. C++ code **38**
A.1. Decorrelation Class Header . . . 38
A.2. Decorrelation Class Source . . . 42
A.3. RF Interface Header . . . 60
A.4. RF Interface Source . . . 66

B. MATLAB code **79**
B.1. Ultrasound Data Class . . . 79
B.2. Decorrelation Computation . . . 83
B.3. Scan Conversion . . . 85
B.4. Experiment Class . . . 88
B.5. MATLAB gui . . . 95

C. Other Code **121**
C.1. ROI R Code . . . 121
C.2. Arduino Code . . . 121
C.3. Siemens Scanner Automation Code . . . 121

LIST OF FIGURES

3.1. Demonstration of echo decorrelation of single a-line . . . 8
3.2. Example decorrelation maps . . . 9
3.3. Block diagram of automated control setup . . . 11
3.4. Control flow chart . . . 12
3.5. Diagram of project tasks . . . 13
3.6. RF generator objective tree . . . 14
3.7. Decorrelation implementation objective tree . . . 14
4.1. Spherical data slice (left), cartesian data slice (right) . . . 16
4.2. Decorrelation over time plotted with temperature . . . 19

4.3. Thresholding example 20
4.4. Demonstration of echo decorrelation as a predictor of thermal lesions 20
4.5. Example ROC curve computed from slice shown in Figure 4.4 21
4.6. UML diagram of MATLAB class implementation 22
4.7. UML diagram of C class structure 25
4.8. RF generator serial data flow chart 26
4.9. RF generator serial output data 27
4.10. Pump block diagram 28
4.11. Pump flowchart 29
4.12. Pump circuit schematic 30
4.13. Image of pump circuit 30
4.14. RFA control GUI 32
4.15. Budget pie chart 33
4.16. List of tasks 34
4.17. Gantt chart 34

1. ABSTRACT

Thermal ablation is the clinical standard for treatment of unresectable soft tissue cancers such as hepatocellular carcinoma. Despite this, treatment dosage is dependent on estimation by the treating physician, which can lead to complications related to non-ideal treatment dosage. Real time monitoring and control for these procedures has the potential to improve treatment outcomes through reducing the likelihood of cancer recurrence as well as reducing the incidence of side effects related to healthy tissue ablation.

Echo decorrelation imaging is a pulse-echo ultrasound imaging method that has been shown to be effective in both real time monitoring and control of thermal ablation procedures. Previous research has been conducted using 2D ultrasound probes, which limits decorrelation imaging to a 2D plane of tissue, however recent advances in computing power have made 3D ultrasound imaging cost-effective and practical. 3D ultrasound imaging is a potentially superior imaging modality for monitoring and control of thermal ablation procedures as it is able to capture the entire ablated volume. This project seeks to extend the echo decorrelation method to 3D as a means to improve monitoring and control techniques for thermal ablation procedures.

2. INTRODUCTION

2.1. PROBLEM

Thermal ablation procedures are limited by the treating physicians ability to accurately control treatment dosage. Improper dosage can lead to complications related to both undertreatment and overtreatment. Undertreatment can lead to recurrence due to insufficient ablation while overtreatment can lead to undesirable damage to neighboring tissue. Image guided monitoring and control of these procedures can reduce the risk of undertreatment and overtreatment by optimizing the volume of ablated tissue in real time.

2.2. SOLUTION

Echo decorrelation imaging is a pulse-echo ultrasound imaging technique maps heat-induced spatial decorrelation between echo signals on a millisecond timescale [4]. This methodology has been shown to effectively predict real-time thermal lesioning in Radio Frequency Ablation (RFA) and High Intensity Focused Ultrasound (HIFU) in-vitro [5] and in-vivo [2] trials of animal liver tissue.

Previous research completed by Abbas et al. [1] has shown the efficacy of echo decorrelation imaging for real-time control of ablation in ex-vivo bovine liver. This previous research was completed using 2D ultrasound arrays, which limits the potential of decorrelation imaging to a single 2D plane of tissue. Recent advances in both computational power and ultrasound probe technology has lead to the emergence of 3D matrix array probes that are capable of imaging entire volumes of tissue during one pulse cycle. This novel imaging modality has the potential to increase the efficacy of echo-decorrelation controlled thermal lesioning.

This project aims to implement real-time control of radio frequency ablation (RFA) using 3D echo decorrelation imaging.

2.3. Credibility

I have experience with automation, ultrasound image processing and MATLAB/C++/Python programming. I also have extensive classroom knowledge of signal processing, control theory, statistics, electronics and computer science. I have completed a co-op term at the University of Cincinnati Biomedical Acoustics Laboratory, which has provided me with experience related to biomedical ultrasound image processing.

2.4. Project Goals and Scope

The aim of this project is to implement automated control of RFA in an ex-vivo bovine model of liver cancer using 3D echo decorrelation imaging. This will involve creation of software to compute decorrelation, as well as software and hardware for to control the devices used for performing ablation procedures. In addition to this statistical analysis must be performed to find an effective cutoff for predicting ablated tissue.

The scope of the project will cover the implementation of 3D echo decorrelation, as well as the control and automation of the necessary subsystems that will be used for performing thermal ablation trials.

Tasks include:

- 3D Echo Decorrelation prototype implementation in MATLAB.
- Automation of Siemens SC2000 ultrasound scanner.
- Creation of software libraries and hardware for the Rita 1500x RF generator.
- Control algorithm prototype implementation in MATLAB
- 3D Echo Decorrelation/Control implementation in C++

3. Discussion

3.1. Project Concept

The objective of this project is the creation of a system for controlling RFA procedures as well as verifying the efficacy of echo decorrelation as both a monitoring and control technique. The project is roughly split into two subsections: Implementation of echo decorrelation (section 3.1.1) and control of the subcomponents needed for automation (section 3.1.2).

3.1.1. Overview of Echo Decorrelation

Echo decorrelation is a pulse-echo ultrasound imaging technique that maps millisecond timescale heat induced decorrelation due to thermal lesioning [4]. A demonstration of this concept on a single a-line of ultrasound echo data can be seen in Figure 3.1.

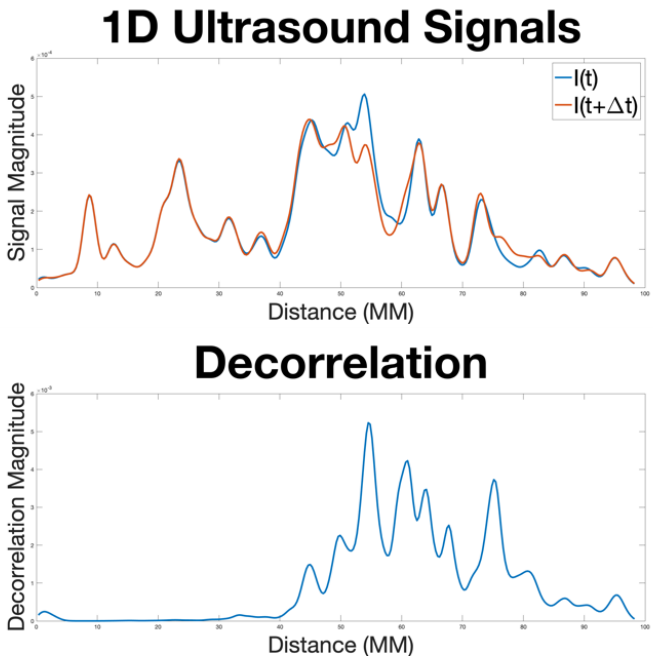


Figure 3.1: Demonstration of echo decorrelation of single a-line

This project extends previously implemented [1] decorrelation methods to 3D. This

method will map a decorrelation parameter to each voxel in a 3D matrix corresponding to local decorrelation at that position in the volume. An example heatmap of a 2D slice of decorrelation map data can be seen in Figure 3.2.

(a) Example slice of decorrelation map (b) Decorrelation map overlaid on b-mode slice

Figure 3.2: Example decorrelation maps

Normalized echo decorrelation maps are computed using the following formula:

$$\Delta(x,y,z,t) = 2 \cdot \left[\frac{\beta^2(x,y,z,t) - |R_{01}(x,y,z,t)|^2}{\tau \cdot [\beta^2(x,y,z,t) + \overline{\beta^2(t)}]} \right] \tag{3.1}$$

The windowed autocorrelation R_{01} between volumes at time t and $t+\tau$ is defined as:

$$R_{01}(x,y,z,t) = \langle I(x,y,z,t) \cdot I(x,y,z,t+\tau)^* \rangle \tag{3.2}$$

Where $\langle \cdot \rangle$ denotes 3D convolution with a gaussian window.

$$\langle I(x,y,z) \rangle = I(x,y,z) * e^{-\frac{(x^2+y^2+z^2)}{2\sigma^2}} \tag{3.3}$$

The zero spatial and temporal lag autocorrelation R_{00} and R_{11} of each volume in the two volume set is computed as:

$$R_{00} = \langle |I(x,y,z,t)|^2 \rangle \tag{3.4}$$

$$R_{11} = \langle |I(x,y,z,t+\tau)|^2 \rangle \tag{3.5}$$

The Integrated Back Scatter (IBS) term β is representative of backscattered echo energy.

It is computed as the square root of the product of the zero spatiotemporal lag autocorrelations at times t and $t+\tau$:

$$\beta(x,y,z,t) = \sqrt{R_{11} \cdot R_{00}} \tag{3.6}$$

The spatial mean term $\overline{\beta^2(t)}$ is defined as:

$$\overline{\beta^2(t)} = \frac{1}{N_x \cdot N_y \cdot N_z} \sum_{x=1}^{N_x} \sum_{y=1}^{N_y} \sum_{z=1}^{N_z} \beta^2(x,y,z,t) \tag{3.7}$$

In the decorrelation formula the term in the numerator $\beta^2(x,y,z,t) - |R_{01}^2(x,y,z,t)|$ measures transient heat induced changes in the autocorrelation at each voxel, such that at regions of high change the decorrelation is high, while at regions of low change the decorrelation is low. The term in the denominator $\tau[\beta^2(x,y,z,t) + \overline{\beta^2(x,y,z,t)}]$ normalizes the decorrelation to both the local IBS and the spatial mean of the IBS. Normalization by both terms is used to remove artifictual decorrelation in regions of both low and high echogenicity[6].

Cumulative decorrelation maps will be used as a predictor in the control algorithm. Cumulative maps are computed as the maximum of each pixel from all computed data sets.

$$\Delta(x,y,z,N)_{cumulative} = \max(\Delta(x,y,z,t), \Delta(x,y,z,t+\tau), ..., \Delta(x,y,z,t+N\tau)) \tag{3.8}$$

Where N is the current data set index.

The implementation of echo decorrelation is covered in more depth in sections 4.1 and 4.2.3.

3.1.2. Overview of Devices and Control

Automated control requires interoperation between the Siemens SC2000 ultrasound machine, the Rita 1500x RF generator and a central control computer. A block diagram of the full control setup can be seen in Figure 3.3.

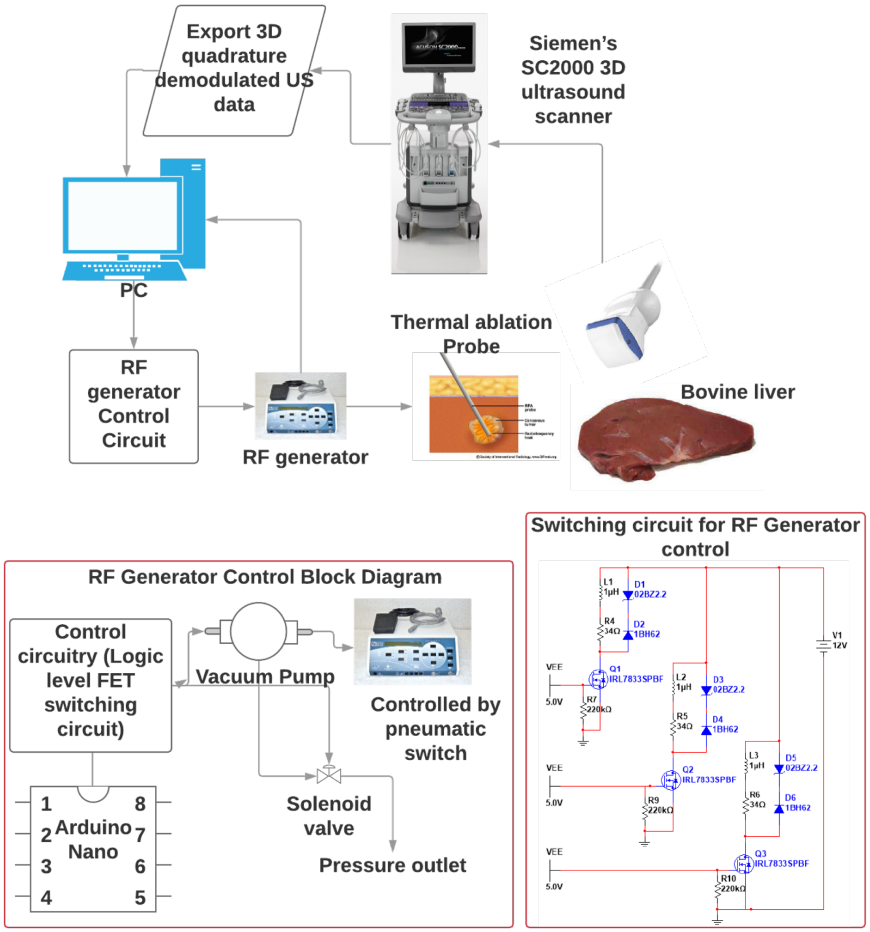


Figure 3.3: Block diagram of automated control setup

The devices will be controlled by a central control program, which orchestrates the devices via serial and network connections. Ultrasound data is transferred from the Siemens SC2000 scanner to the control computer which calculates decorrelation. The control computer, depending on the algorithm described in Figure 3.4, then will decide whether or not to activate the RF generator to end the procedure. The specific operation and interface between these devices is expanded upon in sections 4.3 and 4.4.

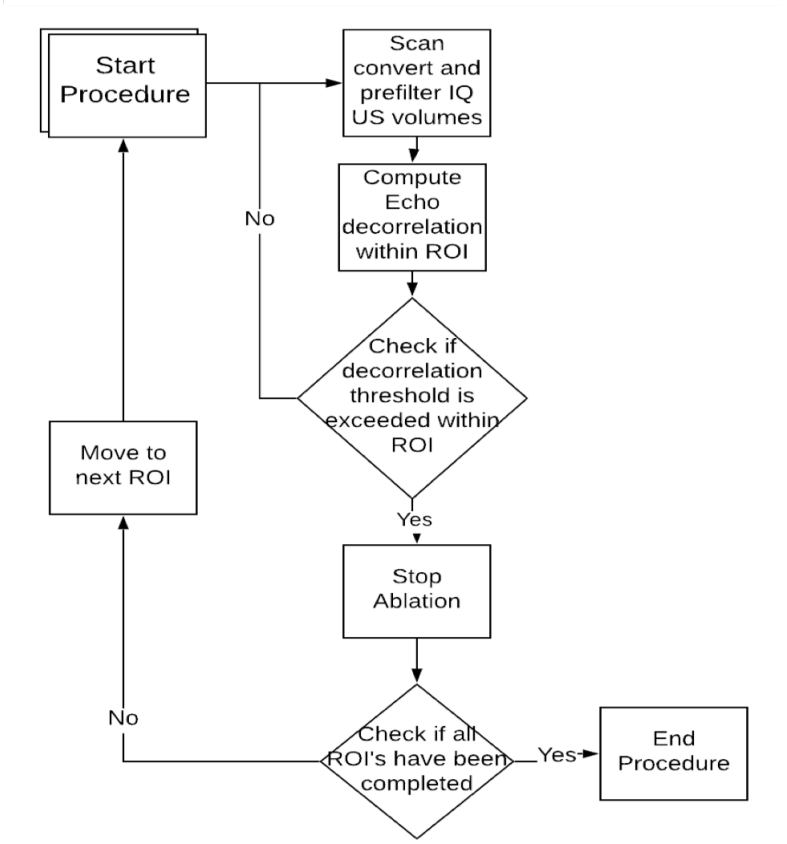


Figure 3.4: Control flow chart

Control is based off of the cumulative decorrelation map $\Delta_{cumulative}$, described in section 3.2, within a user defined region of interest (ROI). When the average cumulative decorrelation within the ROI exceeds a set threshold the procedure will be stopped, thus optimizing the treatment dosage. This threshold is determined empirically through the methods detailed in section 4.1.4. Implementation of the control algorithm will be further expanded on in section 4.5.

3.2. Design Objectives

A high level overview of the tasks required for implementing automated control of RFA with echo decorrelation imaging is displayed as chart in Figure 3.5. The tasks can be roughly

divided into two sections: Implementation of echo decorrelation imaging and control of the machines for automated ablation procedures.

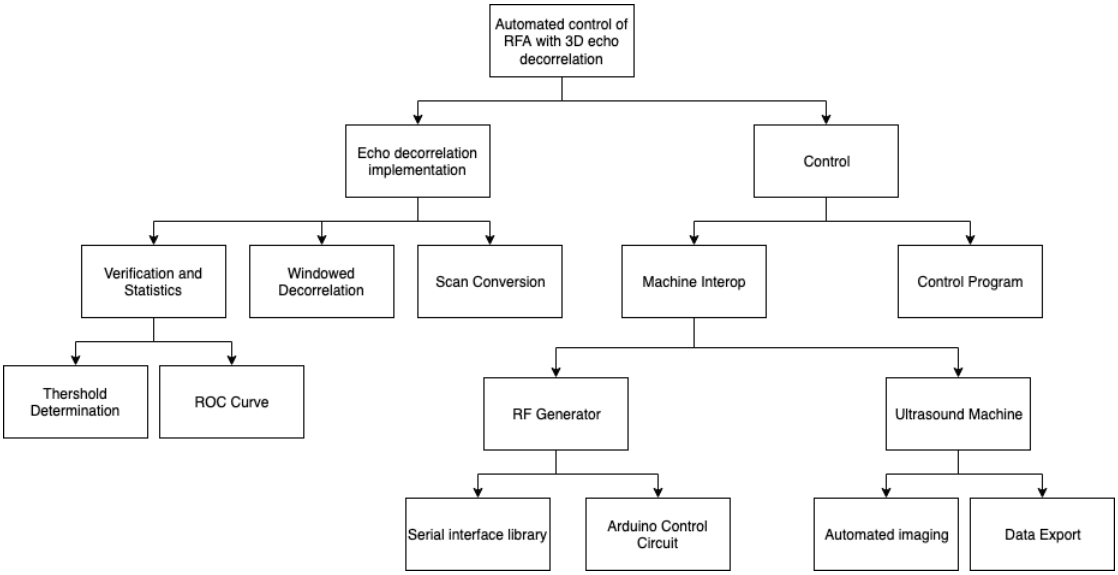


Figure 3.5: Diagram of project tasks

For each specific task an objective tree was constructed. The objective tree for the RF generator can be seen in Figure 3.6, and the objective tree for decorrelation implementation can be seen in Figure 3.7.

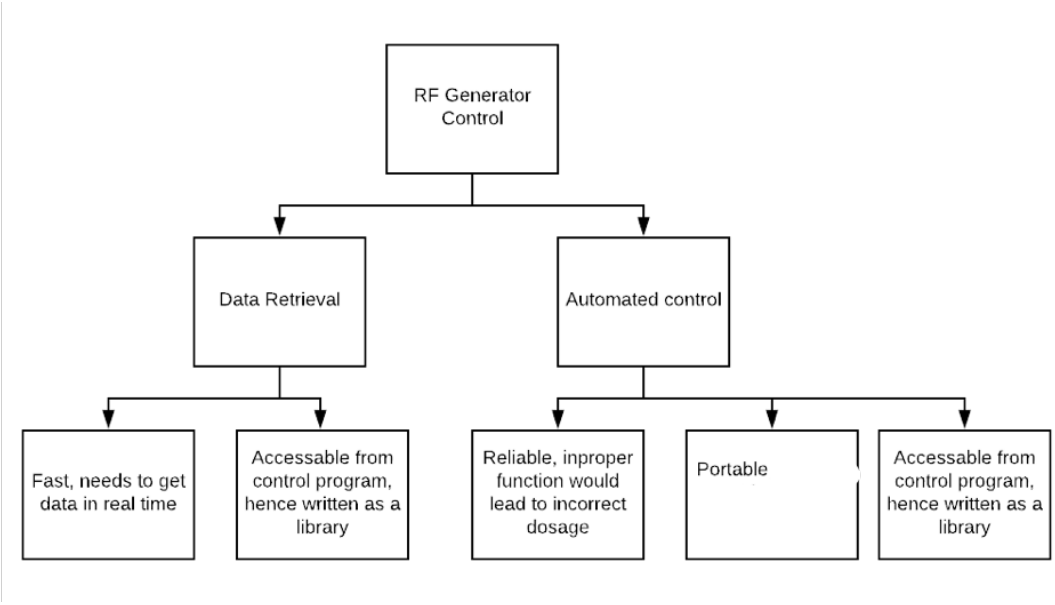


Figure 3.6: RF generator objective tree

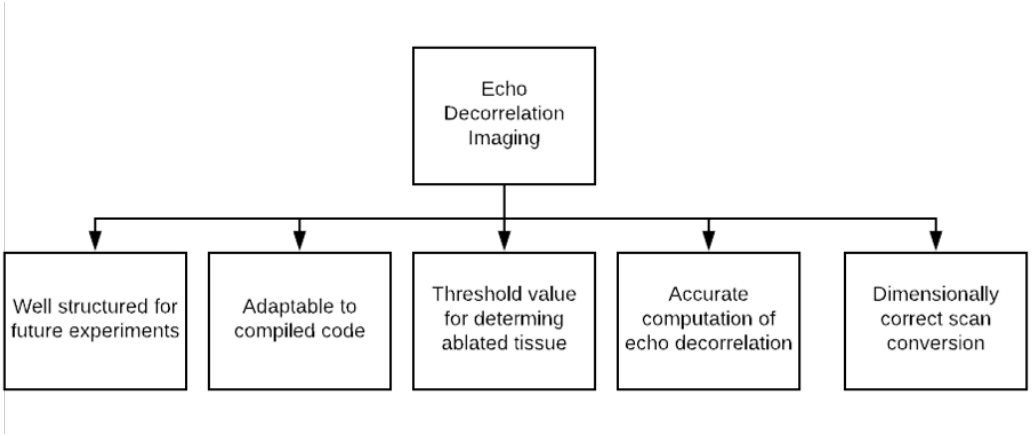


Figure 3.7: Decorrelation implementation objective tree

4. METHODOLOGY

4.1. MATLAB ECHO DECORRELATION IMPLEMENTATION

A prototype implementation of 3D echo decorrelation was constructed in MATLAB [7]. This code takes the form of a library of Object Oriented (OOP) class definitions which were created as a means to organize code, reduce redundancy and optimize the echo decorrelation method before it is translated into compiled C++ code. The function and structure of the MATLAB library will be explored in this section.

4.1.1. DATA ACCESS

The Siemens SC2000 scanner is set up to export a buffer of two In-phase and Quadrature (IQ) demodulated complex spherical volumes, it does so as a standard text file of UTF-8 encoded characters representing the real and imaginary component of each voxel as a list of 20 bit signed 2's complement hexadecimal numbers. These hexadecimal numbers represent the data as a base 2 number where the first bit is the first digit after a decimal point ($0.b_1b_2b_3$...). This data is read into MATLAB, interpreted into standard floating point numbers, and reshaped into a 4D matrix of complex numbers.

One decorrelation map is computed per buffer exported from the scanner, this means that for real time control a continuous stream of discrete buffers will be sent over an ad-hoc ethernet network to the control computer.

4.1.2. SCAN CONVERSION

In order to compute echo decorrelation the spherical ultrasound data must be scan converted onto a Cartesian space. This is accomplished using an mapping from the Cartesian space to an intermediary pyramidal space, and interpolating based on the nearest neighbors in the pyramidal data.

The intermediary space is created based on a scaling factor between the radial sample

distance and the cartesian grid. This factor determines the resolution of the output scan converted volume, and as such affects the computational complexity of all subsequent operations to a large extent.

The inverse mapping from the Cartesian to pyramidal space is defined as follow:

$$R = \sqrt{x^2 + y^2 + z^2} \tag{4.1}$$

$$\mu = \frac{y}{\sqrt{z^2 + y^2}} \tag{4.2}$$

$$\nu = \frac{x}{\sqrt{R^2 - y^2}} \tag{4.3}$$

Nearest neighbor points for each point in the pyramidcal coordinates is used to interpolate the values onto a Cartesian grid. A detailed MATLAB implementation created by Elmira Ghahramani and Dr. Mast is shown in Appendix B.3.

An image of a slice of spherical data is show in Figure 4.14.

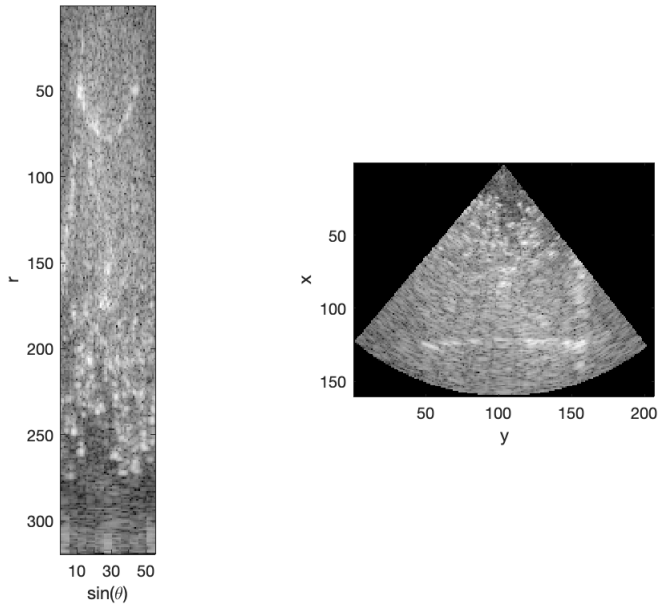


Figure 4.1: Spherical data slice (left), cartesian data slice (right)

4.1.3. DECORRELATION

Decorrelation maps are computed from the scan converted 4D data sets according to the following formula, as described in section 3.1.1:

$$\Delta(x,y,z,t) = 2 \cdot \left[\frac{\beta^2(x,y,z,t) - |R_{01}(x,y,z,t)|^2}{\tau \cdot [\beta^2(x,y,z,t) + \overline{\beta^2(t)}]} \right] \tag{4.4}$$

The MATLAB implementation computes the windowing convolution in the frequency domain due to its much lower time complexity ($\mathcal{O}(n^6)$ for linear convolution and $\mathcal{O}(n^3 \log^3 n)$ for convolution in the frequency domain). The operation then changes from

$$\langle I(x,y,z) \rangle = I(x,y,z) * e^{-\frac{(x^2+y^2+z^2)}{2\sigma^2}} \tag{4.5}$$

to

$$\langle I(x,y,z) \rangle = I(x,y,z) \circledast e^{-\frac{(x^2+y^2+z^2)}{2\sigma^2}} \tag{4.6}$$

Or equivalently in the frequency domain by the convolution theorem:

$$\langle I(x,y,z) \rangle = \mathscr{F}^{-1}\{\mathscr{F}\{I(x,y,z)\} \cdot \mathscr{F}\{e^{-\frac{(x^2+y^2+z^2)}{2\sigma^2}}\}\} \tag{4.7}$$

where $\mathscr{F}$ denotes the 3D Discrete Fourier Transform and $\mathscr{F}^{-1}$ denotes the inverse 3D Discrete Fourier Transform.

Frequency domain convolution is performed on each volume in the two volume set, as well as on their autocorrelation. The output of these convolutions is then used to compute the decorrelation maps using elementwise multiplication and division. This implementation can be seen in appendix B.2.

4.1.4. VERIFICATION

Verifying the MATLAB implementation took the form of ensuring each subcomponent is effectively performing its role.

One way in which the MATLAB implementation could be verified is through checking whether or not decorrelation increases with temperature. Since decorrelation tracks heat induced changes in local correlation, then it should be highly correlated with temperature. This is shown in figure 4.2 which shows temperature received from the RF probe compared with the normalized average decorrelation map at that time. From the plot it is clear to see that the decorrelation and temperature are highly interdependent. The pattern of decorrelation increasing with temperature, and then decreasing after a certain point is expected, and is the reason for using cumulative decorrelation in the control algorithm.

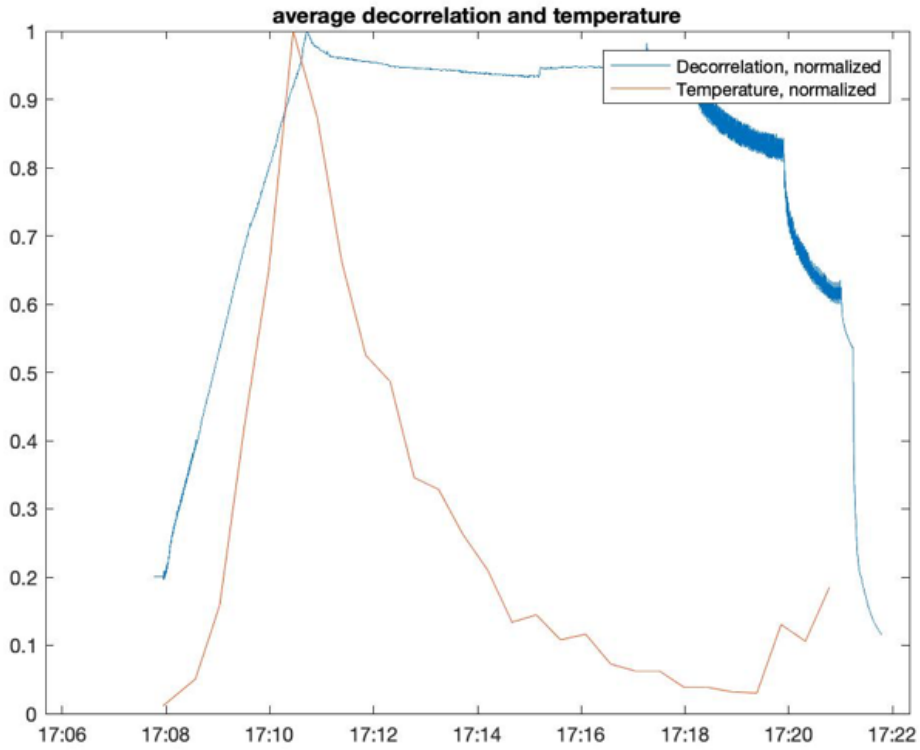


Figure 4.2: Decorrelation over time plotted with temperature

For an initial rough estimation of threshold parameters an ROC curve was constructed from decorrelation maps and scans of ablated tissue. A more robust implementation of the ROC curve described here will be used to verify echo decorrelation, as well as to provide a basis for the chosen threshold value in the control algorithm (Section 4.5). This more robust implementation will consider correlation between voxels as a result of the windowing process, and will allow for more accurate thresholding based on more slices of ablated tissue.

The process of creating the ROC curve starts with the computation of 3D echo decorrelation, and the generation of the normalized decorrelation maps. Images are then taken of the ablated tissue at different depths, these images are aligned with their respective positions in the decorrelation map data. A mask is then determined from the ablated tissue images, indicating the regions where the tissue was ablated. Different threshold values are applied to

the decorrelation maps (figure 4.3), and the number of false negative and false positive points between the thresholded decorrelation maps and the masks from the scans are determined. These points are plotted on an ROC curve to find an optimal threshold.

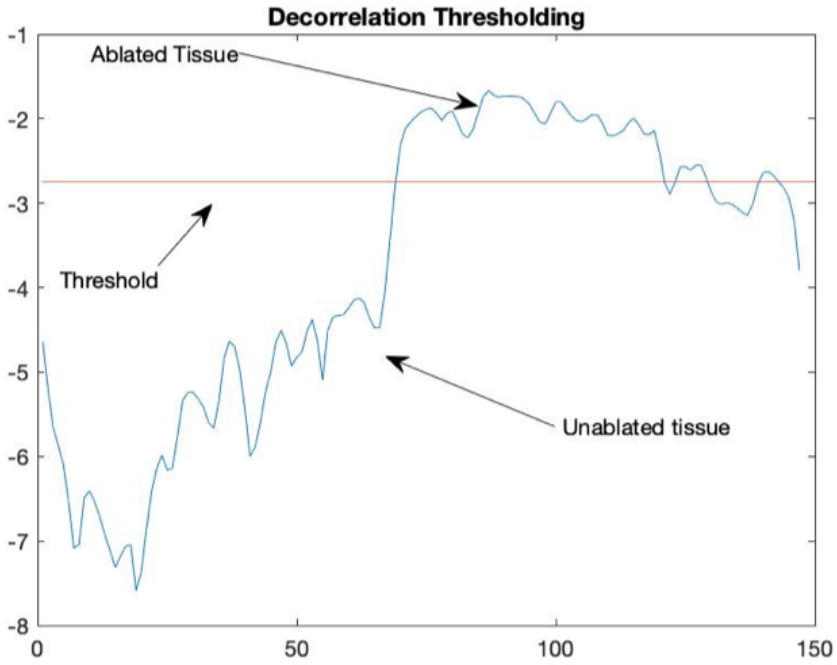
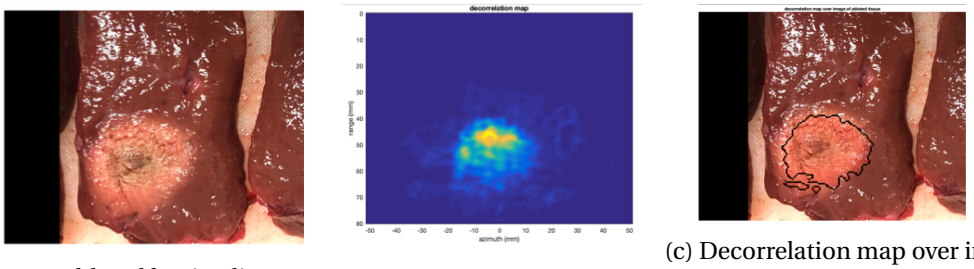


Figure 4.3: Thresholding example



(a) Ablated bovine liver

(b) Decorrelation map

(c) Decorrelation map over image of ablated bovine liver

Figure 4.4: Demonstration of echo decorrelation as a predictor of thermal lesions

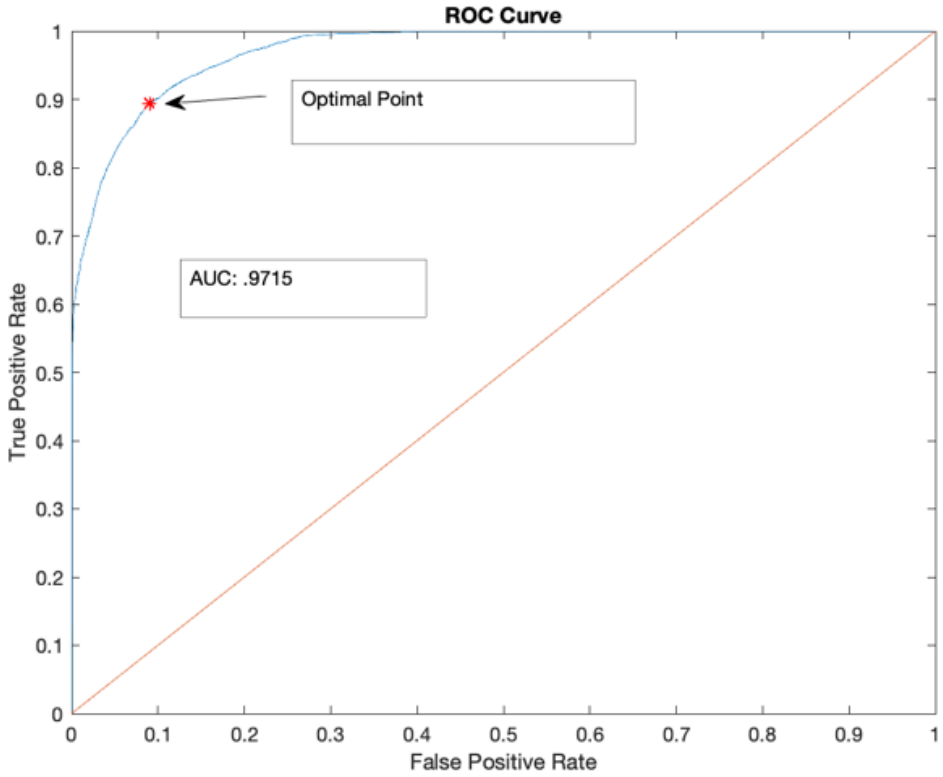


Figure 4.5: Example ROC curve computed from slice shown in Figure 4.4

4.1.5. Program Structure

For ease of programming and prototyping the MATLAB implementation of echo decorrelation follows an Object Oriented Programming (OOP) paradigm. A MATLAB handle class was constructed for computing all relevant decorrelation quantities for each data set. This structure reduces memory overhead by being able to pass class handles to the various functions and operations by reference. This also allowed for superior prototyping speed, readability, extensibility while also acting as a blueprint for an implementation in compiled code. This class can be seen in appendix B.1. An additional class was created for containing the instances of the UsDataClass class, which is convenient for proper interface abstraction when constructing complicated programs with the library, such as the gui explored in section 4.5.

A Unified Modeling Language (UML) diagram of the MATLAB class structure can be seen in Figure 4.6.

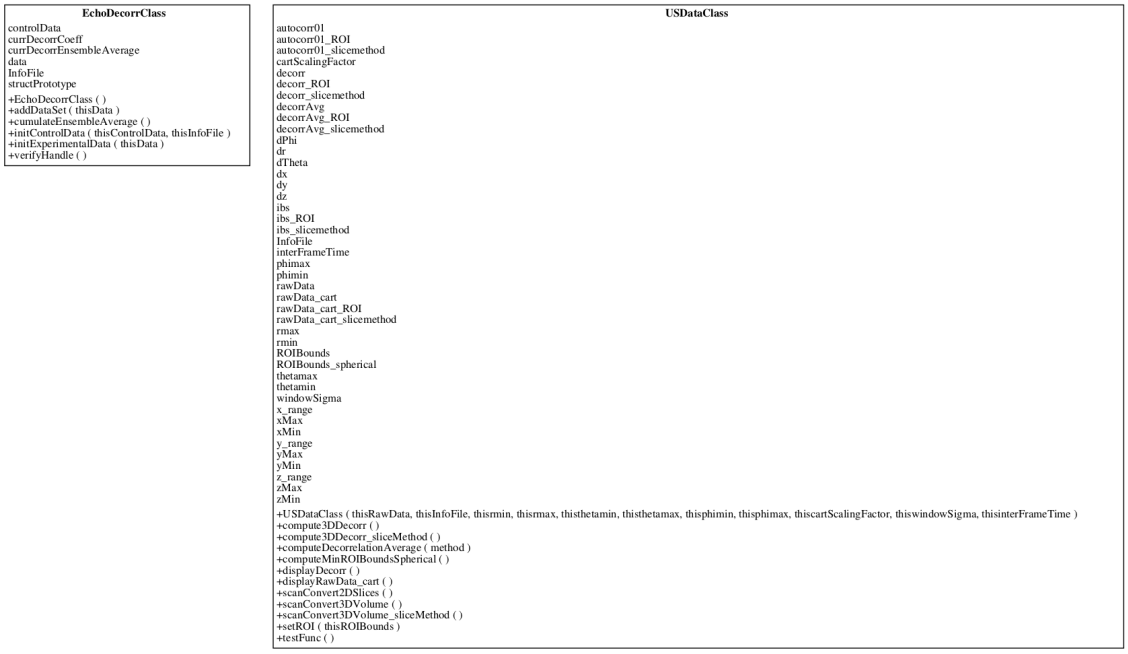


Figure 4.6: UML diagram of MATLAB class implementation

4.2. C++ Echo Decorrelation Imeplentation

The C++ implementation is partially completed, but requires further debugging and optimization. The implementation as it currently exists is explored in this section.

4.2.1. Data Access

The data is exported from the scanner in an identical way as with the MATLAB implementation. The task of importing the data into C++ however required substantial optimizations.

The text file is read into the C++ program using a standard file object, the file is then scanned line by line and the hexadecimal numbers are cast into an integer variable. Due to the non standard nature of the data representation they could not be cast into a float directly.

To convert the formatted hexadecimal numbers into standard floats in as few operations as possible a method was devised using bitwise operations on the interpreted int value.

The 20 bit hexadecimal input is shifted to the bottom bits of the integer and the number is converted to its positive equivalent by computing the two's complement of the number.

$$x_{s1} = \neg(x_{in} \gg b_L) + 1 \tag{4.8}$$

Where $\neg$ is bitwise not, and b_L is the bit distance to the end of the int. the bits of x_{in} for the computation of two's compliment.

A bit shift then shifts the 17 bit positive representation of the data into the first mantissa point of a float representation.

$$x_{s2} = x_{s1} \gg b_{L2} \tag{4.9}$$

This is then shaped into a proper floating point representation by xor'ing the the output with the mask m_2 which removes the sign bit and adds the appropriate mantissa and leading sign bit on the float. After this, due to float mantissas being of the form $1.b_1 b_2 ... b_n$ the leading 1 term must be removed, since the exponent is static and defined in the previous step the leading 1 always adds or subtracts $\frac{1}{2}$ from the number depending on the sign, so this must be done by adding $\pm\frac{1}{2}$ to the output number depending on the sign. The data in the modified int is than referenced and a new float variable is pointed to it, thus storing that data directly into a float.

$$x_{out} = x_{s2} \oplus m_2 \pm \frac{1}{2} \tag{4.10}$$

the full process is defined as:

$$x_{out} = ((\neg(x_{in} \gg b_L) + 1) \gg (b_L - I_L) \oplus m_2) \pm \frac{1}{2} \tag{4.11}$$

This method was able to reduce the time for importing data significantly compared to using pure floating point operations.

4.2.2. Scan Conversion

Scan conversion is accomplished in a similar way to what was explored in section 4.1.2. although with some C-specific optimization methods.

The main difference between the C++ implementation and the MATLAB implementation is that the C++ computation precomputes a vector of inverse maps whenever the relevant parameters are changed, instead of every time a new volume is received by the scanner.

This is accomplished through a data structure that contains the x, y, z cartesian coordinates as well as the pyramical coordinates r, ν, μ and the corresponding coefficients on the nearest neighbor points.

This method is able to scan convert a $200x200x145$ volume in less than 100 milliseconds, which is much faster than the MATLAB method which computes it in approximately 3 seconds.

4.2.3. Decorrelation

Decorrelation is accomplished in a similar way to the MATLAB implementation. It uses the FFTW library [3] for computation of the 3D complex FFT and inverse FFT. This package was compiled using vectorization extensions such as intel SSE as well as multithreading. This significantly improves the performance of the FFT computation, completing all of the required frequency domain convolutions per volume in slightly over a second, which is significantly faster than the MATLAB implementation which can take upwards of 3 seconds to compute the same information.

The speed of both the echo decorrelation algorithm and the scan conversion algorithm is promising for the eventual real time implementation of echo decorrelation.

4.2.4. Program Structure

The structure of the C++ program is similar to the class structure of the MATLAB program. A main class is implemented with experimental parameters, and computes parameters at run

time, such as the quick inverse mapping mentioned in secttion 4.2.2. It allocates buffers that will be reused for computation of data sets as they become available, to reduce the amount of space needed to store the 4D complex matrices. This structure also increases speed by only allocating heap memory when the experimental parameters change, as opposed to when a new volume is added.

This is then enclosed in a larger class structure that is similar to the ExperimentClass in the MATLAB implementation. This part has not yet been completed, but a full UML diagram including the proposed solution can be seen in Figure 4.7.

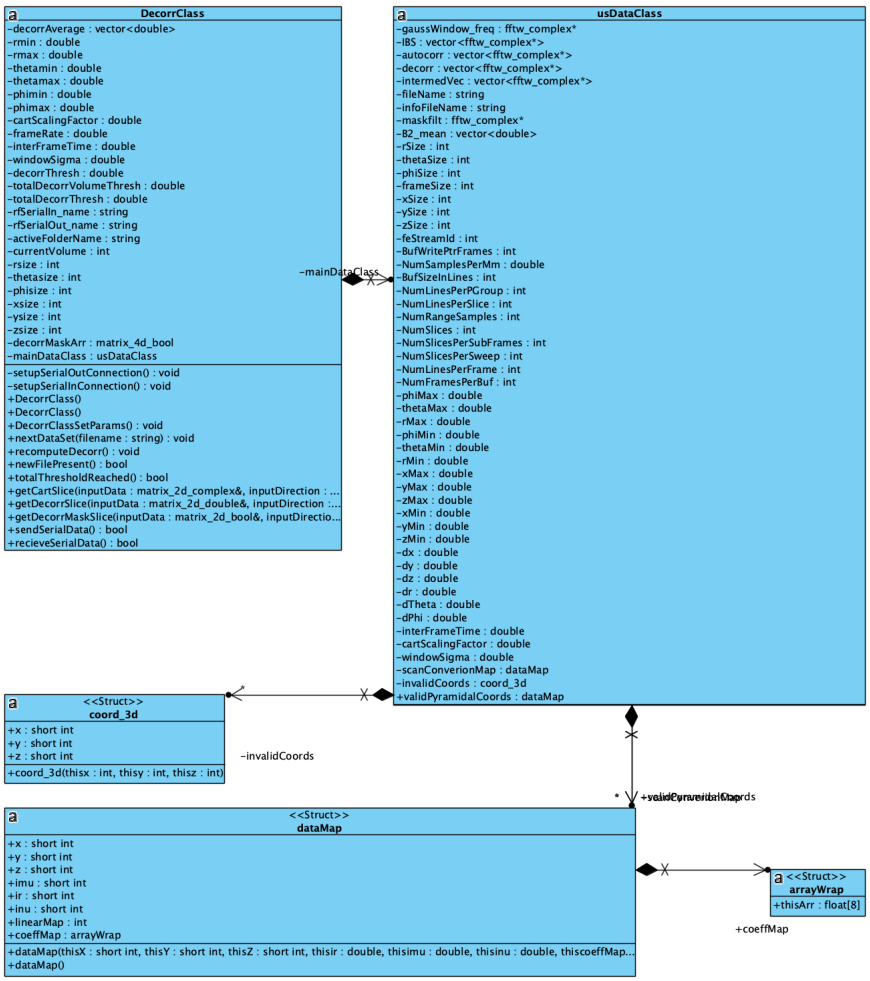


Figure 4.7: UML diagram of C class structure

4.3. RF Generator Interface

The Rita 1500x RF generator with a starburst XLi enhanced probe is used in this project to perform radio frequency ablation. The device came with no software for interfacing it with a computer, so it was necessary to create libraries to interface with the device.

4.3.1. RF Generator Data Output

The RF generator uses a standard RS232 serial output, however there is no documentation for dealing with this output. It was known that the device is capable of outputting data related to temperature, tissue impedance, power delivery, and other useful parameters due to the existence of propretary software that the vendor was not willing to provide. This information is valuable for verifying the efficacy of echo decorrelation, as is described in section 4.1.5.

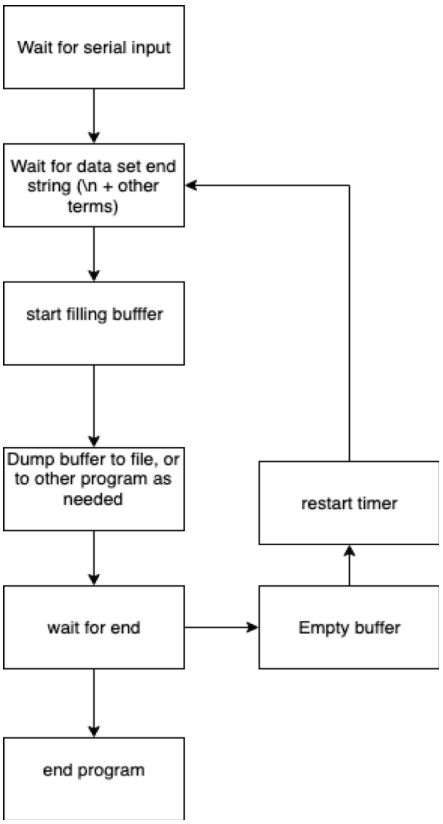


Figure 4.8: RF generator serial data flow chart

A C library for interfacing with this serial stream was created, which allows for calls to obtain serial data through other C program as well as a method for exporting a time series of parameters for analysis in MATLAB. An example output can be seen in Figure 4.9.

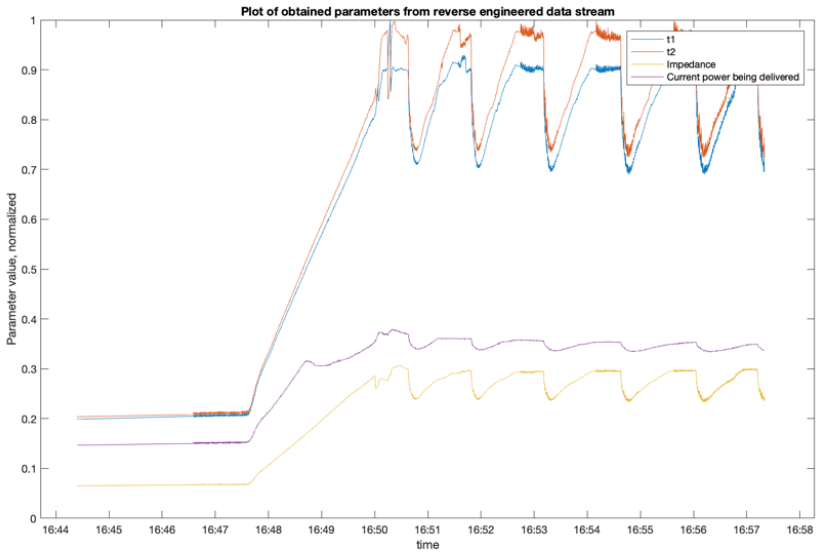


Figure 4.9: RF generator serial output data

4.3.2. RF Generator Control

Starting ablation with the RF generator is only possible using two methods: pressing a button on the front panel, and a pneumatic switch that is meant to be controlled through a foot pedal.

In order to automatically control the device the pneumatic switch would need to be controlled via the control computer. To accomplish this task a micro-controller circuit was constructed using an arduino, a simple logic-level powerFET circuit (Figure 4.12), a vacuum pump and a solenoid valve.

The principle of operation of this device can be seen in Figure 4.10. The vacuum pump is used to trigger the pneumatic switch, which will then build up pressure between the pump and the switch which needs to be released by the solenoid valve in order to be used again.

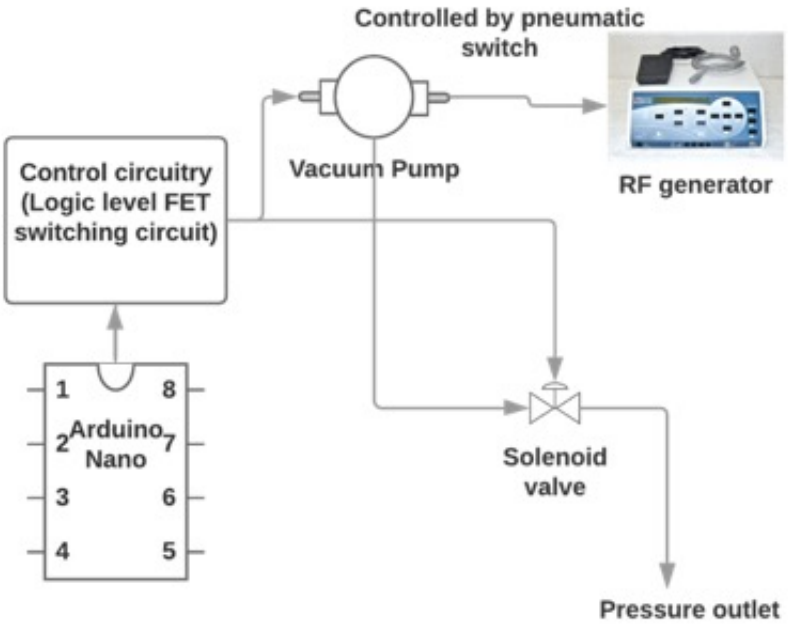


Figure 4.10: Pump block diagram

The circuit is controlled via a serial connection over USB, a control signal is sent by the control computer that triggers the pump to activate. A control flowchart of this process can be seen in Figure 4.11.

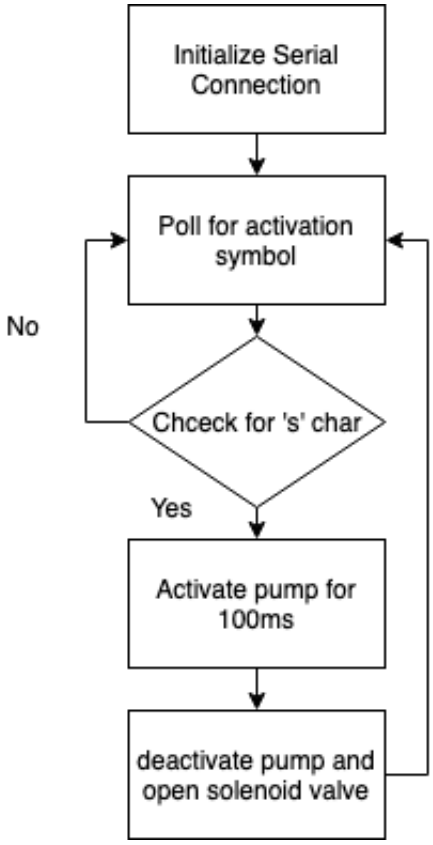


Figure 4.11: Pump flowchart

The circuit used to active the solenoid valve is shown in 4.12. The circuit uses logic level powerFETs to control the high current inductive loads of the valve and pump. A flywheel diode is placed across the loads to eliminate the effects of voltage spikes.

The circuit was constructed and soldered to a prototype board, it was then verified that it worked as expected when signals were sent via a serial stream. After this, code for interacting with the circuit was added to the original library used for getting data from the RF generator, details can be seen in section 4.1.5. A picture of the implemented circuit can be seen in 4.13

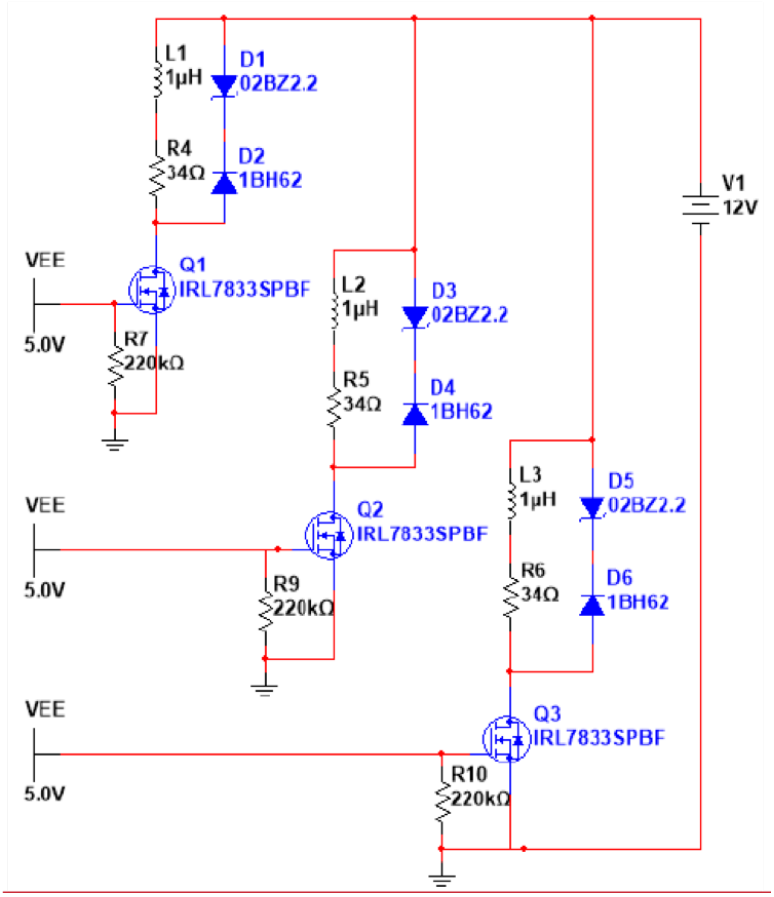


Figure 4.12: Pump circuit schematic

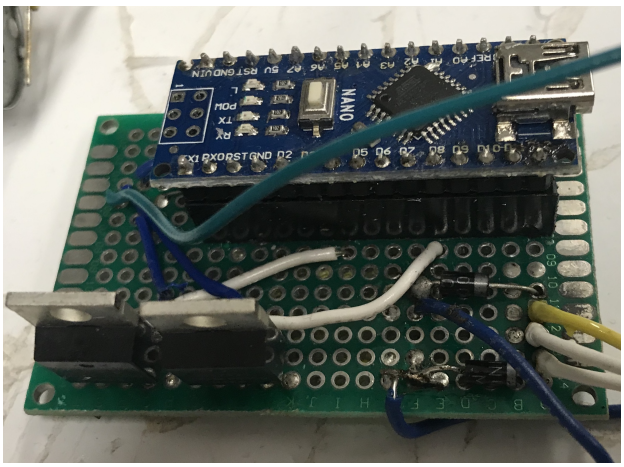


Figure 4.13: Image of pump circuit

4.4. Ultrasound Machine Automation

The siemens scanner had much greater documentation and support than the RF generator. Siemens provided us with scripts for dumping the buffer into a usable hex dump, as well as an early version of the script for interpreting this data. Some modifications were made to these scripts in order for it to have all of the desired characteristics.

One major flaw in the current implementation is the buffer dump process, which takes a significant amount of time. The current method is likely non-ideal, and we hope to work in conjunction with Siemens to develop a better a more efficient method.

4.5. MATLAB Control Algorithm Implementation

The control algorithm described in section 3.1.2 is implemented using an OOP class structure, which is then accessed via a MATLAB graphical user interface (GUI). The ExperimentClass class contains all neccessary aspects of procedure instance, such as the cumulative decorrelation, ROI locations, imaging parameters, and interface definitions. The surrounding GUI code implements decorrelation purely through calling functions from the experiment class, which allows for a highly extensible software interface.

The GUI includes an interface for viewing the ultrasound data sets in 3D by scrubbing through different slices from three directions using a slider. It also includes sliders for setting the dynamic range of the images, as well as the ROI used to control procedures.

The GUI implements serial interaction with the RF generator, as well as having routines for syncing with the network drive of the ultrasound scanner. This is used for control based on the feedback from these machines.

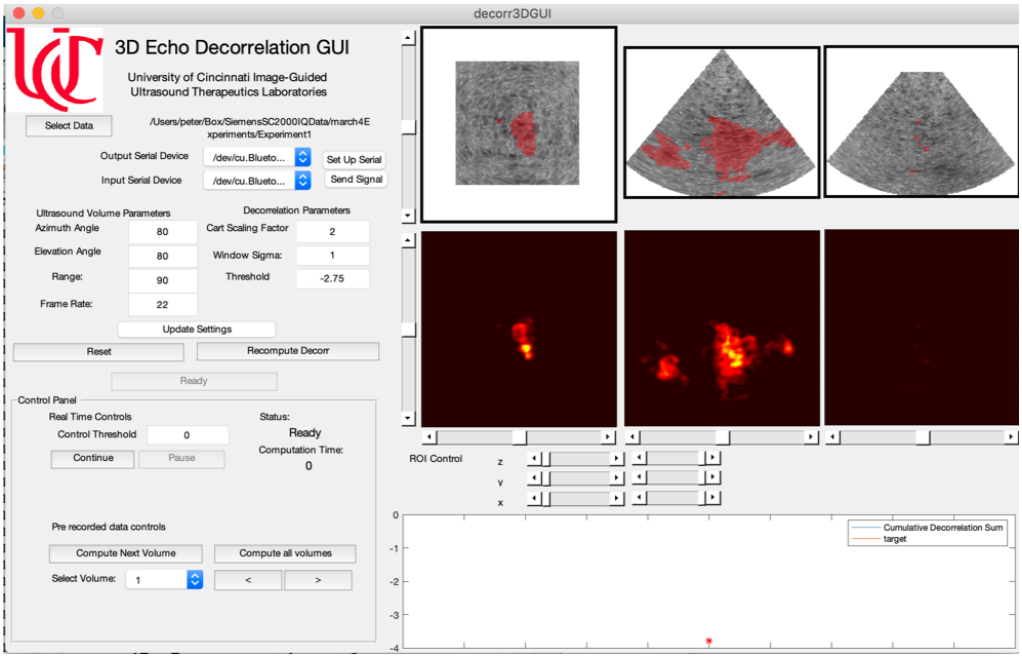


Figure 4.14: RFA control GUI

This GUI has been tested with various imaging parameters and data sets, and is able to compute decorrelation accurately as well as control the devices. The underlying logic is able to effectively control the functions of the GUI, which allows for real time data visualizaiton and control.

4.5.1. Budget

The budget listed in this section covers only the materials used for machine interoperation and testing. It does not include expenses paid for the machinery or other aspects of the project.

A budget listing can be seen in Table 4.1.

Items	Price (USD)	Total (USD)
Case for MicroController, circucit	20	20
MicroController	10	30
Solenoid Valves	16	46
Vacuum Pump	8	54
Wire and solder	.25	54.25
Circuit Components	5	59.25
Prototype Board	.75	60
Ethernet Cables	5	65
USB cables	5	70
Various Other cables (CNC, serial, etc)	20	90
Bovine Liver	50	140
saline, other coupling and analysis materials	2	142

Table 4.1: Budget table

A visual chart of the budget can be seen in figure 4.15.

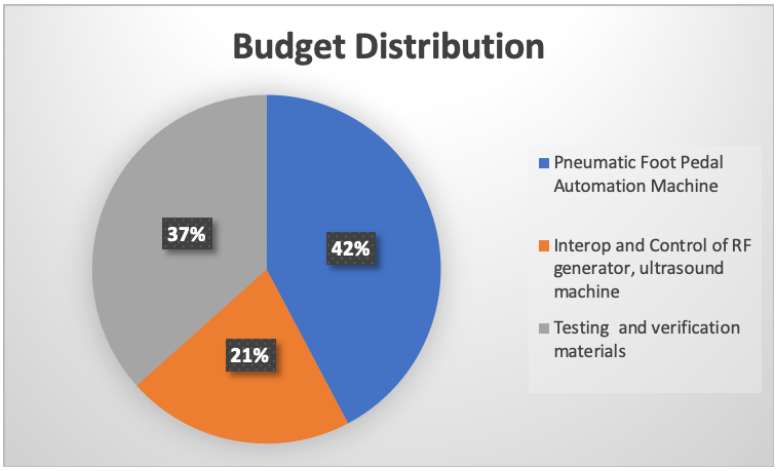


Figure 4.15: Budget pie chart

4.6. Timeline

A timeline of tasks is pictured in 4.16 and a task Gantt chart is pictured in Figure 4.17.

TASK NAME	START DATE	DAY OF MONTH*	END DATE	DURATION* (WORK DAYS)	DAYS COMPLETE*	DAYS REMAINING*	PERCENT COMPLETE
Machine Interoperation							
Pneumatic Pump Machine Design	10/1	1	10/12	12	12	0	100%
Machine interoperation Scripts	10/12	12	10/19	8	7.2	0.8	90%
C++ Libraries for Interfacing with machines	10/19	19	10/26	8	8	0	100%
Interop verification with MATLAB scripts	11/16	16	11/30	15	15	0	100%
Prototype Echo Decorrelation in MATLAB							
Data interpretation and Scan Conversion	10/26	26	12/1	37	37	0	100%
Decorrelation Implementation	12/1	1	2/15	77	73.15	3.85	95%
Full experiment test scripts with interop	1/17	17	3/1	44	39.6	4.4	90%
GUI	4/1	1	4/16	16	11.2	4.8	70%
Echo Decorrelation in C++							
Data Interpretation and Scan Conversion	3/1	1	3/15	15	13.5	1.5	90%
Class structuring	3/10	10	3/20	11	7.7	3.3	70%
Decorrelation Implementation	3/20	20	4/1	13	6.5	6.5	50%
GUI	4/23	23	5/10	18	0	18	0%

Figure 4.16: List of tasks

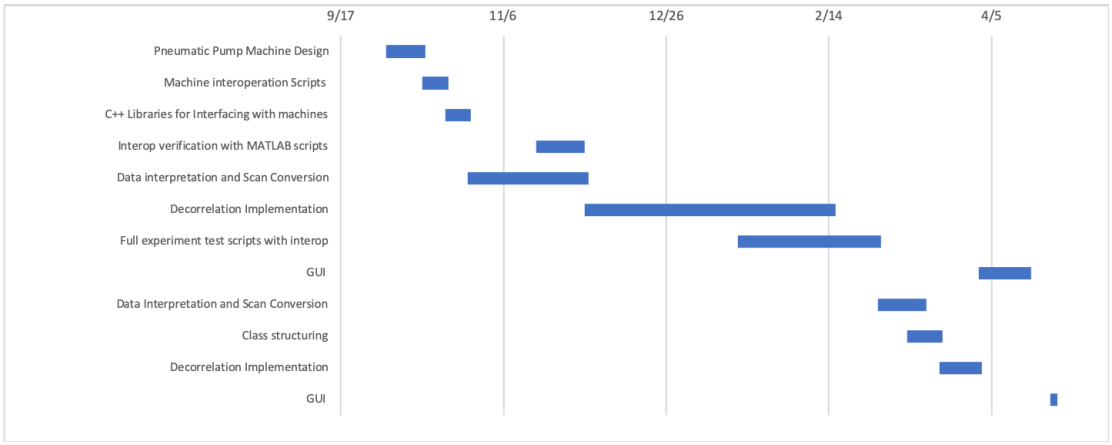


Figure 4.17: Gantt chart

4.7. Problems and Limitations

This project is still in progress and is not yet able to perform truly real time control of thermal ablation procedures. A number of problems need to be overcome before the full project can be implemented, these include:

- The method the Siemens SC2000 scanner exports buffers is non ideal, the standard character encoding and buffer write time slows down volume acquisition significantly.

- The MATLAB code has room for optimization, with some aspects of the program requiring unneccesary overhead.
- The C++ implementation of decorrelation has some differences to the MATLAB version, and must be debugged.

4.8. Future Tasks

Upcoming tasks in the project will mostly revolve around verification of 3D echo decorrelation, as well as creating optimized software for computing decorrelation in real time. The upcoming tasks include:

- A full implementation of the control algorithm in compiled code.
- Sufficient verification of the predictive validity of 3D echo decorrelation, as well as finding the optimal threshold.
- Debugging and optimization of existing decorrelation MATLAB/C++ code.
- More test trials on bovine liver.

5. CONCLUSION

Significant progress has been made on this project, and many components show great promise for the final implementation of the design. The MATLAB implementation is capable of computing 3D echo decorrelation, and produces results in accordance with what is expected. The verification methods employed have shown promising results for echo decorrelation as a prediction of thermal ablation as well. The machine interoperation libraries are functional and libraries for a sustainable interface between the devices has been constructed, which will make implementation into a final full design simple. The prototype MATLAB GUI, while not able to compute true real time decorrelation, is functional and an effective model for a compiled version of the same concept.

There is still a significant amount of tasks that need to be completed in order to implement control in real procedures, as detailed in the future tasks section, but the work done in this capstone project is instrumental to the success of the overall project.

REFERENCES

[1] Mohamed A. Abbass, Jakob K. Killin, Neeraja Mahalingam, Fong M. Hooi, Peter G. Barthe, and T. D. Mast. Real-time spatiotemporal control of high-intensity focused ultrasound thermal ablation using echo decorrelation imaging in ex vivo bovine liver. *Ultrasound in Medicine & Biology*, 44(1):199–213, 2018.

[2] T. R. Fosnight, F. M. Hooi, R. D. Keil, S. Subramanian, P. G. Barthe, , , S. Ahmad, M. B. Rao, and T. D. Mast. Motion-corrected echo decorrelation imaging of in vivo focused and bulk ultrasound ablation in a rabbit liver cancer model. In *2014 IEEE International Ultrasonics Symposium*, pages 2161–2164, Sep. 2014.

[3] Matteo Frigo and Steven G. Johnson. Fftw3.

[4] T. D. Mast, Daniel P. Pucke, Swetha E. Subramanian, William J. Bowlus, Steven M. Rudich, and Joseph F. Buell. Ultrasound monitoring of in vitro radio frequency ablation by echo decorrelation imaging. *Journal of Ultrasound in Medicine*, 27(12):1685–1697, 2008.

[5] T. D. Mast, Daniel P. Pucke, Swetha E. Subramanian, William J. Bowlus, Steven M. Rudich, and Joseph F. Buell. Ultrasound monitoring of in vitro radio frequency ablation by echo decorrelation imaging. *Journal of Ultrasound in Medicine*, 27(12):1685–1697, 2008.

[6] T. D. Mast, Daniel P. Pucke, Swetha E. Subramanian, William J. Bowlus, Steven M. Rudich, and Joseph F. Buell. Ultrasound monitoring of in vitro radio frequency ablation by echo decorrelation imaging. *Journal of Ultrasound in Medicine*, 27(12):1685–1697, 2008.

[7] MATHWORKS. Matlab.

APPENDIX A C++ CODE

A.1 DECORRELATION CLASS HEADER

```
//
//  SiemensSC2000.hpp
//  echoDecorr3D
//
//  Created by Peter Grimm on 3/2/19.
//  Copyright Â© 2019 Peter Grimm. All rights reserved.
//

#ifndef SiemensSC2000_hpp
#define SiemensSC2000_hpp

#include <bitset>
#include <stdio.h>
#include <complex>
#include <fftw3.h>
#include <vector>

#endif /* SiemensSC2000_hpp */
typedef std::complex<double> Complex;
typedef std::vector<std::vector<std::vector<std::vector<Complex>>>> matrix_4d_Complex;
typedef std::vector<std::vector<std::vector<std::vector<double>>>> matrix_4d_double;
typedef std::vector<std::vector<std::vector<Complex>>> matrix_3d_Complex;
typedef std::vector<std::vector<std::vector<double>>> matrix_3d_double;
typedef std::complex<long double> LongComplex;
typedef struct arrayWrap{
    float thisArr[8];
} arrayWrap;
typedef struct dataMap{
    short int x,y,z;
    arrayWrap coeffMap;
    short int imu,ir,inu;
    int linearMap;
    dataMap(short int thisX, short int thisY, short int thisZ,double thisir,double
    thisimu, double thisinu, arrayWrap thiscoeffMap, int thisLinearMap){
        this->x = thisX;
        this->y = thisY;
        this->z = thisZ;
        this->ir = thisir;
        this->imu = thisimu;
        this->inu = thisinu;
        this->coeffMap = thiscoeffMap;
        this->linearMap = thisLinearMap;
    }
    dataMap(){
        this->x = 0;
        this->y = 0;
        this->z = 0;
        this->ir = 0;
        this->imu = 0;
```

```
1048            this->inu = 0;
                }
1050        } dataMap;
            typedef struct coord_3d{
1052            short int x, y , z;
                coord_3d(int thisx, int thisy, int thisz){
1054                this->x = (short int) thisx;
                    this->y = (short int) thisy;
1056                this->z = (short int) thisz;
                }
1058        } coord_3d;
            class usDataClass{
1060        private:
                //parameters
1062            matrix_4d_double decorrOut;
                matrix_4d_double IBS_Product;
1064            matrix_4d_double B2;
                matrix_4d_double R01_square;
1066            matrix_4d_Complex usDataSphere;
                matrix_4d_Complex usDataCart;
1068            matrix_4d_Complex IBS_mat,IBS_mat_conv;
                matrix_4d_Complex autocorr_mat;
1070            matrix_3d_double xMat;
                matrix_3d_double yMat;
1072            matrix_3d_double zMat;
                matrix_3d_double gaussianWindow;
1074            matrix_3d_double gaussianWindow2;
                fftw_complex * gaussWindow_freq;
1076            std::vector<dataMap> scanConverionMap;
                std::vector<coord_3d> invalidCoords;
1078            std::vector<fftw_complex * > IBS;
                std::vector<fftw_complex * > autocorr;
1080            std::vector<fftw_complex * > decorr;
                std::vector<fftw_complex * > intermedVec;
1082            std::string fileName;
                std::string infoFileName;
1084            fftw_complex * maskfilt;
                std::vector<double> B2_mean;
1086            int rSize, thetaSize, phiSize, frameSize;
                int xSize,ySize,zSize;
1088            int feStreamId;
                int BufWritePtrFrames;
1090            double NumSamplesPerMm;
                int BufSizeInLines, NumLinesPerPGroup, NumLinesPerSlice, NumRangeSamples, NumSlices;
1092            int NumSlicesPerSubFrames, NumSlicesPerSweep, NumLinesPerFrame, NumFramesPerBuf;
                double phiMax, thetaMax, rMax,phiMin,thetaMin,rMin;
1094            double xMax,yMax,zMax,xMin,yMin,zMin;
                double dx,dy,dz,dr,dTheta,dPhi;
1096            double interFrameTime;
                double cartScalingFactor;
1098            double windowSigma;
                //functions
1100            // Init Parameters/buffers, only executed once, upon object creation
                void initDataSetParams();
```

```
1102        void initAllocation();
            void generateAxesMat();
1104        void generatePyramidalCoordinates();
            void generateGaussianWindow();
1106        void clearUsedVecs();
            //Read data from file
1108        float hex2Dec_SWFC(std::string hexString);
            void read_scannerDump(std::string fileName);
1110        void readInfoFile(std::string infoFileName);
            // Compute parameters
1112    public:
            std::vector<dataMap> validPyramidalCoords;
1114        usDataClass(std::string thisFileName, std::string thisInfoFileName, double thisRMin,
             double thisRMax, double thisPhiMin, double thisPhiMax, double thisThetamin, double
             thisThetaMax, double thisCartScalingFactor, double thisWindowSigma, double
            thisInterFrameTime){
                //thisRawData,startTime,thisInfoFile,thisrmin,thisrmax,thisthetamin,thisthetamax
            ,thisphimin,thisphimax,thiscartScalingFactor,thiswindowSigma,thisinterFrameTime
1116            this->fileName = thisFileName;
                this->infoFileName = thisInfoFileName;
1118            this->rMin = thisRMin;
                this->rMax = thisRMax;
1120            this->phiMin = thisPhiMin;
                this->phiMax = thisPhiMax;
1122            this->thetaMin = thisThetamin;
                this->thetaMax = thisThetaMax;
1124            this->cartScalingFactor = thisCartScalingFactor;
                this->windowSigma = thisWindowSigma;
1126            this->interFrameTime = thisInterFrameTime;
                this->readInfoFile(this->infoFileName);
1128            this->initDataSetParams();
                this->initAllocation();
1130            this->read_scannerDump(this->fileName);
                this->generateGaussianWindow();
1132            this->generateGaussianWindowType2();
                this->generateAxesMat();
1134            this->generatePyramidalCoordinates();
            }
1136        void computeCartCoordinates();
            void computeIBS();
1138        void computeAutocorr();
            void computeDecorr();
1140        void convolveIBSFourier();
            void convolveAutocorrFourier();
1142        void addNextDataSet(std::string infilename);
            // export/data debugging
1144        void exportCartUSDataToCSV(std::string filename);
            void exportFFT_Complex_Matrix(std::string outFileName, std::vector<fftw_complex*>
            outputVector, int thisFrames);
1146        void export_3D_Double_Matrix(std::string outFileName, matrix_3d_double outputVector)
            ;
            void export_4D_Complex_Matrix(std::string outFileName, matrix_4d_Complex
            outputVector, int numFrames);
1148        void displayParams();
```

```
    void getVecStat();
1150    void generateGaussianWindowType2();
    void exportFFT_Complex_3d(std::string outFileName, fftw_complex* outputVector);
1152    void export_4D_Real_Matrix(std::string outFileName, matrix_4d_double outputVector,
    int numFrames);
    // getters/setters
1154    bool get_slice_cart(char direction, int frame, int index, std::vector<std::vector<
    double>> &sliceMat);
    bool get_slice_decorr(char direction, int frame, int index, std::vector<std::vector<
    double>> &sliceMat);
1156    int getXsize(){return this->xSize;}
    int getYsize(){return this->ySize;}
1158    int getZsize(){return this->zSize;}
};
```

A.2 Decorrelation Class Source

```
//
//  SiemensSC2000.cpp
//  echoDecorr3D
//
//  Created by Peter Grimm on 3/2/19.
//  Copyright Â© 2019 Peter Grimm. All rights reserved.


#include "SiemensSC2000.hpp"
#include <string>
#include <iostream>
#include <bitset>
#include <fstream>
#include <sstream>
#include <complex>
#include <vector>
#include <cstdio>
#include "omp.h"

#define REAL 0
#define IMAG 1

//init
float usDataClass::hex2Dec_SWFC(std::string hexString){
    /* HEX2Dec_SWFC
         Convert hex string to decimal using SWFC convention with swfc = [1 0 17 1]
         this means the number is signed, and that all of the information bits come after
      the decimal place (e.g output = 0.b1b2b3...bn where n = 17)
    */
     unsigned int twoCompMask = 131071; // When XOR'd with 17 bit data bit input (inside
     32 bit unsigned int) flips bit for two's compliment
     unsigned int signedFloatMask = 3187671040;// Signed float mask: When XOR'd with the
     input 32 bit unsigned twos compliment int forms a float with exponent -1 and a 1
     sign bit, also reverses shifted sign bit eg in decimal[1][126-8][bits][padded to
     end]
     unsigned int unsignedFloatMask = 1056964608; // Unsigned float mask: When XOR'd with
      the input forms a float with exponent -1 and a 1 sign bit, also reverses shifted
     sign bit eg in decimal[1][126-8][bits][padded to end]
     float floatCorrection = .5; // Mantissa is of the form 1.b1b2b3... so when exponent
     = -1 there is an additional 2^-1 (.5) term in the float, which must be subtracted/
     added depending on sign
     //get input int
     unsigned int num = std::stoi(hexString, nullptr,16); //convert hex string to int
     // Bitwise manipulation
     if (num >> 17){ // check if signed bit is 1
         num = (twoCompMask^num)+1; // two's compliment
         num = (num<<7)^signedFloatMask; // shift information bits to proper spot for
     exponent -1 (num <<7), then xor with the signedFloatMask
         return *(float*)(&num) + floatCorrection; // dereference num, cast what the
     address points to as a float and add correction
     }
     else{
```

```
        num = (num<<7)^unsignedFloatMask; // same as above, but don't take two's compliment
        return *(float*)(&num) - floatCorrection; // .5 is here because mantissa in a float is of the form 1.b1b2b3..., with an exponent of -1, .5 must is left over
    }
}
void usDataClass::readInfoFile(std::string infoFileName){
    std::string line;
    std::ifstream fileStream;
    std::vector<int> argArray;
    int argInt;
    int startIndex;
    int currArg = 0;
    int argBaseArr [] = {16,16,10,16,16,16,16,16,16,16,16,16};
    //define array
    fileStream.open(infoFileName);
    while(std::getline(fileStream, line)){
        startIndex = (int) line.find("=");
        if (startIndex != -1){
            if(argBaseArr[currArg] == 16){
                argInt = std::stoi(line.substr(startIndex+1,line.find("\r")-1),nullptr, argBaseArr[currArg]);
                argArray.push_back(argInt);
            }
            else{
                argInt = this->NumSamplesPerMm = std::stod(line.substr(startIndex+1,line.find("\r")-1));
                argArray.push_back(argInt);
            }
            currArg++;
        }
    }

    this->feStreamId = argArray[0];
    this->BufWritePtrFrames = argArray[1];
    this->BufSizeInLines = argArray[3];
    this->NumLinesPerPGroup = argArray[4];
    this->thetaSize = argArray[5];
    this->rSize = argArray[6];
    this->phiSize = argArray[7];
    this->NumSlicesPerSubFrames = argArray[8];
    this->NumSlicesPerSweep = argArray[9];
    this->NumLinesPerFrame = argArray[10];
    this->frameSize = argArray[11];

}
void usDataClass::initAllocation(){
    this->usDataCart.assign(this->frameSize,
                            std::vector<std::vector<std::vector<Complex>>>(this->xSize,
                                                                           std::vector<std::vector<Complex>>(this->ySize,

                            std::vector<Complex>(this->zSize,0))));
    this->autocorr_mat.assign(this->frameSize-1,
```

```
                                std::vector<std::vector<std::vector<Complex>>>(this->xSize
    ,
1090                                                                                    std::vector
    <std::vector<Complex>>(this->ySize,

                                std::vector<Complex>(this->zSize,0))));
1092    this->IBS_mat.assign(this->frameSize,
                            std::vector<std::vector<std::vector<Complex>>>(this->xSize,
1094                                                                            std::vector<std
    ::vector<Complex>>(this->ySize,

                        std::vector<Complex>(this->zSize,0))));
1096    this->IBS_mat_conv.assign(this->frameSize,
                                std::vector<std::vector<std::vector<Complex>>>(this->xSize
    ,
1098                                                                                    std::vector
    <std::vector<Complex>>(this->ySize,

                                std::vector<Complex>(this->zSize,0))));
1100    this->usDataSphere.assign(this->frameSize,
                                std::vector<std::vector<std::vector<Complex>>>(this->rSize
    ,
1102                                                                                    std::vector
    <std::vector<Complex>>(this->thetaSize,

                                std::vector<Complex>(this->phiSize,0))));
1104    this->B2.assign(this->frameSize,
                                std::vector<std::vector<std::vector<double>>>(this->xSize,
1106                                                                            std::vector
    <std::vector<double>>(this->ySize,

                                std::vector<double>(this->zSize,0))));
1108    this->R01_square.assign(this->frameSize,
                        std::vector<std::vector<std::vector<double>>>(this->xSize,
1110                                                                    std::vector<std::
    vector<double>>(this->ySize,

                    std::vector<double>(this->zSize,0))));
1112    this->decorrOut.assign(this->frameSize,
                            std::vector<std::vector<std::vector<double>>>(this->xSize,
1114                                                                        std::vector<std
    ::vector<double>>(this->ySize,

                        std::vector<double>(this->zSize,0))));
1116    for(int currFrame = 0; currFrame < this->frameSize-1; currFrame++){
        this->autocorr.push_back((fftw_complex*) fftw_malloc(this->xSize * this->ySize *
     this->zSize * sizeof(fftw_complex)));
1118        this->B2_mean.push_back(0);
    }
1120    for(int currFrame = 0; currFrame < this->frameSize; currFrame++){
        this->IBS.push_back((fftw_complex*) fftw_malloc(this->xSize * this->ySize * this
    ->zSize * sizeof(fftw_complex)));
1122    }
```

```
1124 }
     void usDataClass::read_scannerDump(std::string fileName){
1126     //define params
         std::string line;
1128     int startIndex;
         int infoLines = 11;
1130     std::vector<Complex ***> dataVec;
         std::string realString;
1132     std::string imagString;
         std::ifstream fileStream;
1134     std::vector<int> argArray;
         // open file
1136     fileStream.open(fileName);
         //get info
1138     // Loop through top lines
         for(int infoIndex = 0; infoIndex < 2; infoIndex++){
1140         std::getline(fileStream, line);
         }
1142     //loop through info file data
         for(int infoIndex = 3; infoIndex < infoLines; infoIndex++){
1144         std::getline(fileStream, line);
             argArray.push_back(std::stoi(line.substr(line.find(":")+8,3),nullptr,16));
1146
         }
1148     //loop through final lines before data
         for(int infoIndex = 0; infoIndex < 3; infoIndex++){
1150         std::getline(fileStream, line);
         }
1152     std::getline(fileStream, line);
         //extract information from argArray
1154     this->frameSize = argArray[3];
         this->rSize = argArray[4];
1156     this->thetaSize = argArray[5];
         this->phiSize = argArray[6];
1158     //fill arrays
         for (int frames = 0; frames < this->frameSize; frames++){
1160         //loop through elements in matrix
             for (int z = 0; z < this->phiSize; z++){
1162             for (int q = 0; q < this->thetaSize; q++){
                     for (int  r = 0; r < this->rSize; r++){
1164                     // steps through range, then azimuth, then elevation
                         std::getline(fileStream, line);
1166                     std::stringstream in(line);
                         startIndex = (int)line.find(":");
1168                     realString = line.substr(startIndex+5,5);
                         imagString = line.substr(startIndex+14,5);
1170                     // convert to complex number using the swfc convention
                         usDataSphere[frames][r][q][z] = Complex(hex2Dec_SWFC(realString),
     hex2Dec_SWFC(imagString));
1172                 }
             }
1174     }
     }
1176 }
```

```
void usDataClass::initDataSetParams(){
    this->dr = 1/this->NumSamplesPerMm;
    this->dTheta = (abs(sin(this->thetaMax))+abs(sin(this->thetaMin)))/(this->thetaSize
    -1);
    this->dPhi = (abs(sin(this->phiMax))+abs(sin(this->phiMin)))/(this->phiSize-1);
    this->dx = this->dr*this->cartScalingFactor;
    this->dy = this->dx;
    this->dz = this->dx;
    this->yMax = this->rMax*sin(this->thetaMax);
    this->yMin = this->rMax*sin(this->thetaMin);
    this->xMax = this->rMax*sin(this->phiMax);
    this->xMin = this->rMax*sin(this->phiMin);
    this->zMax = rMax;
    this->zMin = rMin;

    this->xSize = ceil((abs(this->xMax)+abs(this->xMin))/this->dx);
    this->ySize = ceil((abs(this->yMax)+abs(this->yMin))/this->dy);
    this->zSize = ceil((abs(this->zMax)+abs(this->zMin))/this->dz);
}
void usDataClass::generateAxesMat(){
    double cumulativeX = this->xMin,cumulativeY = this->yMin,cumulativeZ = this->zMin;
    //fill xMat,yMat,zMat
    this->xMat.assign(this->xSize, std::vector<std::vector<double>>(this->ySize, std::
    vector<double>(this->zSize,0)));
    this->yMat.assign(this->xSize, std::vector<std::vector<double>>(this->ySize, std::
    vector<double>(this->zSize,0)));
    this->zMat.assign(this->xSize, std::vector<std::vector<double>>(this->ySize, std::
    vector<double>(this->zSize,0)));
        //fill xMat
        for(int curr_x = 0; curr_x < this->xSize; curr_x++){
            for(int curr_y = 0; curr_y < this->ySize; curr_y++){
                for(int curr_z = 0; curr_z < this->zSize; curr_z++){
                    xMat[curr_x][curr_y][curr_z] = cumulativeX;

                }
            }
            cumulativeX = cumulativeX + this->dx;
            //std::cout <<xMat[curr_x][0][0] <<std::endl;
        }
    for(int curr_y = 0; curr_y < this->ySize; curr_y++){
        for(int curr_x = 0; curr_x < this->xSize; curr_x++){
            for(int curr_z = 0; curr_z < this->zSize; curr_z++){
                yMat[curr_x][curr_y][curr_z] = cumulativeY;
            }
        }
        cumulativeY = cumulativeY + this->dy;
        //std::cout <<xMat[0][curr_y][0] <<std::endl;
    }
    //fill zMat, make z outer loop for less flops
    for(int curr_z = 0; curr_z < this->zSize; curr_z++){
        for(int curr_y = 0; curr_y < this->ySize; curr_y++){
            for(int curr_x = 0; curr_x < this->xSize; curr_x++){
                zMat[curr_x][curr_y][curr_z] = cumulativeZ;

```

```
            }
        }
        cumulativeZ = cumulativeZ + this->dz;
        //std::cout <<xMat[0][0][curr_z] <<std::endl;
    }
}
void usDataClass::generatePyramidalCoordinates(){
    //generate validPyramidalCoords
    //dataMap tempDataMap(0,0,0);
    double muMax = sin(this->thetaMax);
    double muMin = -muMax;
    double nuMax = sin(this->phiMax);
    double nuMin = -muMax;
    double dnu = (abs(nuMin)+abs(nuMax))/(this->phiSize-1);
    double dmu = (abs(muMin)+abs(muMax))/(this->thetaSize-1);
    double Lmu,Lnu,LR,drmLR,dmumLmu,dnumLnu;
    int imu,inu,iR;
    int linearPoint;
    arrayWrap coeffArray;
    //need iR,imu,inu
    double k,R0,mu0,nu0,testVal;
    for(int curr_x = 0; curr_x < this->xSize; curr_x++){
        for(int curr_y = 0; curr_y < this->ySize; curr_y++){
            for(int curr_z = 0; curr_z < this->zSize; curr_z++){
                k = pow(yMat[curr_x][curr_y][curr_z],2)+pow(zMat[curr_x][curr_y][curr_z
],2);
                R0 = sqrt(k+pow(xMat[curr_x][curr_y][curr_z],2));
                mu0 = yMat[curr_x][curr_y][curr_z]/sqrt(k);
                nu0 = xMat[curr_x][curr_y][curr_z]/sqrt(pow(R0,2)-pow(yMat[curr_x][
curr_y][curr_z],2));
                linearPoint = curr_x + this->ySize * (curr_y + this->zSize * curr_z);
                if((R0 >= this->rMin) && (R0 <= (this->rMax-this->dr)) && (mu0 >= muMin)
 && (mu0 <= (muMax-dmu)) && (nu0 >= nuMin) && (nu0 <= (nuMax-dmu))){
                    imu = (int) floor((mu0 - muMin)/dmu);
                    testVal = ((imu*dmu)+muMin) - dmu;
                    Lmu = mu0 - testVal;
                    inu = (int) floor((nu0 - nuMin)/dnu) -dnu;
                    Lnu = nu0 - ((inu*dnu)+nuMin)+muMax;
                    iR = (int) floor((R0 - this->rMin)/this->dr) ;
                    LR = R0 - iR*this->dr;
                    drmLR = this->dr-LR;
                    dmumLmu = dmu-Lmu;
                    dnumLnu = dnu-Lnu;
                    coeffArray.thisArr[0] = (float) drmLR*dmumLmu*dnumLnu;
                    coeffArray.thisArr[1] = (float) LR*dmumLmu*dnumLnu;
                    coeffArray.thisArr[2] = (float) drmLR*Lmu*dnumLnu;
                    coeffArray.thisArr[3] = (float) drmLR*dmumLmu*Lnu;
                    coeffArray.thisArr[4] = (float) LR*dmumLmu*Lnu;
                    coeffArray.thisArr[5] = (float) drmLR*Lmu*Lnu;
                    coeffArray.thisArr[6] = (float) LR*Lmu*dnumLnu;
                    coeffArray.thisArr[7] = (float) LR*Lmu*Lnu;
                    this->validPyramidalCoords.push_back(dataMap(curr_x,curr_y,curr_z,iR
,imu,inu,coeffArray,linearPoint));
                }
```

```
                else{
                    this->invalidCoords.push_back(coord_3d(curr_x,curr_y,curr_z));
                }

            }
        }
    }
}
void usDataClass::computeCartCoordinates(){
    dataMap currPoint;
    for(int volIndex = 0; volIndex < this->frameSize; volIndex++ ){
        for(int vecIndex = 0; vecIndex < this->validPyramidalCoords.size()-1; vecIndex
    ++){
            currPoint = this->validPyramidalCoords[vecIndex];
            this->usDataCart[volIndex][currPoint.x][currPoint.y][currPoint.z] =
                this->usDataSphere[volIndex][currPoint.ir][currPoint.imu][currPoint.inu]
     * ((double)currPoint.coeffMap.thisArr[0]) +
                this->usDataSphere[volIndex][currPoint.ir+1][currPoint.imu][currPoint.
    inu] * ((double)currPoint.coeffMap.thisArr[1]) +
                this->usDataSphere[volIndex][currPoint.ir][currPoint.imu+1][currPoint.
    inu] * ((double)currPoint.coeffMap.thisArr[2]) +
                this->usDataSphere[volIndex][currPoint.ir][currPoint.imu][currPoint.inu
    +1] * ((double)currPoint.coeffMap.thisArr[3]) +
                this->usDataSphere[volIndex][currPoint.ir+1][currPoint.imu][currPoint.
    inu+1] * ((double)currPoint.coeffMap.thisArr[4]) +
                this->usDataSphere[volIndex][currPoint.ir][currPoint.imu+1][currPoint.
    inu+1] * ((double)currPoint.coeffMap.thisArr[5]) +
                this->usDataSphere[volIndex][currPoint.ir+1][currPoint.imu+1][currPoint.
    inu] * ((double)currPoint.coeffMap.thisArr[6]) +
                this->usDataSphere[volIndex][currPoint.ir+1][currPoint.imu+1][currPoint.
    inu] * ((double)currPoint.coeffMap.thisArr[7]);
        }
    }
}
void usDataClass::computeAutocorr(){
    dataMap currPoint;
    for(int volIndex = 0; volIndex < this->frameSize-1; volIndex++ ){
        for(int vecIndex = 0; vecIndex < this->validPyramidalCoords.size()-1; vecIndex
    ++){
            currPoint = this->validPyramidalCoords[vecIndex];
            this->autocorr_mat[volIndex][currPoint.x][currPoint.y][currPoint.z] = this->
    usDataCart[volIndex][currPoint.x][currPoint.y][currPoint.z]*std::conj(this->
    usDataCart[volIndex+1][currPoint.x][currPoint.y][currPoint.z]) ;
        }
    }
}
void usDataClass::computeIBS(){
    dataMap currPoint;
    // compute ibs = vol(t=t0)*conj(vol(t=t0+delta t))
    for(int volIndex = 0; volIndex < this->frameSize; volIndex++ ){
        //this->usData_cart_linear.push_back(new Complex);
        for(int vecIndex = 0; vecIndex < this->validPyramidalCoords.size()-1; vecIndex
    ++){
            currPoint = this->validPyramidalCoords[vecIndex];
```

```
1318            this->IBS_mat[volIndex][currPoint.x][currPoint.y][currPoint.z] = std::conj(
    this->usDataCart[volIndex][currPoint.x][currPoint.y][currPoint.z])*this->usDataCart
    [volIndex][currPoint.x][currPoint.y][currPoint.z] ;
        }
1320    }
    //std::string thisFileName = "/Users/peter/Documents/testdata/IBStest14.txt";
1322    //export_4D_Complex_Matrix(thisFileName, IBS_mat,2);
}
1324 void usDataClass::convolveIBSFourier(){
    int thisIndex = 0;
1326    for(int currFrame = 0; currFrame < this->frameSize; currFrame++){
        for(int curr_x = 0; curr_x < this->xSize; curr_x++){
1328            for(int curr_y = 0; curr_y < this->ySize; curr_y++){
                for(int curr_z = 0; curr_z < this->zSize; curr_z++){
1330                this->IBS[currFrame][thisIndex][REAL] = this->IBS_mat[currFrame][
    curr_x][curr_y][curr_z].real();
                    this->IBS[currFrame][thisIndex][IMAG] = this->IBS_mat[currFrame][
    curr_x][curr_y][curr_z].imag();
1332                thisIndex++;
                }
1334            }
        }
1336        thisIndex = 0;
        // 5 threads is the sweet spot for reasons unknown
1338        fftw_plan_with_nthreads(5);
        fftw_plan forwardFFTPlan = fftw_plan_dft_3d(xSize, ySize, zSize, IBS[currFrame],
     IBS[currFrame], FFTW_FORWARD, FFTW_ESTIMATE);
1340        fftw_execute(forwardFFTPlan);
    }
1342    thisIndex = 0;
    //std::string thisFileName = "/Users/peter/Documents/testdata/IBStest12345.txt";
1344    //export_4D_Complex_Matrix(thisFileName, IBS_mat, 2);
    for(int currFrame = 0; currFrame < this->frameSize; currFrame++){
1346        //this->IBS.push_back((fftw_complex*) fftw_malloc(this->xSize * this->ySize *
    this->zSize * sizeof(fftw_complex)));
        for(int curr_x = 0; curr_x < this->xSize; curr_x++){
1348            for(int curr_y = 0; curr_y < this->ySize; curr_y++){
                for(int curr_z = 0; curr_z < this->zSize; curr_z++){
1350                this->IBS[currFrame][thisIndex][IMAG] = this->gaussianWindow[curr_x
    ][curr_y][curr_z] * this->IBS[currFrame][thisIndex][IMAG];
                    this->IBS[currFrame][thisIndex][REAL] = this->gaussianWindow[curr_x
    ][curr_y][curr_z] * this->IBS[currFrame][thisIndex][REAL];
1352                thisIndex++;
                }
1354            }
        }
1356        thisIndex = 0;
        fftw_plan_with_nthreads(5);
1358        fftw_plan forwardFFTPlan = fftw_plan_dft_3d(xSize, ySize, zSize, IBS[currFrame],
     IBS[currFrame], FFTW_BACKWARD, FFTW_ESTIMATE);
        fftw_execute(forwardFFTPlan);
1360    }
    std::complex<long double> tempComplex1, tempComplex2, testVal3;
1362    long double testVal1, testVal2;
```

```
    long double b2_mean_local;
    long double tempDouble;
    for(int currFrame = 0; currFrame < this->frameSize-1; currFrame++){
        b2_mean_local = 0;
        for(auto currPoint : this->validPyramidalCoords){
            tempComplex1 =  Complex(this->IBS[currFrame][currPoint.linearMap][REAL],this
->IBS[currFrame][currPoint.linearMap][IMAG]);
            testVal1 =this->IBS[currFrame+1][currPoint.linearMap][REAL];
            testVal2 =this->IBS[currFrame+1][currPoint.linearMap][IMAG];
            testVal3 = Complex(testVal1,testVal2);
            testVal3 = testVal3*std::conj(testVal3);
            tempComplex1 *= std::conj(tempComplex1);
            tempComplex2 *= Complex(this->IBS[currFrame+1][currPoint.linearMap][REAL],
this->IBS[currFrame+1][currPoint.linearMap][IMAG]);
            tempComplex2 *= std::conj(tempComplex2);
            tempDouble = tempComplex1.real()*testVal3.real();
            b2_mean_local += tempDouble;
            this->B2[currFrame][currPoint.x][currPoint.y][currPoint.z] = tempDouble;
        }
        this->B2_mean[currFrame] = (double) b2_mean_local;

    }
    //std::string thisFileName = "/Users/peter/Documents/testdata/B2test3.txt";
    //export_4D_Real_Matrix(thisFileName, this->B2,2);
}
void usDataClass::convolveAutocorrFourier(){
    int thisIndex = 0;
    Complex currPoint;
    for(int currFrame = 0; currFrame < this->frameSize-1; currFrame++){
        for(int curr_x = 0; curr_x < this->xSize; curr_x++){
            for(int curr_y = 0; curr_y < this->ySize; curr_y++){
                for(int curr_z = 0; curr_z < this->zSize; curr_z++){
                    this->autocorr[currFrame][thisIndex][1] = this->autocorr_mat[
currFrame][curr_x][curr_y][curr_z].real();
                    this->autocorr[currFrame][thisIndex][0] = this->autocorr_mat[
currFrame][curr_x][curr_y][curr_z].imag();
                    thisIndex++;
                }
            }
        }
        thisIndex = 0;
        // spawn threads for fftw
        // 5 threads is the sweet spot for reasons unknown
        fftw_plan_with_nthreads(5);
        fftw_plan forwardFFTPlan = fftw_plan_dft_3d(xSize, ySize, zSize, autocorr[
currFrame], autocorr[currFrame], FFTW_FORWARD, FFTW_ESTIMATE);
        fftw_execute(forwardFFTPlan);
    }
    //std::string thisFileName = "/Users/peter/Documents/testdata/IBStest1234.txt";
    //exportFFT_Complex_Matrix(thisFileName, IBS,2);
    //std::string thisFileName = "/Users/peter/Documents/testdata/IBStest1234.txt";
    //export_4D_Complex_Matrix(fileName, autocorr_mat, 2)
    thisIndex = 0;
    for(int currFrame = 0; currFrame < this->frameSize-1; currFrame++){
```

```
1412            //this->autocorr.push_back((fftw_complex*) fftw_malloc(this->xSize * this->ySize
        * this->zSize * sizeof(fftw_complex)));
                for(int curr_x = 0; curr_x < this->xSize; curr_x++){
1414                for(int curr_y = 0; curr_y < this->ySize; curr_y++){
                        for(int curr_z = 0; curr_z < this->zSize; curr_z++){
1416                        this->autocorr[currFrame][thisIndex][1] = this->gaussianWindow[
        curr_x][curr_y][curr_z] * this->autocorr[currFrame][thisIndex][1];
                            this->autocorr[currFrame][thisIndex][0] = this->gaussianWindow[
        curr_x][curr_y][curr_z] * this->autocorr[currFrame][thisIndex][0];
1418                        //currPoint = Complex(this->autocorr[currFrame][thisIndex][REAL],
        this->autocorr[currFrame][thisIndex][IMAG])*
                            //              Complex(gaussWindow_freq[thisIndex][REAL],
        gaussWindow_freq[thisIndex][IMAG]);
1420                        //this->autocorr[currFrame][thisIndex][1] = currPoint.imag();
                            //this->autocorr[currFrame][thisIndex][0] = currPoint.real();
1422                        thisIndex++;
                        }
1424                }
                }
1426            thisIndex = 0;
                fftw_plan_with_nthreads(5);
1428            fftw_plan forwardFFTPlan = fftw_plan_dft_3d(xSize, ySize, zSize, autocorr[
        currFrame], autocorr[currFrame], FFTW_BACKWARD, FFTW_ESTIMATE);
                fftw_execute(forwardFFTPlan);
1430            // fill R01_square with square of autocorr

1432        }
        Complex tempComplex;
1434    for(int currFrame = 0; currFrame < this->frameSize-1; currFrame++){
            for(auto currPoint : this->validPyramidalCoords){
1436            tempComplex =  Complex(this->autocorr[currFrame][currPoint.linearMap][REAL],
        this->autocorr[currFrame][currPoint.linearMap][IMAG]);
                tempComplex *= std::conj(tempComplex);
1438            this->R01_square[currFrame][currPoint.x][currPoint.y][currPoint.z] =
        tempComplex.real();
            }
1440    }
//    std::string thisFileName = "/Users/peter/Documents/testdata/autocorr6.txt";
1442//    exportFFT_Complex_Matrix(thisFileName, autocorr,1);
}
1444void usDataClass::generateGaussianWindow(){
    this->gaussianWindow.assign(this->xSize, std::vector<std::vector<double>>(this->
    ySize, std::vector<double>(this->zSize,0)));
1446    double sigmaAdj = this->windowSigma/this->dx;
    double sigfaz = this->xSize/(M_PI * 2 * sigmaAdj);
1448    int xSizeAdj = this->xSize -1;
    int ySizeAdj = this->ySize -1;
1450    int zSizeAdj = this->zSize -1;
    double tempX,tempY,tempZ,tempTot;
1452    double sigSquare =pow(sigfaz,2);
    double totalVal = 0;
1454    for(int curr_x = 0; curr_x <= this->xSize/2; curr_x++){
        for(int curr_y = 0; curr_y <= this->ySize/2; curr_y++){
1456            for(int curr_z = 0; curr_z <= this->zSize/2; curr_z++){
```

```
                tempX = pow(curr_x,2);
                tempY = pow(curr_y,2);
                tempZ = pow(curr_z,2);
                tempTot = exp(-((tempX+tempY+tempZ)/2/sigSquare));
                this->gaussianWindow[curr_x][curr_y][curr_z] = tempTot; // corner 000
                this->gaussianWindow[curr_x][curr_y][zSizeAdj  - curr_z] = tempTot; //
    corner 001
                this->gaussianWindow[curr_x][ySizeAdj - curr_y ][curr_z] = tempTot; //
    corner 010
                this->gaussianWindow[curr_x][ySizeAdj - curr_y][zSizeAdj - curr_z] =
    tempTot; // corner 011
                this->gaussianWindow[xSizeAdj - curr_x][curr_y][curr_z] = tempTot; //
    corner 100
                this->gaussianWindow[xSizeAdj - curr_x][curr_y][zSizeAdj - curr_z] =
    tempTot; //corner 101
                this->gaussianWindow[xSizeAdj - curr_x][ySizeAdj - curr_y][curr_z] =
    tempTot; // corner 110
                this->gaussianWindow[xSizeAdj - curr_x][ySizeAdj - curr_y][zSizeAdj -
    curr_z] = tempTot; // corner 111
                totalVal = totalVal + 8* this->gaussianWindow[curr_x][curr_y][curr_z];
            }
        }
    }
    for(int curr_x = 0; curr_x < this->xSize; curr_x++){
        for(int curr_y = 0; curr_y < this->ySize; curr_y++){
            for(int curr_z = 0; curr_z < this->zSize; curr_z++){
                this->gaussianWindow[curr_x][curr_y][curr_z] = this->gaussianWindow[
    curr_x][curr_y][curr_z]/totalVal;
            }
        }
    }
    //std::string thisFileName = "/Users/peter/Documents/testdata/testGauss11.txt";
    //export_3D_Double_Matrix(thisFileName, gaussianWindow);
}
void usDataClass::generateGaussianWindowType2(){
    this->gaussianWindow2.assign(this->xSize, std::vector<std::vector<double>>(this->
    ySize, std::vector<double>(this->zSize,0)));
    this->gaussWindow_freq = ((fftw_complex*) fftw_malloc(this->xSize * this->ySize *
    this->zSize * sizeof(fftw_complex)));
    double sigmaAdj = this->windowSigma/this->dx;
    int xMid = (xSize/2);
    int yMid = (ySize/2);
    int zMid = (zSize/2);
    double tempX,tempY,tempZ,tempTot;
    double sigSquare =pow(sigmaAdj,2);
    double totalVal = 0;
    for(int curr_x = 0; curr_x < this->xSize; curr_x++){
        for(int curr_y = 0; curr_y < this->ySize; curr_y++){
            for(int curr_z = 0; curr_z < this->zSize; curr_z++){
                tempX = pow(curr_x-xMid,2);
                tempY = pow(curr_y-yMid,2);
                tempZ = pow(curr_z-zMid,2);
                tempTot = exp(-((tempX+tempY+tempZ)/2/sigSquare));
                this->gaussianWindow2[curr_x][curr_y][curr_z] = tempTot; // corner 000
```

```
                totalVal = totalVal + this->gaussianWindow2[curr_x][curr_y][curr_z];
1502            }
            }
1504    }
    for(int curr_x = 0; curr_x < this->xSize; curr_x++){
1506        for(int curr_y = 0; curr_y < this->ySize; curr_y++){
            for(int curr_z = 0; curr_z < this->zSize; curr_z++){
1508            this->gaussianWindow2[curr_x][curr_y][curr_z] = this->gaussianWindow2[
    curr_x][curr_y][curr_z]/totalVal;
            }
1510        }
    }
1512    int thisIndex = 0;
    for(int curr_x = 0; curr_x < this->xSize; curr_x++){
1514        for(int curr_y = 0; curr_y < this->ySize; curr_y++){
            for(int curr_z = 0; curr_z < this->zSize; curr_z++){
1516            gaussWindow_freq[thisIndex][REAL] = this->gaussianWindow2[curr_x][curr_y
    ][curr_z] ;
                thisIndex++;
1518            }
            }
1520    }
    fftw_plan_with_nthreads(5);
1522    fftw_plan forwardFFTPlan = fftw_plan_dft_3d(xSize, ySize, zSize, this->
    gaussWindow_freq, this->gaussWindow_freq, FFTW_FORWARD, FFTW_ESTIMATE);
    fftw_execute(forwardFFTPlan);
1524//    std::string thisFileName = "/Users/peter/Documents/testdata/testGauss18.txt";
//    export_3D_Double_Matrix(thisFileName, gaussianWindow2);
1526//    thisFileName = "/Users/peter/Documents/testdata/testGauss17.txt";
//    exportFFT_Complex_3d(thisFileName, gaussWindow_freq);
1528}
void usDataClass::computeDecorr(){
1530    double R00,R11,B2;
    double R01,currDecorr;
1532    double outDecorr1;
    double meanB2 = 0;
1534    double B2Tot = 0;
    int arrLength = this->xSize * this->ySize * this->zSize;
1536//    for (int curr_frame = 0; curr_frame < this->frameSize-1; curr_frame++){
//        for(int index = 0; index < arrLength; index++){
1538//            B2Tot += this->B2[][][]
//        }
1540//    }
    //meanB2 = B2Tot/arrLength;
1542//    for (int curr_frame = 0; curr_frame < this->frameSize-1; curr_frame++){
//        this->decorr.push_back((fftw_complex*) fftw_malloc(this->xSize * this->ySize *
     this->zSize * sizeof(fftw_complex)));
1544//        for(int index = 0; index < arrLength; index++){
//            R00 = this->IBS[curr_frame][index][0];
1546//            R11 = this->IBS[curr_frame+1][index][0];
//            // (a+bi)(c+di) = ac - bd +adi + cbi = (ac-bd) + (ad + cb)i
1548//            B2 = R00*R11;
//            R01 = Complex(this->autocorr[curr_frame][index][0],this->autocorr[
    curr_frame][index][1])*std::conj(Complex(this->autocorr[curr_frame][index][0],this
```

```
    ->autocorr[curr_frame][index][1]));
//              currDecorr = .002*(B2-R01)/(B2+meanB2)/this->interFrameTime;
//              this->decorr[curr_frame][index][0] = currDecorr.real();
//          }
//      }
    Complex tempComplex;
    for(int currFrame = 0; currFrame < this->frameSize-1; currFrame++){
        for(auto currPoint : this->validPyramidalCoords){
            tempComplex =  Complex(this->autocorr[currFrame][currPoint.linearMap][REAL],
    this->autocorr[currFrame][currPoint.linearMap][IMAG]);
            tempComplex *= std::conj(tempComplex);
            B2 = this->B2[currFrame][currPoint.x][currPoint.y][currPoint.z];
            R01 = this->R01_square[currFrame][currPoint.x][currPoint.y][currPoint.z];
            outDecorr1 = .002*(B2-R01)/(B2+this->B2_mean[currFrame])/this->
    interFrameTime;
            this->decorrOut[currFrame][currPoint.x][currPoint.y][currPoint.z] =
    outDecorr1;
        }
    }
    //std::string thisFileName = "/Users/peter/Documents/testdata/decorrtest12.txt";
    //export_4D_Real_Matrix(thisFileName , decorrOut,2);
}
void usDataClass::addNextDataSet(std::string infilename){
   this->read_scannerDump(infilename);
}
//export/debugging
void usDataClass::exportFFT_Complex_Matrix(std::string outFileName, std::vector<
    fftw_complex*> outputVector,int thisFrames){
    int curr_index = 0;
    std::ofstream myfile;
    myfile.open (outFileName);
    myfile << "Real" << " , " <<  "Imag" << "\n";
    for(int curr_frame = 0; curr_frame < thisFrames; curr_frame++){
        for(int curr_x = 0; curr_x < this->xSize; curr_x++){
            for(int curr_y = 0; curr_y < this->ySize; curr_y++){
                for(int curr_z = 0; curr_z < this->zSize; curr_z++){
                    myfile << outputVector[curr_frame][curr_index][0] << "\t" <<
    outputVector[curr_frame][curr_index][1] << "\n";
                    curr_index++;
                }
            }
        }
        curr_index = 0;
    }
    myfile.close();
}
void usDataClass::export_3D_Double_Matrix(std::string outFileName, matrix_3d_double
    outputVector){
    std::ofstream myfile;
    myfile.open (outFileName);
    myfile << "Data" << "\n";
    for(int curr_x = 0; curr_x < this->xSize; curr_x++){
        for(int curr_y = 0; curr_y < this->ySize; curr_y++){
            for(int curr_z = 0; curr_z < this->zSize; curr_z++){
```

```
                myfile << outputVector[curr_x][curr_y][curr_z] << "\n";
1598            }
            }
1600    }
    myfile.close();
1602 }
void usDataClass::export_4D_Complex_Matrix(std::string outFileName, matrix_4d_Complex
    outputVector, int numFrames){
1604    std::ofstream myfile;
    myfile.open (outFileName);
1606    myfile << "Real" << " , " <<  "Imag" << "\n";
    for(int curr_frame = 0; curr_frame < numFrames; curr_frame++){
1608        for(int curr_x = 0; curr_x < this->xSize; curr_x++){
            for(int curr_y = 0; curr_y < this->ySize; curr_y++){
1610                for(int curr_z = 0; curr_z < this->zSize; curr_z++){
                    myfile << outputVector[curr_frame][curr_x][curr_y][curr_z].real() <<
     "\t" <<  outputVector[curr_frame][curr_x][curr_y][curr_z].imag() << "\n";
1612                }
            }
1614        }
    }
1616    myfile.close();
}
1618 void usDataClass::export_4D_Real_Matrix(std::string outFileName, matrix_4d_double
    outputVector,int numFrames){
    std::ofstream myfile;
1620    myfile.open (outFileName);
    myfile << "Data" << "\n";
1622    for(int curr_frame = 0; curr_frame < numFrames; curr_frame++){
        for(int curr_x = 0; curr_x < this->xSize; curr_x++){
1624            for(int curr_y = 0; curr_y < this->ySize; curr_y++){
                for(int curr_z = 0; curr_z < this->zSize; curr_z++){
1626                myfile << outputVector[curr_frame][curr_x][curr_y][curr_z] << "\n";
                }
1628            }
        }
1630    }
    myfile.close();
1632 }
void usDataClass::exportCartUSDataToCSV(std::string filename){
1634    std::ofstream myfile;
    myfile.open (filename);
1636
    for(int curr_frame = 0; curr_frame < this->frameSize; curr_frame++){
1638        for(int curr_x = 0; curr_x < this->xSize; curr_x++){
            for(int curr_y = 0; curr_y < this->ySize; curr_y++){
1640                for(int curr_z = 0; curr_z < this->zSize; curr_z++){
                    myfile << this->usDataCart[curr_frame][curr_x][curr_y][curr_z].real
    () << "," << this->usDataCart[curr_frame][curr_x][curr_y][curr_z].imag() << "\n";
1642                }
            }
1644        }
    }
1646    myfile.close();
```

```
}
void usDataClass::getVecStat(){
    std::cout << "the vector length is: " << validPyramidalCoords.size() << std::endl;
    double maxX = 0;
    double maxY = 0;
    double maxZ = 0;
    double minX = 100;
    double minY = 100;
    double minZ = 100;
    for(int vecIndex = 0; vecIndex < this->validPyramidalCoords.size()-1; vecIndex++){
        // get max value
        if (this->validPyramidalCoords[vecIndex].x > maxX ){
            maxX =this->validPyramidalCoords[vecIndex].x;
        }
        //get min value
        if (this->validPyramidalCoords[vecIndex].x < minX ){
            minX =this->validPyramidalCoords[vecIndex].x;
        }
        //get max value
        if (this->validPyramidalCoords[vecIndex].y > maxY ){
            maxY =this->validPyramidalCoords[vecIndex].y;
        }
        // get min
        if (this->validPyramidalCoords[vecIndex].y < minY ){
            minY =this->validPyramidalCoords[vecIndex].y;
        }
        // get max
        if (this->validPyramidalCoords[vecIndex].z > maxZ ){
            maxZ =this->validPyramidalCoords[vecIndex].z;
        }
        // get min
        if (this->validPyramidalCoords[vecIndex].z < minZ ){
            minZ =this->validPyramidalCoords[vecIndex].z;
        }
    }
    std::cout << "Max X: " << maxX << std::endl;
    std::cout << "Max Y: " << maxY << std::endl;
    std::cout << "Max z: " << maxZ << std::endl;
    std::cout << "Min X: " << minX << std::endl;
    std::cout << "Min Y: " << minY << std::endl;
    std::cout << "Min z: " << minZ << std::endl;
}
void usDataClass::displayParams(){
    //displays parameters for troubleshooting
    std::cout << "dr = " << this->dr << std::endl;
    std::cout << "dTheta = " << this->dTheta << std::endl;
    std::cout << "dPhi = " << this->dPhi << std::endl;
    std::cout << "dx = " << this->dx << std::endl;
    std::cout << "dy = " << this->dy << std::endl;
    std::cout << "dz = " << this->dz << std::endl;
    std::cout << "xMax = " << this->xMax << std::endl;
    std::cout << "xMin = " << this->xMin << std::endl;
    std::cout << "yMax = " << this->yMax << std::endl;
    std::cout << "yMin = " << this->yMin << std::endl;
```

```
    std::cout << "zMax = " << this->zMax << std::endl;
    std::cout << "zMin = " << this->zMin << std::endl;
    std::cout << "x size = " << this->xSize << std::endl;
    std::cout << "y size = " << this->ySize << std::endl;
    std::cout << "z size = " << this->zSize << std::endl;
    std::cout << "r size = " << this->xSize << std::endl;
    std::cout << "theta size = " << this->thetaSize << std::endl;
    std::cout << "phi size = " << this->phiSize << std::endl;
}
void usDataClass::exportFFT_Complex_3d(std::string outFileName, fftw_complex*
    outputVector){
    std::ofstream myfile;
    int curr_index = 0;
    myfile.open (outFileName);
    myfile << "Real" << " , " <<  "Imag" << "\n";
        for(int curr_x = 0; curr_x < this->xSize; curr_x++){
            for(int curr_y = 0; curr_y < this->ySize; curr_y++){
                for(int curr_z = 0; curr_z < this->zSize; curr_z++){
                    myfile << outputVector[curr_index][0] << "\t" <<  outputVector[
    curr_index][1] << "\n";
                    curr_index++;
                }
            }
        }
    myfile.close();
}
bool usDataClass::get_slice_cart(char direction, int frame, int index, std::vector<std::
    vector<double>> &sliceMat){
    /*
       Get Slice of cartesian data
     Parameters: direction
       'x': get a slice at x =index, first dim of the vector should be the length of y,
     second dim is length of z
       'y': get a slice at y =index, first dim of the vector should be the length of x,
     second dim is length of z
       'z': get a slice at z =index, first dim of the vector should be the length of x,
     second dim is length of y
     */
    int vecXSize = (int) sliceMat.size();
    int vecYSize = (int) sliceMat[0].size();
    switch(direction){
    //x slice
    case 'x':
            if((this->ySize == vecXSize || this->zSize == vecYSize) && index < this->
    xSize){
                for(int curr_y = 0; curr_y < this->ySize; curr_y++){
                    for(int curr_z = 0; curr_z < this->zSize; curr_z++){
                        sliceMat[curr_y][curr_z] = std::abs(this->usDataCart[frame][
    index][curr_y][curr_z]);
                    }
                }
                return true;
            }
            else{
```

```
                return false;
1748            }
        //y slice
1750    case 'y':
            if((this->xSize == vecXSize || this->zSize == vecYSize) && index < this->
    ySize){
1752                for(int curr_x = 0; curr_x < this->xSize; curr_x++){
                    for(int curr_z = 0; curr_z < this->zSize; curr_z++){
1754                    sliceMat[curr_x][curr_z] = std::abs(this->usDataCart[frame][
    curr_x][index][curr_z]);
                    }
1756                }
                return true;
1758            }
            else{
1760            return false;
            }
1762    //z slice
    case 'z':
1764         if((this->xSize == vecXSize || this->ySize == vecYSize) && index < this->
    zSize){
                for(int curr_x = 0; curr_x < this->xSize; curr_x++){
1766                for(int curr_y = 0; curr_y < this->ySize; curr_y++){
                    sliceMat[curr_x][curr_y] = std::abs(this->usDataCart[frame][
    curr_x][curr_y][index]);
1768                }
                }
1770            return true;
            }
1772        else{
                return false;
1774        }
    default:
1776        return false;
    }
1778}
bool usDataClass::get_slice_decorr(char direction, int frame, int index, std::vector<std
    ::vector<double>> &sliceMat){
1780    /*
     Get Slice of cartesian data
1782     Parameters: direction
     'x': get a slice at x =index, first dim of the vector should be the length of y,
    second dim is length of z
1784     'y': get a slice at y =index, first dim of the vector should be the length of x,
    second dim is length of z
     'z': get a slice at z =index, first dim of the vector should be the length of x,
    second dim is length of y
1786     */

1788    int vecXSize = (int) sliceMat.size();
    int vecYSize = (int) sliceMat[0].size();
1790    switch(direction){
            //x slice
1792        case 'x':
```

```
            if((this->ySize == vecXSize || this->zSize == vecYSize) && index < this->
    xSize){
1794                for(int curr_y = 0; curr_y < this->ySize; curr_y++){
                        for(int curr_z = 0; curr_z < this->zSize; curr_z++){
1796                        sliceMat[curr_y][curr_z] = std::abs(this->decorrOut[frame][index
    ][curr_y][curr_z]);
                        }
1798                }
                    return true;
1800            }
                else{
1802                return false;
                }
1804            //y slice
            case 'y':
1806            if((this->xSize == vecXSize || this->zSize == vecYSize) && index < this->
    ySize){
                    for(int curr_x = 0; curr_x < this->xSize; curr_x++){
1808                    for(int curr_z = 0; curr_z < this->zSize; curr_z++){
                            sliceMat[curr_x][curr_z] = std::abs(this->decorrOut[frame][
    curr_x][index][curr_z]);
1810                    }
                    }
1812                return true;
            }
1814            else{
                    return false;
1816            }
                //z slice
1818        case 'z':
            if((this->xSize == vecXSize || this->ySize == vecYSize) && index < this->
    zSize){
1820                for(int curr_x = 0; curr_x < this->xSize; curr_x++){
                        for(int curr_y = 0; curr_y < this->ySize; curr_y++){
1822                        sliceMat[curr_x][curr_y] = std::abs(this->decorrOut[frame][
    curr_x][curr_y][index]);
                        }
1824                }
                    return true;
1826            }
                else{
1828                return false;
                }
1830        default:
            return false;
1832    }
}
```

A.3 RF Interface Header

```
/**
 *  @file     RFSerialInterface.h
 *  @author Grimm
 *  @date     5/10/2018
 *  @version 1.0
 *
 *  @brief Functions for gathering data from AngioDynamics Rita 1500x serial line.
 *
 *  @section DESCRIPTION
 *
 *  These functions create a Win32 serial handle and process serial data from the RF
    generator.
 *  There are two sets of functions, one with a seperated handle building and read
    function,
 *  and one where they are combined. The combined call takes a slightly longer time to
    execute.
 *  The seperate functions present a significant delay however.
 *
 *  The RF generator needs to display temperature values on the device before the data
    is sent
 *  TODO: temperature values do not get transferred properly immediately upon turning
    off the current
 *
 */

#ifndef RFSerialInterface_H_INCLUDED
#define RFSerialInterface_H_INCLUDED

#include <iostream>
#include "Windows.h"
#include <atlstr.h>
#include <bitset>
#include "time.h"
#include <fstream>

//
    ////////////////////////////////////////////////////////////////////////////////////////

/// <summary> Serial data struct.
///       Stores all current data available in the serial stream. </summary>
///
/// <remarks> Peter Grimm, 7/12/2018. </remarks>
//
    ////////////////////////////////////////////////////////////////////////////////////////


struct SerialData{
  /** Temperature 1 in celcius, the value marked as 1 on the RF generator front panel,
    default value is -1**/
  double t1;
  /** Temperature 2 in celcius, the value marked as 2 on the RF generator front panel,
    default value is -1**/
```

```
    double t2;
1042 /** Temperature 3 in celcius, the value marked as 3 on the RF generator front panel,
       default value is -1**/
    double t3;
1044 /** Temperature 4 in celcius, the value marked as 4 on the RF generator front panel,
       default value is -1**/
    double t4;
1046 /** The current temperature setting of the RF generator, default value is -1**/
    double currentPower;
1048 /** Timer, default value is -1**/
    double timer;
1050 /** deliveredPower, current power being delivered to tissue , default value is -1**/
    double deliveredPower;
1052 /** Efficiency, related to impedance and delivered power by some measure, unknown
       meaning behind numbers, possibly percent, default value is -1**/
    double efficiency;
1054 /** Amount of time for ablation, counts up to the desired value , default value is -1*
       */
    double RFTime;
1056 /** Target Temperature in celcius, Set on panel, default value is -1**/
    double targetTemp;
1058 /** Indicates the current mode, default value is -1, TODO**/
    double modeIndicator;
1060 /** Indicates which temperatures are used to determine the average temperature value,
       which is the value used by the target temp, default value is -1**/
    double tempButtons;
1062 /** Impedance in Ohms, becomes more inaccurate above 400 ohms, which is the threshold
       for detecting tissue, default value is -1**/
    double impedance;
1064 /** Current Time in the windows data structure 'SYSTEMTIME', lowest values available
       in milliseconds, shows current time regardless of data status**/
    SYSTEMTIME currentTime;
1066 SerialData(){
      t1 = -1;
1068  t2 = -1;
      t3 = -1;
1070  t4 = -1;
      currentPower = -1;
1072  timer = -1;
      deliveredPower = -1;
1074  efficiency = -1;
      RFTime = -1;
1076  targetTemp = -1;
      modeIndicator = -1;
1078  tempButtons = -1;
      impedance = -1;
1080 }
    std::string toString();
1082 std::string getDataHeader();
    bool exportToTSV(std::string fileLocation);
1084 };

1086 //
     ////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
/// <summary> Converts two integers into a concatenated 14 bit byte as a double and
    divides them by 10 (this is the way data is stored in the RF generator)
///       e.g: given inputs 14 (1110) and 27 (11011), the returned value will be the
    concatenated decimal representation of these bits
///
///       so, 14 -> 0b(1110), 27 -> 0b(11011), concatenated(14,27) -> 0b(111011011) = 0d
    (475) and then it is divided by 10 to yield: convertBytesTo14(14,27) -> 47.5
///   </summary>
///
/// <remarks> Peter Grimm, 7/12/2018. </remarks>
///
/// <param name="byte1">  The first byte. </param>
/// <param name="byte2">  The second byte. </param>
///
/// <returns> The concatenated bytes converted to a 14 bit decimal represenation,
    divided by 10. </returns>
//
    ////////////////////////////////////////////////////////////////////////////////////////////////////


double convertBytesTo14(int byte1, int byte2);

//
    ////////////////////////////////////////////////////////////////////////////////////////////////////

/// <summary> Sends serial data initialize signal for the pneumatic pump microcontroller
    .
//
///       When the microcontroller recieves the signal (in this case it is a single 'I'
    character) it turns on the pump for 300 ms.
///       The purpose of this function is to ensure that the pump mechanism is
    initialized before use, due to the potential issue of air leaking from the valve
///   </summary>
///
/// <remarks> Peter Grimm, 7/12/2018. </remarks>
///
/// <param name="hPort">  Handle of the USB port that the microcontroller is connected
    to (use getUSBHandle to obtain this). </param>
///
/// <returns> True if it succeeds, false if it fails. </returns>
//
    ////////////////////////////////////////////////////////////////////////////////////////////////////


bool sendSerialDataInit(HANDLE hPort);

//
    ////////////////////////////////////////////////////////////////////////////////////////////////////

/// <summary> Sends serial data to the pneumatic pump microcontroller.
///       The current command for releasing the pressure from the valve and then
    repressurizing the system immediately after is a single character 's'.
///       uses handle obtained from 'getusbhandle' </summary>
```

```
1124 ///
     /// <remarks> Peter Grimm, 7/12/2018. </remarks>
1126 ///
     /// <param name="hPort">       Handle of the port. </param>
     /// <param name="bytes_to_send">  [in,out] If non-null, the bytes to send. </param>
1128 ///
     /// <returns> True if it succeeds, false if it fails. </returns>
1130 //
         ////////////////////////////////////////////////////////////////////////////////////////////////////


1132 bool sendSerialData(HANDLE hPort, char * bytes_to_send);

1134 //
         ////////////////////////////////////////////////////////////////////////////////////////////////////

     /// <summary> Gets the USB windows handle for interacting with the USB microcontroller.
1136 ///       This datatype can be used with the 'sendserialdata' commands, and they use the
          handle from this function as a parameter </summary>
     ///
1138 /// <remarks> Peter Grimm, 7/12/2018. </remarks>
     ///
1140 /// <param name="PortSpecifier">  Cstring of the port number (e.g COMx, where x is the
         COM port that the MCU is connect to) </param>
     ///
1142 /// <returns> The USB handle. </returns>
     //
         ////////////////////////////////////////////////////////////////////////////////////////////////////


1144
     HANDLE getUSBHandle(CString PortSpecifier);
1146
     //
         ////////////////////////////////////////////////////////////////////////////////////////////////////

1148 /// <summary> Returns a Serial data struct containing all of the current data in the
         serial stream from the RF generator.
     ///       Returns -1 for all values in the struct (null constructor settings) if the
         data packet has not been transferred properly.
1150 ///
     ///       This function works by creating a windows handle struct based on the input
         port specifier(referring to the COM port of the RF generator) as a Cstring
1152 ///        </summary>
     ///
1154 /// <remarks> Peter Grimm, 7/12/2018. </remarks>
     ///
1156 /// <param name="PortSpecifier">  Cstring of the port number (e.g COMx, where x is the
         COM port that the MCU is connect to) </param>
     ///
1158 /// <returns> The rf data packet. </returns>
     //
         ////////////////////////////////////////////////////////////////////////////////////////////////////


1160
```

```
struct SerialData GetRFDataPacket(CString PortSpecifier);

//
    ////////////////////////////////////////////////////////////////////////////////////////////////

/// <summary> Creates a serial windows handle, for use with the getRFTemp_nohandle
    function.
///   </summary>
///
/// <remarks> Peter Grimm, 7/12/2018. </remarks>
///
/// <param name="PortSpecifier">  Cstring of the port number (e.g COMx, where x is the
    COM port that the MCU is connect to) </param>
///
/// <returns> The serial handle. </returns>
//
    ////////////////////////////////////////////////////////////////////////////////////////////////


HANDLE getSerialHandle(CString PortSpecifier);

//
    ////////////////////////////////////////////////////////////////////////////////////////////////

/// <summary> Obtains current available serial data while using the handle created in
    getserialhandle, this method is slower than getRFPacket</summary>
///
/// <remarks> Peter Grimm, 7/12/2018. </remarks>
///
/// <param name="myHandle"> Handle of the serial RF generator. </param>
///
/// <returns> Null if it fails, else the rf temporary no handle. </returns>
//
    ////////////////////////////////////////////////////////////////////////////////////////////////


int * GetRFTemp_noHandle(HANDLE myHandle);

//
    ////////////////////////////////////////////////////////////////////////////////////////////////

/// <summary> Queries if a given file exists, returns false if it does not exist, true
    if it does. </summary>
///
/// <remarks> Peter Grimm, 7/12/2018. </remarks>
///
/// <param name="fileName"> Filename of the file. </param>
///
/// <returns> True if it succeeds, false if it fails. </returns>
//
    ////////////////////////////////////////////////////////////////////////////////////////////////


bool fileExists(const char *fileName);
```

```
1200 #endif
```

A.4 RF Interface Source

```
/**
 *  @file     RFSerialInterface.cpp
 *  @author   Peter Grimm
 *  @date     5/10/2018
 *  @version  1.0
 *
 *  @brief Functions for gathering data from AngioDynamics Rita 1500x serial line.
 *
 *  @section DESCRIPTION
 *
 *  These functions create a Win32 serial handle and process serial data from the RF generator.
 *  There are two sets of functions, one with a seperated handle building and read function,
 *  and one where they are combined. The combined call takes a slightly longer time to execute.
 *  The seperate functions present a significant delay however.
 *
 *  The RF generator needs to display temperature values on the device before the data is sent
 *  TODO: temperature values do not get transferred properly immediately upon turning off the current
 *
 */
#include "RFSerialInterface.h"

using namespace std;

//
    ////////////////////////////////////////////////////////////////////////////////////////////////////

/// @fn std::string SerialData::getDataHeader()
///
/// @brief  Prints the header for the output TSV file, should only be used at the beginning of a TSV file
///
/// @author Peter Grimm
/// @date 7/12/2018
///
/// @return A std::string that includes a tab seperated header followed by a newline
//
    ////////////////////////////////////////////////////////////////////////////////////////////////////


std::string SerialData::getDataHeader() {
  std::string tempString;
  tempString = "Day-"
    "Month-"
    "Year \t"
    "Hours \t"
    "Minutes \t"
    "Seconds \t"
```

```
1042      "Milliseconds \t"
          "t1 \t"
1044      "t2 \t"
          "t3 \t"
1046      "t4 \t"
          "currentPower\t"
1048      "timer \t"
          "deliveredPower \t"
1050      "efficiency \t"
          "RFTime \t"
1052      "targetTemp \t"
          "modeIndicator \t"
1054      "tempButtons \t"
          "impedance \t";
1056    return (tempString);
  }
1058
  //
      ////////////////////////////////////////////////////////////////////////////////////////////////////

1060  /// @fn std::string SerialData::toString()
  ///
1062  /// @brief  Convert the serialData object into a string representation, for use in the
      TSV file
  ///
1064  /// @author Peter Grimm
  /// @date 7/12/2018
1066  ///
  /// @return A std::string that includes a tab seperated string for the TSV file
1068  //
      ////////////////////////////////////////////////////////////////////////////////////////////////////


1070  std::string SerialData::toString() {
    string tempString;
1072    tempString = to_string(static_cast<long double>((this->currentTime).wDay)) + "-"
      + to_string(static_cast<long double>((this->currentTime).wMonth)) + "-"
1074      + to_string(static_cast<long double>((this->currentTime).wYear)) + "\t"
      + to_string(static_cast<long double>((this->currentTime).wHour)) + "\t"
1076      + to_string(static_cast<long double>((this->currentTime).wMinute)) + "\t"
      + to_string(static_cast<long double>((this->currentTime).wSecond)) + "\t"
1078      + to_string(static_cast<long double>((this->currentTime).wMilliseconds)) + "\t"
      + to_string(static_cast<long double>(this->t1)) + "\t"
1080      + to_string(static_cast<long double>(this->t2)) + "\t"
      + to_string(static_cast<long double>(this->t3)) + "\t"
1082      + to_string(static_cast<long double>(this->t4)) + "\t"
      + to_string(static_cast<long double>(this->currentPower)) + "\t"
1084      + to_string(static_cast<long double>(this->timer)) + "\t"
      + to_string(static_cast<long double>(this->deliveredPower)) + "\t"
1086      + to_string(static_cast<long double>(this->efficiency)) + "\t"
      + to_string(static_cast<long double>(this->RFTime)) + "\t"
1088      + to_string(static_cast<long double>(this->targetTemp)) + "\t"
      + to_string(static_cast<long double>(this->modeIndicator)) + "\t"
1090      + to_string(static_cast<long double>(this->tempButtons)) + "\t"
```

```
    + to_string(static_cast<long double>(this->impedance)) + "\t";
  return (tempString);
}

//
    ////////////////////////////////////////////////////////////////////////////////////////////////

/// @fn bool fileExists(const char *fileName)
///
/// @brief  Queries if a given file exists
///
/// @author Peter Grimm
/// @date 7/12/2018
///
/// @param  fileName  Filename of the file.
///
/// @return True if it succeeds, false if it fails.
//
    ////////////////////////////////////////////////////////////////////////////////////////////////


bool fileExists(const char *fileName)
{
  ifstream infile(fileName);
  return infile.good();
}

//
    ////////////////////////////////////////////////////////////////////////////////////////////////

/// @fn bool SerialData::exportToTSV(std::string path)
///
/// @brief  Export to tsv
///
/// @author Peter Grimm
/// @date 7/12/2018
///
/// @param  path  Full pathname of the file.
///
/// @return True if it succeeds, false if it fails.
//
    ////////////////////////////////////////////////////////////////////////////////////////////////


bool SerialData::exportToTSV(std::string path) {
  const char *mycharp = path.c_str();
  bool headerNeeded = false;
  if (!fileExists(mycharp)) {  //Checks if it is the first data packet in the TSV
    cout << "file does not exist, creating new file...";
    headerNeeded = true;
  }
  ofstream thisFile(path, std::ios_base::app);
  if (thisFile.is_open()) {
    if (headerNeeded)
```

```
      thisFile << this->getDataHeader() << "\n"; //output header to TSV
1138
    thisFile << this->toString() << "\n"; //output data to TSV
1140    thisFile.close();
    return true;
1142  }
  else return false;
1144 }

1146 //
    ////////////////////////////////////////////////////////////////////////////////////////////////

/// @fn double convertBytesTo14(int byte1, int byte2)
1148 ///
/// @brief  Convert two bytes into 14 bit decimal representation.
1150 ///
/// @author Peter Grimm
1152 /// @date 7/12/2018
///
1154 /// @param  byte1 The first byte.
/// @param  byte2 The second byte.
1156 ///
/// @return The bytes converted to 14.
1158 //
    ////////////////////////////////////////////////////////////////////////////////////////////////


1160 double convertBytesTo14(int byte1, int byte2) {
  //Function convertBytesTo14
1162  //arguments: (7bit)int byte1 and (7 bit) int byte2, to change into a 14 bit combined
    int
  //return value: double that is the combined 14 bit appended int of the two 'byte'
    inputs divided by 10
1164  //description: given two 7-bit ints, appends their binary equivalent together and
    converts that to an int, it then divides it by 10

1166  int total;
  std::string binary1 = std::bitset<7>(byte1).to_string();
1168  std::string binary2 = std::bitset<7>(byte2).to_string();
  std::string totBinary = binary1 + binary2;
1170  unsigned long decimal = std::bitset<14>(totBinary).to_ulong();
  return ((double)decimal) / 10;
1172 }

1174 //
    ////////////////////////////////////////////////////////////////////////////////////////////////

/// @fn HANDLE getUSBHandle (CString PortSpecifier)
1176 ///
/// @brief  Gets USB handle for use with automated pneumatic trigger
1178 ///
/// @author Peter Grimm
1180 /// @date 7/12/2018
///
```

```
1182 /// @param  PortSpecifier Information describing the port as a string, generally called
         COM#
     ///
1184 /// @return The USB handle.
     //
         ////////////////////////////////////////////////////////////////////////////////////////////////////

1186
     HANDLE getUSBHandle(CString PortSpecifier)
1188 {
       //serial varaiables
1190   DCB dcb;
       DWORD dwBytesTransferred;
1192   DWORD dwCommModemStatus;
       int serialBuffer[256];//Buffer for storing Rxed Data
1194   DWORD NoBytesRead;
       //declare serial handle
1196   HANDLE hPort = CreateFile(
         PortSpecifier,
1198     GENERIC_READ | GENERIC_WRITE,
         0,
1200     NULL,
         OPEN_EXISTING,
1202     0,
         NULL
1204   );
       //return no data if com port does not connect
1206   if (!GetCommState(hPort, &dcb)) {
         CloseHandle(hPort);
1208     return nullptr;
       }
1210   COMMTIMEOUTS comTimeOut;
       comTimeOut.ReadIntervalTimeout = 50;
1212
       dcb.BaudRate = CBR_115200; //115200 Baud
1214   dcb.ByteSize = 8; //8 data bits
       dcb.Parity = NOPARITY; //no parity
1216   dcb.StopBits = ONESTOPBIT; //1 stop
       if (!SetCommState(hPort, &dcb)) {
1218     CloseHandle(hPort);
         return 0;
1220   }

1222   SetCommMask(hPort, EV_RXCHAR | EV_ERR); //receive character event
       return hPort;
1224 }

1226 //
         ////////////////////////////////////////////////////////////////////////////////////////////////////

     /// @fn bool fdata(HANDLE hPort, char* bytes_to_send)
1228 ///
     /// @brief  Sends serial data to the arduino MCU in accordance to the protocols it's
         able to
```

```
1230 ///     handle, currently the correct way to initiate a pump trigger is with an array
         that starts with 's'
     ///
1232 /// @author Peter Grimm
     /// @date 7/12/2018
1234 ///
     /// @param        hPort     Handle of the port.
1236 /// @param [in,out] bytes_to_send If non-null,a pointer to the array of the bytes to
         send.
     ///
1238 /// @return A bool, true if successfull, false if not.
     //
         ////////////////////////////////////////////////////////////////////////////////////////////////////


1240
     bool sendSerialData(HANDLE hPort, char* bytes_to_send) {
1242   //variable decleration
       DWORD bytes_written, total_bytes_written = 0;
1244   fprintf(stderr, "Sending bytes...");
       WriteFile(hPort, bytes_to_send, 1, &bytes_written, NULL);
1246   //if (!WriteFile(hPort, bytes_to_send, 2, &bytes_written, NULL))
       //{
1248     //fprintf(stderr, "Error\n");

1250   return 1;
       //}
1252   //}
       //else if (dwCommModemStatus & EV_ERR) {
1254   //CloseHandle(hPort);
       //return 0;
1256   //}

1258   //return 0;
     }
1260
     //
         ////////////////////////////////////////////////////////////////////////////////////////////////////

1262 /// @fn int sendSerialDataInit(HANDLE hPort)
     ///
1264 /// @brief  Init MCU circuit, should be completed before use of the pneumatic circuit
     ///
1266 /// @author Peter Grimm
     /// @date 7/12/2018
1268 ///
     /// @param  hPort Handle of the port.
1270 ///
     /// @return A bool, true if successfull, false if not.
1272 //
         ////////////////////////////////////////////////////////////////////////////////////////////////////


1274 bool sendSerialDataInit(HANDLE hPort) {
       DWORD bytes_written, total_bytes_written = 0;
```

```
1276    char bytes_to_send[1];
        bytes_to_send[0] = 'I';
1278
        fprintf(stderr, "Sending bytes...");
1280    WriteFile(hPort, bytes_to_send, 1, &bytes_written, NULL);
        //if (!WriteFile(hPort, bytes_to_send, 2, &bytes_written, NULL))
1282    //{
        //fprintf(stderr, "Error\n");
1284
        return 1;
1286    //}
        //}
1288    //else if (dwCommModemStatus & EV_ERR) {
        //CloseHandle(hPort);
1290    //return 0;
        //}
1292
        //return 0;
1294  }

1296  //
          ////////////////////////////////////////////////////////////////////////////////////////////////

      /// @fn struct SerialData GetRFDataPacket(CString PortSpecifier)
1298  ///
      /// @brief  Gets rf data packet
1300  ///
      /// @author Peter Grimm
1302  /// @date 7/12/2018
      ///
1304  /// @param  PortSpecifier Information describing the port.
      ///
1306  /// @return The rf data packet in the form of a struct, returns default struct if no
          data is found.
      //
          ////////////////////////////////////////////////////////////////////////////////////////////////


1308
      struct SerialData GetRFDataPacket(CString PortSpecifier)
1310  {
        //logic variables
1312    struct SerialData tempSerialData;
        int prevChar = 0;
1314    int prev3Char = 0;
        char TempChar = 'a'; //Temporary character used for reading
1316    int i = 0;
        bool secondFlag = false;
1318    bool resetFlag = false;
        SYSTEMTIME timeVar;
1320    //serial varaiables
        DCB dcb;
1322    DWORD dwBytesTransferred;
        DWORD dwCommModemStatus;
1324    int serialBuffer[100];//Buffer for storing Rxed Data
```

```
    DWORD NoBytesRead;
1326 //declare serial handle
    HANDLE hPort = CreateFile(
1328   PortSpecifier,
    GENERIC_READ,
1330   0,
    NULL,
1332   OPEN_EXISTING,
    0,
1334   NULL
    );
1336
    if (hPort == INVALID_HANDLE_VALUE) {
1338   CloseHandle(hPort);
    return tempSerialData;
1340
    }
1342
    //return no data if com port does not connect
1344 if (!GetCommState(hPort, &dcb)) {
    CloseHandle(hPort);
1346   return tempSerialData;
    }
1348 if (!EscapeCommFunction(hPort, CLRDTR)) {

1350 }

1352 Sleep(200);

1354 if (!EscapeCommFunction(hPort, SETDTR)) {
    CloseHandle(hPort);
1356   return tempSerialData;
    }
1358 COMMTIMEOUTS comTimeOut;

1360 comTimeOut.ReadIntervalTimeout = 25;
    comTimeOut.ReadTotalTimeoutConstant = 40;
1362 comTimeOut.ReadTotalTimeoutMultiplier = 1;
    comTimeOut.WriteTotalTimeoutMultiplier = 1;
1364 comTimeOut.WriteTotalTimeoutConstant = 1;
    if (!SetCommTimeouts(hPort, &comTimeOut)) {
1366   CloseHandle(hPort);
    return tempSerialData;
1368 }

1370 dcb.BaudRate = CBR_9600; //9600 Baud
    dcb.ByteSize = 7; //7 data bits
1372 dcb.Parity = NOPARITY; //no parity
    dcb.StopBits = ONESTOPBIT; //1 stop
1374 if (!SetCommState(hPort, &dcb)) {
    CloseHandle(hPort);
1376   return tempSerialData;
    }
1378
```

```
1380
    SetCommMask(hPort, EV_RXCHAR | EV_ERR); //receive character event
1382

1384  if (/*dwCommModemStatus &*/ EV_RXCHAR) {
      //grab one data frame, delimitted by characters "3 80 24"
1386    do
        {
1388
          prev3Char = prevChar;
1390      prevChar = (int)TempChar;

1392      ReadFile(hPort,           //Handle of the Serial port
            &TempChar,        //Temporary character
1394        sizeof(TempChar),//Size of TempChar
            &NoBytesRead,     //Number of bytes read
1396        NULL);

1398      if (secondFlag && resetFlag) {
            i = 0;
1400        resetFlag = false;

1402      }
          if (secondFlag) {
1404        serialBuffer[i] = (int)TempChar;
            i++;
1406      }
          if ((int)TempChar == 3 && prevChar == 127 /*&& prev3Char == 3*/) {
1408        if (i >= 59 && secondFlag /*&& serialBuffer[43] == 48 /*&& serialBuffer[6] == 0
    */) {
            tempSerialData.t1 = convertBytesTo14(serialBuffer[19], serialBuffer[20]); //
1410        tempSerialData.t2 = convertBytesTo14(serialBuffer[21], serialBuffer[22]); //
            tempSerialData.t3 = convertBytesTo14(serialBuffer[23], serialBuffer[24]); //
1412        tempSerialData.t4 = convertBytesTo14(serialBuffer[25], serialBuffer[26]); //
            tempSerialData.currentPower = convertBytesTo14(serialBuffer[9], serialBuffer
    [10]);
1414        tempSerialData.timer = convertBytesTo14(serialBuffer[11], serialBuffer[12]) /
    60;
            tempSerialData.deliveredPower = convertBytesTo14(serialBuffer[13],
    serialBuffer[14]); //
1416        tempSerialData.efficiency = std::bitset<7>(serialBuffer[2]).to_ulong();//note:
     only 1 byte
            tempSerialData.RFTime = convertBytesTo14(serialBuffer[15], serialBuffer[16]) /
     60;
1418        tempSerialData.targetTemp = convertBytesTo14(serialBuffer[37], serialBuffer
    [38]);
            /**/tempSerialData.modeIndicator = convertBytesTo14(serialBuffer[35],
    serialBuffer[36]);
1420        tempSerialData.tempButtons = std::bitset<7>(serialBuffer[39]).to_ulong();
            tempSerialData.impedance = convertBytesTo14(serialBuffer[17], serialBuffer
    [18]);
1422        GetSystemTime(&tempSerialData.currentTime);
            //27-35 are aux temps
```

```
1424            //55 rate?
                //
1426            //1 and 2 significant?
                //3 and 4/41 and 42 something to do with power
1428            //17 and 18, something to do with efficiency
                //49-57 seem insignificant, besides 55
1430            CloseHandle(hPort);
                return tempSerialData;
1432          }
              secondFlag = true;
1434          resetFlag = true;
            }
1436        if (i >= 60 && secondFlag) {
              secondFlag = false;
1438          i = 0;
            }
1440      } while (i < 100);
    }
1442    else /*if (dwCommModemStatus & EV_ERR)*/ {
      CloseHandle(hPort);
1444    return tempSerialData;
    }
1446    CloseHandle(hPort);
    return tempSerialData;
1448 }

1450 //
      ////////////////////////////////////////////////////////////////////////////////////////////////

    /// @fn HANDLE getSerialHandle(CString PortSpecifier)
1452 ///
    /// @brief  **NOTE**:this function can delay data reads. Gets serial handle for RF
        serial stream.
1454 ///
    /// @author Peter Grimm
1456 /// @date 7/12/2018
    ///
1458 /// @param  PortSpecifier Information describing the port.
    ///
1460 /// @return The serial handle.
    //
      ////////////////////////////////////////////////////////////////////////////////////////////////

1462
    HANDLE getSerialHandle(CString PortSpecifier)
1464 {
    //Function GetRFTemp
1466  //arguments: Com interface string
    //return value: Temperatures as a pointer to a 1x4 array
1468  //if no data is processed, returns 1x4 array of -1

1470  //serial varaiables
    DCB dcb;
1472  DWORD dwBytesTransferred;
```

```
  DWORD dwCommModemStatus;
1474  DWORD NoBytesRead;
  //declare serial handle
1476  HANDLE hPort = CreateFile(
    PortSpecifier,
1478    GENERIC_READ,
    0,
1480    NULL,
    OPEN_EXISTING,
1482    0,
    NULL
1484  );
  //return no data if com port does not connect
1486  if (!GetCommState(hPort, &dcb)) {
    return nullptr;
1488  }
  COMMTIMEOUTS comTimeOut;
1490  comTimeOut.ReadIntervalTimeout = 50;

1492  dcb.BaudRate = CBR_9600; //9600 Baud
  dcb.ByteSize = 7; //7 data bits
1494  dcb.Parity = NOPARITY; //no parity
  dcb.StopBits = ONESTOPBIT; //1 stop
1496  if (!SetCommState(hPort, &dcb)) {
    return nullptr;
1498  }

1500  return hPort;
}
1502

//
    ////////////////////////////////////////////////////////////////////////////////////////////////////

1504 /// @fn int * GetRFTemp_noHandle(HANDLE myHandle)
///
1506 /// @brief  **NOTE** not working properly, use GetRFDataPacket Gets rf temporary no
    handle
///
1508 /// @author Peter Grimm
/// @date 7/12/2018
1510 ///
/// @param  myHandle  Handle of my.
1512 ///
/// @return Null if it fails, else the rf temporary no handle.
1514 //
    ////////////////////////////////////////////////////////////////////////////////////////////////////


1516 int * GetRFTemp_noHandle(HANDLE myHandle)
{
1518  //Function GetRFTemp
  //arguments: Com interface string
1520  //return value: Temperatures as a pointer to a 1x4 array
  //if no data is processed, returns 1x4 array of -1
```

```
1522  //logic variables
      int count = 0;
1524  int tempArray[4] = { -1,-1,-1,-1 };
      int prevChar = 0;
1526  int prev3Char = 0;
      char TempChar = 'a'; //Temporary character used for reading
1528  int i = 0;
      bool secondFlag = false;
1530  bool resetFlag = false;
      //serial varaiables
1532  DCB dcb;
      DWORD dwBytesTransferred;
1534  DWORD dwCommModemStatus;
      int serialBuffer[256];//Buffer for storing Rxed Data
1536  DWORD NoBytesRead;
      //declare serial handle
1538  HANDLE hPort = myHandle;
      //return no data if com port does not connect
1540  if (!GetCommState(hPort, &dcb)) {
        return tempArray;
1542  }
      COMMTIMEOUTS comTimeOut;
1544  comTimeOut.ReadIntervalTimeout = 50;

1546  dcb.BaudRate = CBR_9600; //9600 Baud
      dcb.ByteSize = 7; //7 data bits
1548  dcb.Parity = NOPARITY; //no parity
      dcb.StopBits = ONESTOPBIT; //1 stop
1550  if (!SetCommState(hPort, &dcb)) {
        return tempArray;
1552  }

1554  SetCommMask(hPort, EV_RXCHAR | EV_ERR); //receive character event
      WaitCommEvent(hPort, &dwCommModemStatus, 0); //wait for character
1556
      if (dwCommModemStatus & EV_RXCHAR) {
1558    //grab one data frame, delimitted by characters "3 80 24"
        do
1560    {
          prev3Char = prevChar;
1562      prevChar = (int)TempChar;

1564      ReadFile(hPort,           //Handle of the Serial port
            &TempChar,        //Temporary character
1566        sizeof(TempChar),//Size of TempChar
            &NoBytesRead,     //Number of bytes read
1568        NULL);
          //WaitCommEvent (hPort, &dwCommModemStatus, 0);
1570      if (secondFlag && resetFlag) {

1572        i = 0;
            count++;
1574        resetFlag = false;
          }
```

```
1576        serialBuffer[i] = (int)TempChar;
            if ((int)TempChar == 24 && prevChar == 80 && prev3Char == 3) {
1578          if (secondFlag && serialBuffer[43] == 49 /*&& serialBuffer[44] == 49*/) {
                tempArray[3] = convertBytesTo14(serialBuffer[17], serialBuffer[18]);
1580            tempArray[2] = convertBytesTo14(serialBuffer[19], serialBuffer[20]);
                tempArray[1] = convertBytesTo14(serialBuffer[23], serialBuffer[24]);
1582            tempArray[0] = convertBytesTo14(serialBuffer[21], serialBuffer[22]);

1584            return tempArray;

1586          }
              secondFlag = true;
1588          resetFlag = true;
            }
1590        i++;

1592      } while (i < 256);
        }
1594    else if (dwCommModemStatus & EV_ERR) {
          return tempArray;
1596    }
        return tempArray;
1598  }
```

APPENDIX B MATLAB CODE

B.1 ULTRASOUND DATA CLASS

```
%% USDataClass: Data class for computation of echo decorrelation
classdef USDataClass < handle
    % USDataClass: Data class for computation of echo decorrelation
    %   each set of data has is an object of type USDataClass
    %   contains the data parameters, raw spherical data, cartesian scan converted data, ibs,
    %   and decorrelation
    % Author: Peter Grimm 12/21/2019
    properties
        % data properties
        rawData;            % raw spherical volume data for one data set export
        rawData_cart;       % scan converted cartesian data from spherical data
        ROICart;
        ibs;                % Integrated backscatter for every (cartesian) volume
        decorr;             % decorr (cartesian) between volume at t and t+tau
        autocorr01;         % autocorr between volume at t and t+tau
        decorrAvg;          % average of decorrelation in entire data set, either ensemble or running
        time;
        cumulativeDecorr;
        cumulativeDecorrSum;
        decorrThresh;
        decorrMap;
        decorrSumPixels;
        decorrVolEstimate;
        % bounds
        ROIBounds;          % bounds of region of interest [xMin xMax yMin yMax zMin zMax]
        ROIBounds_spherical; %TODO, minimum bounds in spherical coordinates

        % bounded data properties
        rawData_cart_ROI;
        ibs_ROI;
        decorr_ROI;
        autocorr01_ROI;
        decorrAvg_ROI;
        % parameters
        %The following would ideally be part of InfoFile in the future
        InfoFile;           % Info file provided by siemens SC2000 scanner
        rmax,rmin,thetamax,thetamin,phimax,phimin; % bounds in cm of the spherical data
        xMin,xMax,yMin,yMax,zMin,zMax; % bounds in cm of cartesian data
        x_range,y_range,z_range; % range in cartesian plane
        windowSigma;        % sigma of gaussian smoothing kernel
        dPhi;               % angular difference between successive phi
        dTheta;             % angular difference between successive theta
        dr;                 % distance in cm in the radius direction(cm/pixel)
        dx;                 % distance in cm in the x direction (cm/pixel)
        dy;                 % distance in cm in the x direction (cm/pixel)
        dz;                  % distance in cm in the x direction (cm/pixel)
```

```
1046        interFrameTime;     % time between volume recordings (cm/pixel)
            cartScalingFactor; % Factor to scale cartesian distances by (dx/dr)
1048                           % e.g Given dr, to find dx take
                               % dr*cartScalingFactor = dx
1050                           % reduces resolution by a factor of
                               % cartScalingFactor for faster scan conversion
1052        maskfilt;          % Gaussian Mask, for FFT based filtering
            rawData_cart_slicemethod;
1054        ibs_slicemethod;               % Integrated backscatter for every (cartesian)
    volume
            decorr_slicemethod;            % decorr (cartesian) between volume at t and t+
    tau
1056        autocorr01_slicemethod;        % autocorr between volume at t and t+tau
            decorrAvg_slicemethod;         % average of decorrelation in entire data set,
    either ensemble or running
1058        % scan conversion properties
            dmu;
1060        dnu;
            frustumPoints;
1062        imu;
            Lmu;
1064        inu;
            Lnu;
1066        iR;
            LR;
1068        R0;
            mu0;
1070        nu0;
    end
1072
    methods
1074        %% Constructor method
            function obj = USDataClass(thisRawData,startTime,thisInfoFile,thisrmin,thisrmax,
    thisthetamin,thisthetamax,thisphimin,thisphimax,thiscartScalingFactor,
    thiswindowSigma,thisinterFrameTime,decorrThresh)
1076            obj.rawData = thisRawData;
                obj.InfoFile = thisInfoFile;
1078            obj.rmax = thisrmax;
                obj.rmin = thisrmin;
1080            obj.thetamax = thisthetamax;
                obj.thetamin = thisthetamin;
1082            obj.phimax = thisphimax;
                obj.phimin = thisphimin;
1084            obj.cartScalingFactor = thiscartScalingFactor;
                obj.windowSigma = thiswindowSigma;
1086            obj.interFrameTime = thisinterFrameTime;
                obj.time = startTime;
1088        end
            %% Scan Conversion Method
1090        %% Display functions
            function displayRawData_cart(obj)
1092            imagesc(squeeze(abs(obj.rawData_cart(:,:,30,1))))
            end
1094        function displayDecorr(obj)
```

```
            imagesc(squeeze(abs(obj.decorr(:,:,20,1))))
        end
        %% compute ensemble average
        function computeDecorrelationAverage(obj,method)
            if strcmp(method,'ensemble') == 1
                obj.decorrAvg = sum(obj.decorr,4)/size(obj.decorr,4);

            elseif strcmp(metod,'running') == 1

            else
                error('please enter proper method name (ensemble or running)');
            end
        end
        %% compute ensemble average in ROI
        function computeDecorrelationAverage_ROI(obj,method)
            if strcmp(method,'ensemble') == 1
                obj.decorrAvg = sum(subvolume(obj.decorr_slicemethod,obj.ROIBounds),4)/
    size(obj.decorr_slicemethod,4);

            elseif strcmp(method,'running') == 1

            else
                error('please enter proper method name (ensemble or running)');
            end
        end
        %% calculate minimum spherical coordinate bounds that encapsulates the cartesian
     ROI bounds
        % potentially useful for speeding up computation given small enough
        % sigma to make windowing effect irrelevant
        function computeMinROIBoundsSpherical(obj)
            [minX maxX minY maxY minZ maxZ] = obj.ROIBounds;
            minTheta = arctan(minY/minX);
            maxTheta = arctan(minY/maxX);
            minPhi = arctan(sqrt(minX^2 + minY^2)/minZ);
            %todo
            maxPhi = minPhi;
            %minR not correct TODO
            minR = sqrt(minX^2 + minY^2 + minZ^2);
            maxR = sqrt(maxX^2 + maxY^2 + maxZ^2);
            obj.ROIBounds_spherical = [minR, maxR,minTheta,maxTheta,minPhi,maxPhi];
        end
        function setROI(obj,thisROIBounds)
            if(isempty(obj.decorr))
               error('execute computedecorr3d before getting decorrelation within bounds
    ')
            end

            obj.ROIBounds = thisROIBounds;

            % convert cm to pixels
            minX = floor(obj.ROIBounds(1)/obj.dx)+1;
            maxX = floor(obj.ROIBounds(2)/obj.dx)+1;
            minY = floor(obj.ROIBounds(3)/obj.dy)+1;
            maxY = floor(obj.ROIBounds(4)/obj.dy)+1;
```

```
1146            minZ = floor(obj.ROIBounds(5)/obj.dz)+1;
                maxZ = floor(obj.ROIBounds(6)/obj.dz)+1;
1148            obj.rawData_cart_ROI = obj.rawData_cart(minX:maxX,minY:maxY,minZ:maxZ,:);
                obj.ibs_ROI = obj.ibs(minX:maxX,minY:maxY,minZ:maxZ,:);
1150            obj.decorr_ROI = obj.decorr(minX:maxX,minY:maxY,minZ:maxZ,:);
                obj.autocorr01_ROI = obj.autocorr01(minX:maxX,minY:maxY,minZ:maxZ,:);
1152            obj.decorrAvg_ROI = obj.decorrAvg(minX:maxX,minY:maxY,minZ:maxZ,:);
        end
1154    function accumulateDecorr(obj,prevItem)
            obj.cumulativeDecorr = max(prevItem.cumulativeDecorr,obj.decorr);
1156        obj.cumulativeDecorrSum = sum(obj.cumulativeDecorr(:));
            obj.decorrMap = obj.cumulativeDecorr;
1158        obj.decorrMap(find(obj.decorrMap >= obj.decorrThresh)) = 1;
            obj.decorrMap(find(obj.decorrMap < obj.decorrThresh)) = 0;
1160        obj.decorrSumPixels = sum(obj.decorrMap(:));
        end
1162    function initAccumulateDecorr(obj)
            obj.cumulativeDecorr = obj.decorr;
1164        obj.cumulativeDecorrSum = sum(obj.cumulativeDecorr(:));
            obj.decorrMap = obj.cumulativeDecorr;
1166        obj.decorrMap(find(obj.decorrMap >= obj.decorrThresh)) = 1;
            obj.decorrMap(find(obj.decorrMap < obj.decorrThresh)) = 0;
1168        obj.decorrSumPixels = sum(obj.decorrMap(:));
        end
1170  end

1172

end
```

B.2 Decorrelation Computation

```
function compute3DDecorr( obj )
%COMPUTE3DDECORR Summary of this function goes here
%   Detailed explanation goes here

 % COMPUTE3DDECORR Summary of this function goes here
%   Detailed explanation goes here

    %
    %%
    % *Define Guassian Window*
    x_range_length = size(obj.x_range,2);
    y_range_length = size(obj.y_range,2);
    z_range_length = size(obj.z_range,2);
    sigx = obj.windowSigma/obj.dx;
    sigy = obj.windowSigma/obj.dy;
    sigz = obj.windowSigma/obj.dz;
    x_mid = ceil(x_range_length/2);
    y_mid = ceil(y_range_length/2);
    z_mid = ceil(z_range_length/2);
    sigfaz = x_range_length/(2*pi*sigx);
    sigfra = x_range_length/(2*pi*sigy);
    sigfel = x_range_length/(2*pi*sigz);
    %coeffX =    1/sqrt(2*pi*sigx^2);
    %coeffY =    1/sqrt(2*pi*sigy^2);
    %coeffZ =    1/sqrt(2*pi*sigz^2);
%     xmask = coeffX*exp(-(((1:vol_x_length)-x_mid).^2)/(2*sigx^2));
%     ymask = coeffY*exp(-(((1:vol_y_length)-y_mid).^2)/(2*sigy^2));
%     zmask = coeffZ*exp(-(((1:vol_z_length)-z_mid).^2)/(2*sigz^2));
%     azMask   = filtFactAz .* exp(-((1:nSigPad)-azId).^2/2/sigFAz^2);
%     raMask   = filtFactRa .* exp(-((1:nRowPad)-rangeId).^2/2/sigFRa^2);
%     [am,rm]  = meshgrid(azMask,raMask);
%     maskFilt = fftshift(am.*rm);
    xmask = exp(-(((1:x_range_length)-x_mid).^2)/2/sigfaz^2);
    ymask = exp(-(((1:y_range_length)-y_mid).^2)/2/sigfra^2);
    zmask = exp(-(((1:z_range_length)-z_mid).^2)/2/sigfel^2);

    [x_mask_mat,y_mask_mat,z_mask_mat] = ndgrid(zmask,xmask,ymask);
    %maskfilt = fftshift(x_mask_mat.*y_mask_mat.*z_mask_mat);
    %maskfilt = maskfilt/sum(maskfilt(:));
    maskfilt = (fftshift(x_mask_mat.*y_mask_mat.*z_mask_mat));
    maskfilt = maskfilt/sum(maskfilt(:));

    %imagesc(maskfilt(:,:,1))
    % *compute windowed ibs and autocorr01*
    %compute ibs and autocorr before windowing
    obj.ibs = abs(obj.rawData_cart).^2;
    obj.autocorr01 = obj.rawData_cart(:,:,:,1:(end-1)).*conj(obj.rawData_cart(:,:,:,2:
    end));
    % set NaN values to small number
    obj.autocorr01(find(isnan(obj.autocorr01))) = realmin('double');
    obj.ibs(find(isnan(obj.ibs))) = realmin('double');
    %compute windowed ibs
```

```
    for currVolume = 1:size(obj.ibs,4)
1052      obj.ibs(:,:,:,currVolume) = abs(ifftn(fftn(obj.ibs(:,:,:,currVolume)).*maskfilt));
        %obj.ibs(:,:,:,currVolume) = imgaussfilt3(abs(obj.ibs(:,:,:,currVolume)),sigx);
1054    end
    %compute autcorrelation and decorrelation
1056    for currVolume = 1:(size(obj.ibs,4)-1)
        obj.autocorr01(:,:,:,currVolume) = abs(ifftn(fftn(obj.autocorr01(:,:,:,
    currVolume)).*maskfilt));
1058      %obj.autocorr01(:,:,:,currVolume) = imgaussfilt3((obj.autocorr01(:,:,:,currVolume
    )),sigx);
    end
1060    for currVolume = 1:(size(obj.ibs,4)-1)
        %obj.decorr(:,:,:,currVolume) = (1 - abs(obj.autocorr01(:,:,:,currVolume)).^2./(
    obj.ibs(:,:,:,currVolume).*obj.ibs(:,:,:,currVolume+1))./obj.interFrameTime;
1062        R00 = obj.ibs(:,:,:,currVolume);
        R11 = obj.ibs(:,:,:,currVolume+1);
1064        B2 = R00.*R11;
        R01 = abs(obj.autocorr01(:,:,:,currVolume)).^2;
1066        %thisMean = mean(abs(obj.autocorr01(:)));
        tau = obj.interFrameTime;
1068        obj.decorr(:,:,:,currVolume) = 2*10^-3*(B2-R01)./(B2 + mean(B2(:)))/tau;
        %obj.decorr(:,:,:,currVolume) = (1 - abs(obj.autocorr01(:,:,:,currVolume)).^2./(
    thisMean+obj.ibs(:,:,:,currVolume).*obj.ibs(:,:,:,currVolume+1))./;
1070        %obj.decorr(:,:,:,currVolume) = 2*((B2 - (obj.autocorr01)).^2./(obj.
    interFrameTime*(mean(B2(:)) + B2)));
    end
1072    % set values outside of volume to small number
    obj.autocorr01(find(isnan(obj.rawData_cart(:,:,:,1:(end-1))))) = realmin('double');
1074    obj.ibs(find(isnan(obj.rawData_cart))) = realmin('double');
    obj.decorr(find(isnan(obj.rawData_cart(:,:,:,1:(end-1))))) = realmin('double');
1076 end
```

B.3 SCAN CONVERSION

```
function scanConv_Frust( obj )
% This script reads IQ data in the first section (SIEMENS Scripts), then
% does the 3D interpolation on spherical IQ data with function named
% "frustumInterp". In the last section the spherical and
% interpolated data in cartesian system are displayed.

% Writers: Elmira Ghahramani Z., Dr. Douglas Mast
% Image-guided Ultrasound Therapeutics Laboratories
% University of Cincinnati
% Contacts: ghahraea@mail.uc.edu
%           masttd@UCMAIL.UC.EDU
% Date last updated: 02/19/2019

%% Read IQ data

for volIndex = 1:size(obj.rawData,4)
    Isph = squeeze(obj.rawData(:,:,:,volIndex));

    %% set up the parameters
    %tic
    %global p LR Lmu Lnu iR imu inu R0
    sizeR = size(Isph,1);
    sizeAz = size(Isph,2);
    sizeEl = size(Isph,3);

    % define coordinates on pyramidal grid
    obj.dr = 1/obj.InfoFile.NumSamplesPerMm;          % range (mm)
    Rvec = obj.rmin:obj.dr:obj.rmax;

    % sin theta (azimuth)
    mumax = sin(obj.thetamax); mumin=-mumax;
    muvec = linspace(mumin,mumax,sizeAz);
    dmu = muvec(2)-muvec(1);

    % sin phi (elevation)
    numax = sin(obj.phimax); numin=-numax;
    nuvec = linspace(numin,numax,sizeEl);
    dnu = nuvec(2)-nuvec(1);

    [R,mu,nu] = ndgrid(Rvec,muvec,nuvec);

    % Cartesian grid, onto which we're scan-converting (interpolating)
    %obj.cartScalingFactor =2;
    obj.dz = obj.cartScalingFactor*obj.dr;%0.5;  % spatial step
    obj.dx = obj.dz;
    obj.dy = obj.dz;
    maxY = obj.rmax*sin(obj.thetamax); maxZ = obj.rmax*sin(obj.phimax);
    obj.z_range = 0:obj.dz:obj.rmax;  % depth
    obj.y_range = -maxY:obj.dz:maxY; %-32:obj.dz:32;  % azimuth
    obj.x_range = -maxZ:obj.dz:maxZ; %-31:obj.dz:31;  % elevation;
    [z,y,x] = ndgrid(obj.z_range,obj.y_range,obj.x_range);
```

```
    %defining image cross sections (default: half range, az., el.)
%     ixmid = find(abs(obj.x_range)==min(abs(obj.x_range)));
%     iymid = find(abs(obj.y_range)==min(abs(obj.y_range)));
%     izmid = ceil(size(obj.z_range,2)/2);

    % pyramidal coordinates of Cartesian grid points
    R0 = sqrt(x.^2+y.^2+z.^2);
    mu0 = y./sqrt(z.^2+y.^2);
    nu0 = x./(sqrt(R0.^2-y.^2));

    % find Cartesian points inside pyramid to interpolate
    p = find(R0>=obj.rmin & R0<=obj.rmax-obj.dr & mu0>=mumin & mu0<=mumax-dmu & ...
        nu0>=numin & nu0<=numax-dnu); % points for valid interpolation

    %image_initialization_time = toc

    %tic;
    % for each point to interpolate, find nearest previous neighbors, and
    %  distances from them along R, mu, nu directions
    imu = zeros(size(R0));
    Lmu = zeros(size(R0));
    imu(p) = floor((mu0(p) - mumin)/dmu) + 1;
    Lmu(p) = mu0(p) - muvec(imu(p))';

    inu = zeros(size(R0));
    Lnu = zeros(size(R0));
    inu(p) = floor((nu0(p) - numin)/dnu) + 1;
    Lnu(p) = nu0(p) - nuvec(inu(p))';

    iR = zeros(size(R0));
    LR = zeros(size(R0));
    iR(p) = floor((R0(p) - obj.rmin)/obj.dr) + 1;
    LR(p) = R0(p) - Rvec(iR(p))';

    %interpolation_initialization_time = toc
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %tic;
    Icart = zeros(size(R0));
    for ip = 1:length(p)
        q = p(ip);

        % differences to be used below, trying to save a few flops
        drmLR = obj.dr-LR(q);
        dmumLmu = dmu-Lmu(q);
        dnumLnu = dnu-Lnu(q);
        Icart(q) = Isph(iR(q),imu(q),inu(q)) ...
            * drmLR * dmumLmu * dnumLnu + ...
            Isph(iR(q)+1,imu(q),inu(q)) ...
            * LR(q) * dmumLmu * dnumLnu + ...
            Isph(iR(q),imu(q)+1,inu(q)) ...
            * drmLR * Lmu(q) * dnumLnu + ...
            Isph(iR(q),imu(q),inu(q)+1) ...
            * drmLR * dmumLmu * Lnu(q) + ...
            Isph(iR(q)+1,imu(q),inu(q)+1) ...
```

```
1106                * LR(q) * dmumLmu * Lnu(q) + ...
                    Isph(iR(q),imu(q)+1,inu(q)+1) ...
1108                * drmLR * Lmu(q) * Lnu(q) + ...
                    Isph(iR(q)+1,imu(q)+1,inu(q)) ...
1110                * LR(q) * Lmu(q) * dnumLnu + ...
                    Isph(iR(q)+1,imu(q)+1,inu(q)) ...
1112                * LR(q) * Lmu(q) * Lnu(q);
        end
1114    obj.rawData_cart(:,:,:,volIndex) = Icart/(obj.dr*dmu*dnu);
        %interpolation_time = toc
1116
    end
1118end
```

B.4 Experiment Class

```
1000 classdef ExperimentClass < handle
    %EXPERIMENTCLASS: Performs processess on data sets for an experiment
1002 %  organizes decorrelation code for a simple to use and expandable
    %  interface
1004 %  designed to be easily accessed for a GUI application
     properties
1006     % main arrays
         ultrasoundDataSeries;
1008     cumulativeDecorr;
         decorrelationMapSeries;
1010     decorrSumSeries;
         decorrSumSeriesROI;
1012     decorrVolume;
         averageDecorr;
1014     % ultrasound data parameters
         rmin;
1016     rmax;
         cartScalingFactor;
1018     sigma;
         interFrameTime;
1020     thetamin;
         thetamax;
1022     phimin;
         phimax;
1024     decorrThresh;
         % experiment parameters
1026     numDataSets;
         activeFolder;
1028     activeFolderDir;
         folderIndex;
1030     defaultDataFileName;
         totalThresh;
1032     totalThreshVolume;
         inSerialString;
1034     outSerialString;
         outSerialObj;
1036     inSerialObj;
         ROI_xRange;
1038     ROI_yRange;
         ROI_zRange;
1040 end

1042 methods
         function ExperimentClassSetParams(obj,thisrmin,thisrmax,thisthetamin,
     thisthetamax,thisphimin,thisphimax,thiscartScalingFactor,thissigma,
     thisinterFrameTime,thisdecorrthresh,thistotalThreshVolume)
1044         %EXPERIMENTCLASS Construct an instance of this class
             %   Detailed explanation goes here
1046         obj.rmin = thisrmin;
             obj.rmax = thisrmax;
1048         obj.cartScalingFactor = thiscartScalingFactor;
             obj.sigma = thissigma;
```

```
1050            obj.interFrameTime = thisinterFrameTime;
                obj.thetamin = thisthetamin;
1052            obj.thetamax = thisthetamax;
                obj.phimin = thisphimin;
1054            obj.phimax = thisphimax;
                obj.decorrThresh = thisdecorrthresh;
1056            obj.defaultDataFileName = 'bufApl0Out_0x0_0x0.data.dm.pmcr';

1058            obj.totalThreshVolume = thistotalThreshVolume;
        end
1060    function obj = ExperimentClass()

1062    end
        function obj = addNextDataSetViaFilename(obj, thisFileName)
1064        % Compute decorr of data set

1066        Dm = read_lbdump(thisFileName);
            tempDataSet = USDataClass(Dm.data,Dm.startTime, Dm.Info,obj.rmin,obj.rmax,
    obj.thetamin,obj.thetamax,obj.phimin,obj.phimax,obj.cartScalingFactor,obj.sigma,obj
    .interFrameTime);
1068        tempDataSet.scanConv_Frust();
            tempDataSet.compute3DDecorr();
1070        tempDataSet.decorrThresh = obj.decorrThresh;
            % compute ablated volume
1072
            if(isempty(obj.cumulativeDecorr))
1074            % set initial cumulative decorrelation to the result of the
                % first volume's decorr
1076            obj.cumulativeDecorr(1).decorr = tempDataSet.decorr;
                % compute decorrelation sum
1078            sizeOfVol = size(tempDataSet.decorr);
                obj.decorrSumSeries = sum(obj.cumulativeDecorr(1).decorr(:))/prod(
    sizeOfVol(1:3));
1080            % create mask of pixels which exceed the threshold
                obj.decorrelationMapSeries(1).decorrMap = tempDataSet.decorr;
1082            % remove elements below the threshold
                tempAblatedPoints = find(obj.decorrelationMapSeries(1).decorrMap >= obj.
    decorrThresh);
1084            obj.decorrelationMapSeries(1).decorrMap(tempAblatedPoints) = 1;
                %set elements above threshold to 1
1086            obj.decorrelationMapSeries(1).decorrMap(find(obj.decorrelationMapSeries
    (1).decorrMap ~= 1)) = 0;
                % each point above the threshold has a volume of dx^3, so
1088            % the number of points * dx^3 gives the volume of ablated
                % tissue
1090            obj.averageDecorr(1) = log10(mean(obj.cumulativeDecorr(1).decorr(:)));
                obj.decorrVolume(1) = .001*(length(tempAblatedPoints))*tempDataSet.dx^3;
1092        else
                % find max value between current decorrelation and previous
1094            % cumulative decorrelation
                obj.cumulativeDecorr(obj.folderIndex).decorr = max(obj.cumulativeDecorr(
    obj.folderIndex-1).decorr,tempDataSet.decorr);
1096            % compute sum
                sizeOfVol = size(tempDataSet.decorr);
```

```
1098                obj.decorrSumSeries(obj.folderIndex) = sum(obj.cumulativeDecorr(obj.
    folderIndex).decorr(:))/prod(sizeOfVol(1:3));
                    % create decorrelation map for current volume
1100                obj.decorrelationMapSeries(obj.folderIndex).decorrMap = obj.
    cumulativeDecorr(obj.folderIndex).decorr;
                    tempAblatedPoints = find(obj.decorrelationMapSeries(obj.folderIndex).
    decorrMap >= obj.decorrThresh);
1102                obj.decorrelationMapSeries(obj.folderIndex).decorrMap(tempAblatedPoints)
     = 1;
                    obj.decorrelationMapSeries(obj.folderIndex).decorrMap(find(obj.
    decorrelationMapSeries(obj.folderIndex).decorrMap ~= 1)) = 0;
1104                % each point above the threshold has a volume of dx^3, so
                    % the number of points * dx^3 gives the volume of ablated
1106                % tissue
                    interVal = obj.cumulativeDecorr(obj.folderIndex).decorr(obj.ROI_zRange
    (1):obj.ROI_zRange(2),obj.ROI_yRange(1):obj.ROI_yRange(2),obj.ROI_xRange(1):obj.
    ROI_xRange(2));
1108                obj.decorrSumSeriesROI(obj.folderIndex) = mean(interVal(:));
                    obj.averageDecorr(obj.folderIndex) = log10(mean(obj.cumulativeDecorr(obj
    .folderIndex).decorr(:)));
1110
                    obj.decorrVolume(obj.folderIndex) = .001*(length(tempAblatedPoints))*
    tempDataSet.dx^3; % in cm^3
1112            end
                % append ultrasound data to internal data struct
1114            obj.ultrasoundDataSeries = [obj.ultrasoundDataSeries tempDataSet];

1116        end
            function obj = initDataFolderGUI(obj)
1118            try
                    if ispc
1120                    basePath = strcat('C:\Users\',getenv('username'),'\Box\
    SiemensSC2000IQData');
                    elseif ismac
1122                    basePath = strcat('/Users/',getenv('USER'),'/box');
                    end
1124            catch
                    basePath = matlabroot;
1126            end
                obj.activeFolder =  uigetdir(basePath);
1128            fullDirectory  = dir(obj.activeFolder);
                obj.activeFolderDir = fullDirectory(3:end);
1130            obj.folderIndex = 1;
            end
1132
            function newFilePresent = checkFolder(obj)
1134            obj.activeFolderDir  = dir(obj.activeFolder);
                if ismac
1136                obj.activeFolderDir = obj.activeFolderDir(3:end);
                else
1138                obj.activeFolderDir = obj.activeFolderDir(3:end);
                end
1140            if(length(obj.activeFolderDir) >= obj.folderIndex)
                    newFilePresent = 1;
```

```
1142            else
                    newFilePresent = 0;
1144            end
            end
1146
            function nextDataSetInFolder(obj)
1148            if ispc
                    fullPath =strcat(obj.activeFolder,'\',{obj.activeFolderDir(obj.
    folderIndex).name},'\',obj.defaultDataFileName);
1150            elseif ismac
                    fullPath =strcat(obj.activeFolder,'/',{obj.activeFolderDir(obj.
    folderIndex).name},'/',obj.defaultDataFileName);
1152            end
                while(~exist(fullPath{1},'file'))
1154                pause(.01);
                    display('waiting for file');
1156            end
                obj.addNextDataSetViaFilename(fullPath{1});
1158            obj.folderIndex = obj.folderIndex+1;
            end
1160        function dataSlice = getDataSlice_cart(obj,direction,set,frame,index)
                switch direction
1162                case 'z'
                        dataSlice = squeeze(obj.ultrasoundDataSeries(set).rawData_cart(:,:,
    index,frame));
1164                case 'y'
                        dataSlice = squeeze(obj.ultrasoundDataSeries(set).rawData_cart(:,
    index,:,frame));
1166                case 'x'
                        dataSlice = squeeze(obj.ultrasoundDataSeries(set).rawData_cart(index
    ,:,:,frame));
1168                otherwise
                        dataSlice = 1;
1170            end
            end
1172        function dataSlice = getDataSlice_decorr(obj,direction,set,frame,index)
                switch direction
1174                case 'z'
                        dataSlice = squeeze(obj.ultrasoundDataSeries(set).decorr(:,:,index,
    frame));
1176                case 'y'
                        dataSlice = squeeze(obj.ultrasoundDataSeries(set).decorr(:,index,:,
    frame));
1178                case 'x'
                        dataSlice = squeeze(obj.ultrasoundDataSeries(set).decorr(index,:,:,
    frame));
1180                otherwise
                end
1182        end
            function dataSlice = getDataSlice_cumulativeDecorr(obj,direction,set,frame,index
    )
1184            switch direction
                    case 'z'
```

```
1186                    dataSlice = squeeze(obj.cumulativeDecorr(set).decorr(:,:,index,frame
    ));
                case 'y'
1188                    dataSlice = squeeze(obj.cumulativeDecorr(set).decorr(:,index,:,frame
    ));
                case 'x'
1190                    dataSlice = squeeze(obj.cumulativeDecorr(set).decorr(index,:,:,frame
    ));
                otherwise
1192            end
        end
1194
        function dataSlice = getDataSlice_decorrMask(obj,direction,set,frame,index)
1196            switch direction
                case 'z'
1198                    dataSlice = squeeze(obj.decorrelationMapSeries(set).decorrMap(:,:,
    index,frame));
                case 'y'
1200                    dataSlice = squeeze(obj.decorrelationMapSeries(set).decorrMap(:,
    index,:,frame));
                case 'x'
1202                    dataSlice = squeeze(obj.decorrelationMapSeries(set).decorrMap(index
    ,:,:,frame));
                otherwise
1204            end
        end
1206
        function dataSlice = getDataSlice_ROI(obj,direction,set,frame,index)
1208            subVolume = obj.ultrasoundDataSeries(set).rawData_cart(:,:,:,frame);
            maskVol = zeros(size(subVolume));
1210            maskVol(obj.ROI_zRange(1):obj.ROI_zRange(2),obj.ROI_yRange(1):obj.ROI_yRange
    (2),obj.ROI_xRange(1):obj.ROI_xRange(2)) = 1;
            %subVolume([1:obj.ROI_zRange(1),obj.ROI_zRange(2):end],[1:obj.ROI_yRange(1),
    obj.ROI_yRange(2):end],[1:obj.ROI_zRange(1),obj.ROI_zRange(2):end]) = 0;
1212            subVolume = subVolume .* maskVol;
            switch direction
1214                case 'z'
                    dataSlice = squeeze(subVolume(:,:,index,frame));
1216                case 'y'
                    dataSlice = squeeze(subVolume(:,index,:,frame));
1218                case 'x'
                    dataSlice = squeeze(subVolume(index,:,:,frame));
1220                otherwise
            end
1222        end

1224        function computeDecorrStats(obj, tempDataSet, dataIndex)
            if(isempty(obj.cumulativeDecorr))
1226                obj.cumulativeDecorr = tempDataSet.decorr;
                obj.decorrelationMapSeries;
1228                obj.decorrSumSeries;
                obj.decorrVolume;
1230            else
                obj.cumulativeDecorr = max(obj.cumulativeDecorr,tempDataSet.decorr);
```

```
1232            end
        end
1234
        function boolOut = decorrExceedsThresh(obj)
1236            if((obj.totalThresh) <= log10(obj.decorrSumSeriesROI(obj.folderIndex-1)))
                    boolOut =  1;
1238            else
                    boolOut =  0;
1240            end
        end
1242        function recomputeDecorr(obj)
            for currentVol = 1:length(obj.ultrasoundDataSeries)
1244                obj.ultrasoundDataSeries(currentVol).compute3DDecorr();
                    obj.ultrasoundDataSeries(currentVol).decorrThresh = obj.decorrThresh;
1246                if(currentVol == 1)
                        obj.cumulativeDecorr(currentVol).decorr = obj.ultrasoundDataSeries(
    currentVol).decorr;
1248                    % compute sum
                        sizeOfVol = size(obj.ultrasoundDataSeries(currentVol).decorr);
1250                    obj.decorrSumSeries(currentVol) = sum(obj.cumulativeDecorr(
    currentVol).decorr(:))/prod(sizeOfVol(1:3));
                    % create decorrelation map for current volume
1252                obj.decorrelationMapSeries(currentVol).decorrMap = obj.
    cumulativeDecorr(currentVol).decorr;
                    tempAblatedPoints = find(obj.decorrelationMapSeries(currentVol).
    decorrMap >= obj.decorrThresh);
1254                obj.decorrelationMapSeries(currentVol).decorrMap(tempAblatedPoints)
    = 1;
                    obj.decorrelationMapSeries(currentVol).decorrMap(find(obj.
    decorrelationMapSeries(currentVol).decorrMap ~= 1)) = 0;
1256                % each point above the threshold has a volume of dx^3, so
                    % the number of points * dx^3 gives the volume of ablated
1258                % tissue
                    obj.decorrVolume(currentVol) = .001*(length(tempAblatedPoints))*obj.
    ultrasoundDataSeries(currentVol).dx^3; % in cm^3
1260            else
                    obj.cumulativeDecorr(currentVol).decorr = max(obj.cumulativeDecorr(
    currentVol-1).decorr , obj.ultrasoundDataSeries(currentVol).decorr);
1262                % compute sum
                    sizeOfVol = size(obj.ultrasoundDataSeries(currentVol).decorr);
1264                obj.decorrSumSeries(currentVol) = sum(obj.cumulativeDecorr(
    currentVol).decorr(:))/prod(sizeOfVol(1:3));
                    % create decorrelation map for current volume
1266                obj.decorrelationMapSeries(currentVol).decorrMap = obj.
    cumulativeDecorr(currentVol).decorr;
                    tempAblatedPoints = find(obj.decorrelationMapSeries(currentVol).
    decorrMap >= obj.decorrThresh);
1268                obj.decorrelationMapSeries(currentVol).decorrMap(tempAblatedPoints)
    = 1;
                    obj.decorrelationMapSeries(currentVol).decorrMap(find(obj.
    decorrelationMapSeries(currentVol).decorrMap ~= 1)) = 0;
1270                % each point above the threshold has a volume of dx^3, so
                    % the number of points * dx^3 gives the volume of ablated
1272                % tissue
```

```
                    obj.decorrVolume(currentVol) = .001*(length(tempAblatedPoints))*obj.
    ultrasoundDataSeries(currentVol).dx^3; % in cm^3
1274                end
            end
1276        end
        function sendSerialData(obj)
1278        %pause(3)

1280        fprintf(obj.outSerialObj,'S');

1282    end
        function setUpSerialOutConnection(obj)
1284        obj.outSerialObj = serial(obj.outSerialString,'BaudRate', 115200);
            fopen(obj.outSerialObj);
1286    end
        function removeSerialConnection()
1288        fclose(obj.outSerialObj);
        end
1290    function updateROIDataSet(obj)
            for currN = 1:length(obj.ultrasoundDataSeries)
1292            interVal = obj.cumulativeDecorr(currN).decorr(obj.ROI_zRange(1):obj.
    ROI_zRange(2),obj.ROI_yRange(1):obj.ROI_yRange(2),obj.ROI_xRange(1):obj.ROI_xRange
    (2));
                obj.decorrSumSeriesROI(currN) = mean(interVal(:));
1294        end
        end
1296    function initExperiment(obj)
            interVal = obj.cumulativeDecorr(1).decorr(obj.ROI_zRange(1):obj.ROI_zRange
    (2),obj.ROI_yRange(1):obj.ROI_yRange(2),obj.ROI_xRange(1):obj.ROI_xRange(2));
1298        obj.decorrSumSeriesROI(1) = mean(interVal(:));
        end
1300    end
end
```

```
1000 function varargout = decorr3DGUI(varargin)
% DECORR3DGUI MATLAB code for decorr3DGUI.fig
1002 %      DECORR3DGUI, by itself, creates a new DECORR3DGUI or raises the existing
%      singleton*.
1004 %
%      H = DECORR3DGUI returns the handle to a new DECORR3DGUI or the handle to
1006 %      the existing singleton*.
%
1008 %      DECORR3DGUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in DECORR3DGUI.M with the given input arguments.
1010 %
%      DECORR3DGUI('Property','Value',...) creates a new DECORR3DGUI or raises the
1012 %      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before decorr3DGUI_OpeningFcn gets called.  An
1014 %      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to decorr3DGUI_OpeningFcn via varargin.
1016 %
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
1018 %      instance to run (singleton)".
%
1020 % See also: GUIDE, GUIDATA, GUIHANDLES

1022 % Edit the above text to modify the response to help decorr3DGUI

1024 % Last Modified by GUIDE v2.5 11-Apr-2019 20:50:12

1026 % Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
1028 gui_State = struct('gui_Name',       mfilename, ...
                    'gui_Singleton',  gui_Singleton, ...
1030                'gui_OpeningFcn', @decorr3DGUI_OpeningFcn, ...
                    'gui_OutputFcn',  @decorr3DGUI_OutputFcn, ...
1032                'gui_LayoutFcn',  [] , ...
                    'gui_Callback',   []);
1034 if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
1036 end

1038 if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
1040 else
    gui_mainfcn(gui_State, varargin{:});
1042 end
% End initialization code - DO NOT EDIT
1044


1046 % --- Executes just before decorr3DGUI is made visible.
function decorr3DGUI_OpeningFcn(hObject, eventdata, handles, varargin)
1048 % This function has no output args, see OutputFcn.
% hObject    handle to figure
1050 % eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% varargin   command line arguments to decorr3DGUI (see VARARGIN)

% Choose default command line output for decorr3DGUI
handles.output = hObject;

% Update handles structure
setSerialPopUps(hObject, eventdata,handles);
handles.currComputationTime = 0;
handles.maxbscan = 1;
handles.maxdecorr = 1;
guidata(hObject, handles);
axes(handles.axes12);
disableButtonsForStart(hObject, eventdata, handles);

imshow('index.png');
% UIWAIT makes decorr3DGUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = decorr3DGUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in dataSelect.
function dataSelect_Callback(hObject, eventdata, handles)
% hObject    handle to dataSelect (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.activeExperiment = ExperimentClass();
handles.activeExperiment.initDataFolderGUI();
handles.myString = sprintf(handles.activeExperiment.activeFolder);
set(handles.dataSelectionString, 'String', handles.myString);
enableButtonsForStart(hObject,eventdata,handles)
updateSettingsButton_Callback(hObject, eventdata, handles)
guidata( hObject, handles);
% --- Executes on slider movement.
function ySlider_Callback(hObject, eventdata, handles)
% hObject    handle to ySlider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

updateSliceImage_ROICorrected(hObject,eventdata,handles,'y');


% --- Executes during object creation, after setting all properties.
function ySlider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ySlider (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in updateSettingsButton.
function updateSettingsButton_Callback(hObject, eventdata, handles)
% hObject    handle to updateSettingsButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.sigmaLocal = (str2double(get(handles.windowSigma, 'String')));
handles.azimuthAngleLocal = 2*pi*(str2double(get(handles.azimuthAngle, 'String')))/360;
handles.elevationAngleLocal = 2*pi*(str2double(get(handles.rangeAngle, 'String')))/360;
handles.rmaxLocal = (str2double(get(handles.rMax, 'String')));
handles.rminLocal =0 ;
handles.cartScalingFactorLocal = (str2double(get(handles.cartScalingFactor, 'String')));
handles.frameRate = (str2double(get(handles.framerate, 'String')));
handles.decorrthreshLocal = 10^(str2double(get(handles.threshVal, 'String')));
handles.thetaminLocal = -handles.azimuthAngleLocal/2;
handles.thetamaxLocal = handles.azimuthAngleLocal/2;
handles.phiminLocal = -handles.elevationAngleLocal/2;
handles.phimaxLocal = handles.elevationAngleLocal/2;
handles.interFrameTimeLocal = 1/handles.frameRate;
handles.totalThreshLocal = (str2double(get(handles.totalThresh, 'String')));
handles.activeExperiment.ExperimentClassSetParams(handles.rminLocal,handles.rmaxLocal,
    handles.thetaminLocal,handles.thetamaxLocal,handles.phiminLocal,handles.phimaxLocal
    ,handles.cartScalingFactorLocal,handles.sigmaLocal,handles.interFrameTimeLocal,
    handles.decorrthreshLocal,handles.totalThreshLocal);
guidata( hObject, handles);
function rangeAngle_Callback(hObject, eventdata, handles)
% hObject    handle to rangeAngle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of rangeAngle as text
%        str2double(get(hObject,'String')) returns contents of rangeAngle as a double


% --- Executes during object creation, after setting all properties.
function rangeAngle_CreateFcn(hObject, eventdata, handles)
% hObject    handle to rangeAngle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
1156
1158 % --- Executes on slider movement.
     function xSlider_Callback(hObject, eventdata, handles)
1160 % hObject    handle to xSlider (see GCBO)
     % eventdata  reserved - to be defined in a future version of MATLAB
1162 % handles    structure with handles and user data (see GUIDATA)

1164 % Hints: get(hObject,'Value') returns position of slider
     %        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
1166 updateSliceImage_ROICorrected(hObject,eventdata,handles,'x');

1168 % --- Executes during object creation, after setting all properties.
     function xSlider_CreateFcn(hObject, eventdata, handles)
1170 % hObject    handle to xSlider (see GCBO)
     % eventdata  reserved - to be defined in a future version of MATLAB
1172 % handles    empty - handles not created until after all CreateFcns called

1174 % Hint: slider controls usually have a light gray background.
     if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
1176     set(hObject,'BackgroundColor',[.9 .9 .9]);
     end
1178


1180 % --- Executes on slider movement.
     function zSlider_Callback(hObject, eventdata, handles)
1182 % hObject    handle to zSlider (see GCBO)
     % eventdata  reserved - to be defined in a future version of MATLAB
1184 % handles    structure with handles and user data (see GUIDATA)

1186 % Hints: get(hObject,'Value') returns position of slider
     %        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
1188

     updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
1190

     % --- Executes during object creation, after setting all properties.
1192 function zSlider_CreateFcn(hObject, eventdata, handles)
     % hObject    handle to zSlider (see GCBO)
1194 % eventdata  reserved - to be defined in a future version of MATLAB
     % handles    empty - handles not created until after all CreateFcns called
1196

     % Hint: slider controls usually have a light gray background.
1198 if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
         set(hObject,'BackgroundColor',[.9 .9 .9]);
1200 end


1202

     % --- Executes on button press in beginButton.
1204 function beginButton_Callback(hObject, eventdata, handles)
     % hObject    handle to beginButton (see GCBO)
1206 % eventdata  reserved - to be defined in a future version of MATLAB
     % handles    structure with handles and user data (see GUIDATA)
1208 set(handles.beginButton, 'String', "Processing...");
     drawnow;
```

```
handles.activeN = 1;
if(handles.activeExperiment.checkFolder())
    handles.activeExperiment.nextDataSetInFolder();
    updateSliceImage_ROICorrected(hObject,eventdata,handles,'x');
    updateSliceImage_ROICorrected(hObject,eventdata,handles,'y');
    updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
    set(handles.beginButton, 'String', "Ready");
    updateFrameDropDown(hObject,eventdata,handles)
    set(handles.beginButton,'Enable','off')
    enableButtonsControlPanel(hObject, eventdata, handles);
    setROIRange_sliders(hObject, eventdata, handles);
    handles.activeExperiment.initExperiment();
else
    set(handles.beginButton, 'String', "The folder is empty");
end
guidata( hObject, handles);
function azimuthAngle_Callback(hObject, eventdata, handles)
% hObject    handle to azimuthAngle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of azimuthAngle as text
%        str2double(get(hObject,'String')) returns contents of azimuthAngle as a double


% --- Executes during object creation, after setting all properties.
function azimuthAngle_CreateFcn(hObject, eventdata, handles)
% hObject    handle to azimuthAngle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function windowSigma_Callback(hObject, eventdata, handles)
% hObject    handle to windowSigma (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of windowSigma as text
%        str2double(get(hObject,'String')) returns contents of windowSigma as a double


% --- Executes during object creation, after setting all properties.
function windowSigma_CreateFcn(hObject, eventdata, handles)
% hObject    handle to windowSigma (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function rMax_Callback(hObject, eventdata, handles)
% hObject    handle to rMax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of rMax as text
%        str2double(get(hObject,'String')) returns contents of rMax as a double


% --- Executes during object creation, after setting all properties.
function rMax_CreateFcn(hObject, eventdata, handles)
% hObject    handle to rMax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function cartScalingFactor_Callback(hObject, eventdata, handles)
% hObject    handle to cartScalingFactor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of cartScalingFactor as text
%        str2double(get(hObject,'String')) returns contents of cartScalingFactor as a
    double


% --- Executes during object creation, after setting all properties.
function cartScalingFactor_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cartScalingFactor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end


function framerate_Callback(hObject, eventdata, handles)
% hObject    handle to framerate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of framerate as text
%        str2double(get(hObject,'String')) returns contents of framerate as a double


% --- Executes during object creation, after setting all properties.
function framerate_CreateFcn(hObject, eventdata, handles)
% hObject    handle to framerate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function frameSlider_Callback(hObject, eventdata, handles)
% hObject    handle to frameSlider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider


% --- Executes during object creation, after setting all properties.
function frameSlider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to frameSlider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function computeNextVol_Callback(hObject, eventdata, handles)
% hObject    handle to computeNextVol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
1366
% Hints: get(hObject,'String') returns contents of computeNextVol as text
1368 %        str2double(get(hObject,'String')) returns contents of computeNextVol as a
     double
     %tic
1370 set(handles.computeNextVol, 'String', "Processing...");
drawnow;
1372 if(handles.activeExperiment.checkFolder())
     handles.activeExperiment.nextDataSetInFolder();
1374     handles.activeN = handles.activeExperiment.folderIndex-1;
     set(handles.computeNextVol, 'String', "Compute Next Volume");
1376 else
     set(handles.computeNextVol, 'String', "No more volumes remain");
1378     drawnow;
end
1380 updateSliceImage_ROICorrected(hObject,eventdata,handles,'x');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'y');
1382 updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
updateDecorrPlot(hObject,eventdata,handles)
1384 updateFrameDropDown(hObject,eventdata,handles)
drawnow;
1386 guidata( hObject, handles);

1388 % --- Executes during object creation, after setting all properties.
function computeNextVol_CreateFcn(hObject, eventdata, handles)
1390 % hObject    handle to computeNextVol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
1392 % handles    empty - handles not created until after all CreateFcns called

1394 % Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
1396 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
     defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
1398 end


1400


1402 function cycleDisplayVol_Callback(hObject, eventdata, handles)
% hObject    handle to cycleDisplayVol (see GCBO)
1404 % eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
1406
% Hints: get(hObject,'String') returns contents of cycleDisplayVol as text
1408 %        str2double(get(hObject,'String')) returns contents of cycleDisplayVol as a
     double


1410
% --- Executes during object creation, after setting all properties.
1412 function cycleDisplayVol_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cycleDisplayVol (see GCBO)
1414 % eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
1416
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function runVols_Callback(hObject, eventdata, handles)
% hObject    handle to runVols (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of runVols as text
%        str2double(get(hObject,'String')) returns contents of runVols as a double
set(handles.computeNextVol, 'String', "Processing...");
drawnow;
while(handles.activeExperiment.checkFolder())
    handles.activeExperiment.nextDataSetInFolder();
    handles.activeN = handles.activeExperiment.folderIndex-1;
    set(handles.computeNextVol, 'String', "Compute Next Volume");
    updateSliceImage_ROICorrected(hObject,eventdata,handles,'x');
    updateSliceImage_ROICorrected(hObject,eventdata,handles,'y');
    updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
    updateFrameDropDown(hObject,eventdata,handles)
    updateDecorrPlot(hObject,eventdata,handles)
    drawnow;
end
set(handles.computeNextVol, 'String', "No more volumes remain");
drawnow;

guidata( hObject, handles);



% --- Executes during object creation, after setting all properties.
function runVols_CreateFcn(hObject, eventdata, handles)
% hObject    handle to runVols (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in backVol.
function backVol_Callback(hObject, eventdata, handles)
% hObject    handle to backVol (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.activeN = handles.activeN - 1;
if(handles.activeN <= 1)
    handles.activeN =  1;
end
set(handles.selectVolPopup, 'Value',handles.activeN);
updateSliceImage_ROICorrected(hObject,eventdata,handles,'x');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'y');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
updateDecorrPlot(hObject,eventdata,handles);
guidata( hObject, handles);
% --- Executes on button press in forwardVol.
function forwardVol_Callback(hObject, eventdata, handles)
% hObject    handle to forwardVol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.activeN = handles.activeN + 1;
if(handles.activeN > handles.activeExperiment.folderIndex-1)
    handles.activeN =  handles.activeExperiment.folderIndex-1;
end
set(handles.selectVolPopup, 'Value',handles.activeN);
updateSliceImage_ROICorrected(hObject,eventdata,handles,'x');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'y');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
updateDecorrPlot(hObject,eventdata,handles);
guidata( hObject, handles);

% --- Executes on selection change in selectVolPopup.
function selectVolPopup_Callback(hObject, eventdata, handles)
% hObject    handle to selectVolPopup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns selectVolPopup contents as
    cell array
%        contents{get(hObject,'Value')} returns selected item from selectVolPopup
%handles.sigma = (str2double(get(handles.selectVolPopup, 'Value')));
% --- Executes during object creation, after setting all properties.
handles.activeN = get(handles.selectVolPopup, 'Value');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'x');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'y');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
updateDecorrPlot(hObject,eventdata,handles)
guidata( hObject, handles);
function selectVolPopup_CreateFcn(hObject, eventdata, handles)
% hObject    handle to selectVolPopup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in realTimeButton.
function realTimeButton_Callback(hObject, eventdata, handles)
% hObject    handle to realTimeButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of realTimeButton


% --- Executes on button press in prerecordButton.
function prerecordButton_Callback(hObject, eventdata, handles)
% hObject    handle to prerecordButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of prerecordButton

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

function threshVal_Callback(hObject, eventdata, handles)

% --- Executes on button press in pauseExperiment.
function pauseExperiment_Callback(hObject, eventdata, handles)
% hObject    handle to pauseExperiment (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in EndButton.
function EndButton_Callback(hObject, eventdata, handles)
% hObject    handle to EndButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in continueButton.
function continueButton_Callback(hObject, eventdata, handles)
% hObject    handle to continueButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.continueButton,'value',0);
set(handles.continueButton,'enable','off');
set(handles.pauseButton,'enable','on');
handles.activeExperiment.totalThresh =  str2double(get(handles.totalThresh,'String'));
updateDecorrPlot(hObject,eventdata,handles)
set(handles.currentStatusString,'String','Waiting for volumes');
```

```
drawnow;
if (handles.activeExperiment.decorrExceedsThresh())
    set(handles.currentStatusString,'String','Threshold Reached');
    set(handles.continueButton,'enable','on');
    set(handles.pauseButton,'enable','off');
else
    handles.activeExperiment.sendSerialData();
    while(~get(handles.pauseButton,'value') && ~handles.activeExperiment.
    decorrExceedsThresh())
        if(handles.activeExperiment.checkFolder())
            set(handles.currentStatusString,'String','Processing Volume');
            drawnow;
            tic;
            handles.activeExperiment.nextDataSetInFolder();
            handles.currComputationTime = toc;
            handles.activeN = handles.activeExperiment.folderIndex-1;
            updateSliceImage_ROICorrected(hObject,eventdata,handles,'x');
            updateSliceImage_ROICorrected(hObject,eventdata,handles,'y');
            updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
            updateDecorrPlot(hObject,eventdata,handles)
            updateFrameDropDown(hObject,eventdata,handles)
            computationTimeString = strcat(num2str(handles.currComputationTime),{' '},'s
    ');
            set(handles.compTimeString,'String',computationTimeString)
            drawnow
            %pause(.01);
        else
            set(handles.currentStatusString,'String','Waiting for volumes');
            drawnow;
        end

        pause(.01);
    end
end
if(get(handles.pauseButton,'value'))
    set(handles.currentStatusString,'String','Paused');
    handles.activeExperiment.sendSerialData();
end
set(handles.pauseButton,'value',0)
if (handles.activeExperiment.decorrExceedsThresh())
    set(handles.currentStatusString,'String','Threshold Reached');
    set(handles.continueButton,'enable','on');
    set(handles.pauseButton,'enable','off');
    handles.activeExperiment.sendSerialData();
end

drawnow;
guidata( hObject, handles);

% --- Executes on button press in resetButton.
function resetButton_Callback(hObject, eventdata, handles)
% hObject    handle to resetButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
disableButtonsForStart(hObject, eventdata, handles)
cla(handles.xCartVol);
cla(handles.yCartVol);
cla(handles.zCartVol);
cla(handles.xDecorrVol);
cla(handles.yDecorrVol);
cla(handles.zDecorrVol);
cla(handles.decorrPlot);
set(handles.selectVolPopup,'String',{' '});
set(handles.beginButton, 'String', 'Initialize');
removeSerialConnection();
guidata(hObject,handles);


function totalThresh_Callback(hObject, eventdata, handles)
% hObject    handle to totalThresh (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of totalThresh as text
%        str2double(get(hObject,'String')) returns contents of totalThresh as a double


% --- Executes during object creation, after setting all properties.
function totalThresh_CreateFcn(hObject, eventdata, handles)
% hObject    handle to totalThresh (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function updateSliceImage(hObject,eventdata,handles,direction)
tempSize = size(handles.activeExperiment.ultrasoundDataSeries(handles.activeN).
    rawData_cart);
decorrSliderValue = .1*(.0001+get(handles.dynRangeDecorrSlider,'Value'));
bscanSliderValue = 10*(.0001+get(handles.dynRangeBScanSlider,'Value'));
switch direction
    case 'x'
        SliderValue = floor((tempSize(1)-1)*get(handles.xSlider,'Value'))+1;

        % x axis
        axes(handles.xCartVol);

        imagesc(abs(log10(handles.activeExperiment.getDataSlice_cart(direction,handles.
    activeN,1,SliderValue))),[0 bscanSliderValue]);
        colormap(handles.xCartVol,gray);

        edgeVals = (squeeze(abs(handles.activeExperiment.getDataSlice_decorrMask(
    direction,handles.activeN,1,SliderValue))));
        alphamask(edgeVals,[256 0 0],.3);
```

```
1676            % decorr plots
                % x axis
1678            axes(handles.xDecorrVol);
                imagesc(squeeze(abs(handles.activeExperiment.getDataSlice_cumulativeDecorr(
        direction,handles.activeN,1,SliderValue))),[0, decorrSliderValue]);
1680            set(handles.xDecorrVol,'XTicklabel',[])
                set(handles.xDecorrVol,'YTicklabel',[])
1682            colormap(handles.xDecorrVol,hot(20));
        case 'y'
1684            SliderValue = floor((tempSize(2)-1)*get(handles.ySlider,'Value'))+1;
                % x axis
1686            axes(handles.yCartVol);

1688            imagesc(abs(log10(handles.activeExperiment.getDataSlice_cart(direction,handles.
        activeN,1,SliderValue))),[0 bscanSliderValue]);
                colormap(handles.yCartVol,gray);
1690            edgeVals = (squeeze(abs(handles.activeExperiment.getDataSlice_decorrMask(
        direction,handles.activeN,1,SliderValue))));
                alphamask(edgeVals,[256 0 0],.3);
1692            % decorr plots
                % x axis
1694            axes(handles.yDecorrVol);
                imagesc(squeeze(abs(handles.activeExperiment.getDataSlice_cumulativeDecorr(
        direction,handles.activeN,1,SliderValue))),[0, decorrSliderValue]);
1696            set(handles.yDecorrVol,'XTicklabel',[])
                set(handles.yDecorrVol,'YTicklabel',[])
1698            colormap(handles.yDecorrVol,hot(20));
        case 'z'
1700            SliderValue = floor((tempSize(3)-1)*get(handles.zSlider,'Value'))+1;
                % x axis
1702            axes(handles.zCartVol);

1704            imagesc(abs(log10(handles.activeExperiment.getDataSlice_cart(direction,handles.
        activeN,1,SliderValue))),[0 bscanSliderValue]);
                colormap(handles.zCartVol,gray);
1706            edgeVals = (squeeze(abs(handles.activeExperiment.getDataSlice_decorrMask(
        direction,handles.activeN,1,SliderValue))));
                alphamask(edgeVals,[256 0 0],.3);
1708            % decorr plots
                axes(handles.zDecorrVol);
1710            imagesc(squeeze(abs(handles.activeExperiment.getDataSlice_cumulativeDecorr(
        direction,handles.activeN,1,SliderValue))),[0, decorrSliderValue]);
                set(handles.zDecorrVol,'XTicklabel',[])
1712            set(handles.zDecorrVol,'YTicklabel',[])
                colormap(handles.zDecorrVol,hot(20));
1714        otherwise
    end
1716
    function resetAll(hObject,eventdata,handles,direction)
1718        set(handles.beginButton,'Enable','on')
        set(handles.beginButton,'String','Initialize')
1720
    function updateFrameDropDown(hObject,eventdata,handles)
1722        numFrameCell = {};
```

```
    for currCell = 1:handles.activeExperiment.folderIndex-1
        numFrameCell{currCell} = currCell;
    end
    set(handles.selectVolPopup, 'String',numFrameCell);
    set(handles.selectVolPopup, 'Value',handles.activeExperiment.folderIndex-1);
function updateDecorrPlot(hObject,eventdata,handles)
    threshValue = str2double(get(handles.totalThresh,'String'));
    currLength  = length(handles.activeExperiment.decorrSumSeries);
    threshSeries = threshValue*ones(currLength,1);
    axes(handles.decorrPlot);
    set(handles.decorrPlot, 'YScale', 'log')
    plot(1:currLength,handles.activeExperiment.averageDecorr,1:currLength,threshSeries
    ,1:currLength,log10(handles.activeExperiment.decorrSumSeriesROI))
    hold on;
    plot(handles.activeN, handles.activeExperiment.averageDecorr(handles.activeN),'r*')
    plot(handles.activeN, log10(handles.activeExperiment.decorrSumSeriesROI(handles.
    activeN)),'r*')
    hold off;
    drawnow;
    legend('Cumulative Decorrelation Sum','target')


% --- Executes on button press in recomputeDecorr.
function recomputeDecorr_Callback(hObject, eventdata, handles)
% hObject    handle to recomputeDecorr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.recomputeDecorr, 'String', 'Processing...');
drawnow;
handles.activeExperiment.recomputeDecorr();
updateSliceImage_ROICorrected(hObject,eventdata,handles,'x');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'y');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
updateDecorrPlot(hObject,eventdata,handles);
set(handles.recomputeDecorr, 'String', 'Recompute Decorr');
drawnow;
guidata(hObject, handles);
function sendSerialData(hObject,eventdata, handles)


% --- Executes on selection change in inSerialPopUp.
function inSerialPopUp_Callback(hObject, eventdata, handles)
% hObject    handle to inSerialPopUp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns inSerialPopUp contents as
    cell array
%        contents{get(hObject,'Value')} returns selected item from inSerialPopUp


% --- Executes during object creation, after setting all properties.
function inSerialPopUp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to inSerialPopUp (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in outSerialPopUp.
function outSerialPopUp_Callback(hObject, eventdata, handles)
% hObject    handle to outSerialPopUp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns outSerialPopUp contents as
    cell array
%        contents{get(hObject,'Value')} returns selected item from outSerialPopUp


% --- Executes during object creation, after setting all properties.
function outSerialPopUp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to outSerialPopUp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function setSerialPopUps(hObject, eventdata,handles)
tempDir = seriallist;
tempArr = {};
for n = 1:length(tempDir)
    tempArr{n} = tempDir(n);
end
set(handles.inSerialPopUp, 'String',tempArr);
set(handles.outSerialPopUp, 'String',tempArr);


% --- Executes on slider movement.
function dynRangeDecorrSlider_Callback(hObject, eventdata, handles)
% hObject    handle to dynRangeDecorrSlider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
```

```
%handles.dynRangeMaxDecorr = handles.dynRangeMinVal+4*(handles.dynRangeMinVal)*((get(
    handles.dynRangeDecorrSlider,'Value')));
updateSliceImage_ROICorrected(hObject,eventdata,handles,'x');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'y');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
drawnow;
guidata(hObject, handles);
% --- Executes during object creation, after setting all properties.
function dynRangeDecorrSlider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dynRangeDecorrSlider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function dynRangeBScanSlider_Callback(hObject, eventdata, handles)
% hObject    handle to dynRangeBScanSlider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
updateSliceImage_ROICorrected(hObject,eventdata,handles,'x');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'y');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
drawnow;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function dynRangeBScanSlider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dynRangeBScanSlider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

function disableButtonsForStart(hObject, eventdata, handles)
set(handles.resetButton, 'Enable', 'off');
set(handles.recomputeDecorr, 'Enable', 'off');
set(handles.beginButton, 'Enable', 'off');
set(handles.continueButton, 'Enable', 'off');
set(handles.computeNextVol, 'Enable', 'off');
set(handles.runVols, 'Enable', 'off');
set(handles.selectVolPopup, 'Enable', 'off');
set(handles.backVol, 'Enable', 'off');
set(handles.forwardVol, 'Enable', 'off');
```

```
set(handles.xSlider, 'Enable', 'off');
set(handles.ySlider, 'Enable', 'off');
set(handles.zSlider, 'Enable', 'off');
set(handles.dynRangeBScanSlider, 'Enable', 'off');
set(handles.dynRangeDecorrSlider, 'Enable', 'off');
set(handles.updateSettingsButton, 'Enable', 'off');
set(handles.pauseButton,'enable','off');

function enableButtonsForStart(hObject,eventdata,handles)
set(handles.resetButton, 'Enable', 'on');
set(handles.recomputeDecorr, 'Enable', 'on');
set(handles.beginButton, 'Enable', 'on');
set(handles.updateSettingsButton, 'Enable', 'on');

function enableButtonsControlPanel(hObject, eventdata, handles)
set(handles.continueButton, 'Enable', 'on');
set(handles.computeNextVol, 'Enable', 'on');
set(handles.runVols, 'Enable', 'on');
set(handles.selectVolPopup, 'Enable', 'on');
set(handles.backVol, 'Enable', 'on');
set(handles.forwardVol, 'Enable', 'on');
set(handles.xSlider, 'Enable', 'on');
set(handles.ySlider, 'Enable', 'on');
set(handles.zSlider, 'Enable', 'on');
set(handles.dynRangeBScanSlider, 'Enable', 'on');
set(handles.dynRangeDecorrSlider, 'Enable', 'on');


% --- Executes on button press in pauseButton.
function pauseButton_Callback(hObject, eventdata, handles)
% hObject    handle to pauseButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of pauseButton
set(handles.continueButton,'enable','on');
set(handles.pauseButton,'enable','off');



function dynRangeMin_Callback(hObject, eventdata, handles)
% hObject    handle to dynRangeMin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of dynRangeMin as text
%        str2double(get(hObject,'String')) returns contents of dynRangeMin as a double
tempStrinArr = get(handles.outSerialPopUp,'String');
handles.activeExperiment.outSerialString = tempStrinArr{get(handles.outSerialPopUp, '
    Value')};

handles.dynRangeMinVal = get(handles.dynRangeMin,'String');
guidata(hObject, handles);
```

```
% --- Executes during object creation, after setting all properties.
function dynRangeMin_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dynRangeMin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in sendButton.
function sendButton_Callback(hObject, eventdata, handles)
% hObject    handle to sendButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%handles.activeExperiment.setUpSerialOutConnection();
handles.activeExperiment.sendSerialData();


% --- Executes on button press in serialSetupButton.
function serialSetupButton_Callback(hObject, eventdata, handles)
% hObject    handle to serialSetupButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
tempStrinArr = get(handles.outSerialPopUp,'String');
handles.activeExperiment.outSerialString = tempStrinArr{get(handles.outSerialPopUp, '
    Value')}
display(handles.activeExperiment.outSerialString);
handles.activeExperiment.setUpSerialOutConnection();


% --- Executes on button press in roiSelectButton.
function roiSelectButton_Callback(hObject, eventdata, handles)
% hObject    handle to roiSelectButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.xCartVol);
[BW Xi_xslice, Yi_xSlice] = roipoly;
axes(handles.yCartVol);
[BW Xi_yslice, Yi_ySlice] = roipoly;
axes(handles.zCartVol);
[BW Xi_zslice, Yi_zSlice] = roipoly;


% --- Executes on slider movement.
function xroiSlider_1_Callback(hObject, eventdata, handles)
% hObject    handle to xroiSlider_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
updateSliceImage_ROICorrected(hObject,eventdata,handles,'x');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'y');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
handles.activeExperiment.updateROIDataSet();
drawnow;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function xroiSlider_1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to xroiSlider_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function xroiSlider_2_Callback(hObject, eventdata, handles)
% hObject    handle to xroiSlider_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
updateSliceImage_ROICorrected(hObject,eventdata,handles,'x');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'y');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
handles.activeExperiment.updateROIDataSet();
drawnow;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function xroiSlider_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to xroiSlider_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function yroiSlider_1_Callback(hObject, eventdata, handles)
% hObject    handle to yroiSlider_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
```

```
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
updateSliceImage_ROICorrected(hObject,eventdata,handles,'x');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'y');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
handles.activeExperiment.updateROIDataSet();
drawnow;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function yroiSlider_1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to yroiSlider_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function yroiSlider_2_Callback(hObject, eventdata, handles)
% hObject    handle to yroiSlider_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
updateSliceImage_ROICorrected(hObject,eventdata,handles,'x');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'y');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
handles.activeExperiment.updateROIDataSet();
drawnow;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function yroiSlider_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to yroiSlider_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function zroiSlider_1_Callback(hObject, eventdata, handles)
% hObject    handle to zroiSlider_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
```

```
2090 %        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
     updateSliceImage_ROICorrected(hObject,eventdata,handles,'x');
2092 updateSliceImage_ROICorrected(hObject,eventdata,handles,'y');
     updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
2094 drawnow;
     guidata(hObject, handles);
2096
     % --- Executes during object creation, after setting all properties.
2098 function zroiSlider_1_CreateFcn(hObject, eventdata, handles)
     % hObject    handle to zroiSlider_1 (see GCBO)
2100 % eventdata  reserved - to be defined in a future version of MATLAB
     % handles    empty - handles not created until after all CreateFcns called
2102
     % Hint: slider controls usually have a light gray background.
2104 if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
         set(hObject,'BackgroundColor',[.9 .9 .9]);
2106 end


2108
     % --- Executes on slider movement.
2110 function slider19_Callback(hObject, eventdata, handles)
     % hObject    handle to slider19 (see GCBO)
2112 % eventdata  reserved - to be defined in a future version of MATLAB
     % handles    structure with handles and user data (see GUIDATA)
2114
     % Hints: get(hObject,'Value') returns position of slider
2116 %        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
     updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
2118 drawnow;
     guidata(hObject, handles);
2120
     % --- Executes during object creation, after setting all properties.
2122 function slider19_CreateFcn(hObject, eventdata, handles)
     % hObject    handle to slider19 (see GCBO)
2124 % eventdata  reserved - to be defined in a future version of MATLAB
     % handles    empty - handles not created until after all CreateFcns called
2126
     % Hint: slider controls usually have a light gray background.
2128 if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
         set(hObject,'BackgroundColor',[.9 .9 .9]);
2130 end


2132
     function updateSliceImage_ROICorrected(hObject,eventdata,handles,direction)
2134 tempSize = size(handles.activeExperiment.ultrasoundDataSeries(handles.activeN).
         rawData_cart);
     decorrSliderValue = .1*(.0001+get(handles.dynRangeDecorrSlider,'Value'));
2136 bscanSliderValue = 10*(.0001+get(handles.dynRangeBScanSlider,'Value'));
     xMax = tempSize(3);
2138 yMax = tempSize(2);
     zMax = tempSize(1);
2140 xMid = floor(xMax/2);
     yMid = floor(yMax/2);
2142 zMid = floor(zMax/2);
```

```
handles.x_roi_left = floor((xMid-1)*get(handles.xroiSlider_1,'Value'))+1; %1:94
handles.x_roi_right = xMid+floor((xMid)*get(handles.xroiSlider_2,'Value')); % 94:188
handles.y_roi_left = floor((yMid-1)*get(handles.xroiSlider_3,'Value'))+1; %1:94
handles.y_roi_right = floor((yMid)+floor((yMid)*get(handles.xroiSlider_4,'Value')));
handles.z_roi_left = floor((zMid)*get(handles.yroiSlider_1,'Value'))+1; % 1:74
handles.z_roi_right = floor(zMid-1)+floor((zMid)*get(handles.yroiSlider_2,'Value'));
handles.activeExperiment.ROI_xRange = [handles.x_roi_left,handles.x_roi_right]
handles.activeExperiment.ROI_yRange = [handles.y_roi_left,handles.y_roi_right]
handles.activeExperiment.ROI_zRange = [handles.z_roi_left,handles.z_roi_right]

switch direction
    case 'x'
        SliderValue = floor((tempSize(1)-1)*get(handles.xSlider,'Value'))+1;

        % x axis
        axes(handles.xCartVol);
        tempPic = handles.activeExperiment.getDataSlice_cart(direction,handles.activeN
    ,1,SliderValue);

        imagesc(abs(log10(tempPic)),[0 bscanSliderValue]);
        colormap(handles.xCartVol,gray);

        edgeVals = (squeeze(abs(handles.activeExperiment.getDataSlice_decorrMask(
    direction,handles.activeN,1,SliderValue))));
        alphamask(edgeVals,[256 0 0],.3);
        hold on;
        rectangle('position',drawRect(handles.x_roi_left,handles.y_roi_left,handles.
    x_roi_right,handles.y_roi_right),'LineWidth',3)
        hold off;
        % decorr plots
        % x axis
        axes(handles.xDecorrVol);
        imagesc(squeeze(abs(handles.activeExperiment.getDataSlice_cumulativeDecorr(
    direction,handles.activeN,1,SliderValue))),[0, decorrSliderValue]);
        set(handles.xDecorrVol,'XTicklabel',[])
        set(handles.xDecorrVol,'YTicklabel',[])
        colormap(handles.xDecorrVol,hot(20));
    case 'y'
        SliderValue = floor((tempSize(2)-1)*get(handles.ySlider,'Value'))+1;
        % y axis
        axes(handles.yCartVol);
        % y axis
        tempPic = handles.activeExperiment.getDataSlice_cart(direction,handles.activeN
    ,1,SliderValue);

%         tempPic(:,1:handles.x_roi_left) = 0;
%         tempPic(:,handles.x_roi_right:end) = 0;
%         tempPic(1:handles.z_roi_left,:) = 0;
%         tempPic(handles.z_roi_right:end,:) = 0;

        imagesc(abs(log10(tempPic)),[0 bscanSliderValue]);
        colormap(handles.yCartVol,gray);
```

```
        edgeVals = (squeeze(abs(handles.activeExperiment.getDataSlice_decorrMask(
    direction,handles.activeN,1,SliderValue))));
        alphamask(edgeVals,[256 0 0],.3);
        hold on;
        rectangle('position',drawRect(handles.x_roi_left,handles.z_roi_left,handles.
    x_roi_right,handles.z_roi_right),'LineWidth',3)
        hold off;
        % decorr plots
        % x axis
        axes(handles.yDecorrVol);
        imagesc(squeeze(abs(handles.activeExperiment.getDataSlice_cumulativeDecorr(
    direction,handles.activeN,1,SliderValue))),[0, decorrSliderValue]);
        set(handles.yDecorrVol,'XTicklabel',[])
        set(handles.yDecorrVol,'YTicklabel',[])
        colormap(handles.yDecorrVol,hot(20));
    case 'z'
        SliderValue = floor((tempSize(1)-1)*get(handles.zSlider,'Value'))+1;
        % x axis
        axes(handles.zCartVol);
        % x axis
        tempPic = handles.activeExperiment.getDataSlice_cart(direction,handles.activeN
    ,1,SliderValue);
%         tempPic(:,1:handles.x_roi_left) = 0;
%         tempPic(:,handles.x_roi_right:end) = 0;
%         tempPic(1:handles.y_roi_left,:) = 0;
%         tempPic(handles.y_roi_right:end,:) = 0;

        imagesc(abs(log10(tempPic)),[0 bscanSliderValue]);
        colormap(handles.zCartVol,gray);
        edgeVals = (squeeze(abs(handles.activeExperiment.getDataSlice_decorrMask(
    direction,handles.activeN,1,SliderValue))));
        alphamask(edgeVals,[256 0 0],.3);
        hold on;
        rectangle('position',drawRect(handles.y_roi_left,handles.z_roi_left,handles.
    y_roi_right,handles.z_roi_right),'LineWidth',3)
        hold off;
        % decorr plots
        axes(handles.zDecorrVol);
        imagesc(squeeze(abs(handles.activeExperiment.getDataSlice_cumulativeDecorr(
    direction,handles.activeN,1,SliderValue))),[0, decorrSliderValue]);
        set(handles.zDecorrVol,'XTicklabel',[])
        set(handles.zDecorrVol,'YTicklabel',[])
        colormap(handles.zDecorrVol,hot(20));
    otherwise
end


% --- Executes on slider movement.
function xroiSlider_3_Callback(hObject, eventdata, handles)
% hObject    handle to xroiSlider_3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
```

```
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
updateSliceImage_ROICorrected(hObject,eventdata,handles,'x');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'y');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
drawnow;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function xroiSlider_3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to xroiSlider_3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function xroiSlider_4_Callback(hObject, eventdata, handles)
% hObject    handle to xroiSlider_4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
updateSliceImage_ROICorrected(hObject,eventdata,handles,'x');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'y');
updateSliceImage_ROICorrected(hObject,eventdata,handles,'z');
drawnow;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function xroiSlider_4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to xroiSlider_4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

function setROIRange_sliders(hObject, eventdata, handles)
tempSize = size(handles.activeExperiment.ultrasoundDataSeries(handles.activeN).
    rawData_cart);
xMax = tempSize(3);
yMax = tempSize(2);
zMax = tempSize(1);
xMid = floor(xMax/2);
yMid = floor(yMax/2);
zMid = floor(zMax/2);

```

```
handles.x_roi_left = floor((xMid-1)*get(handles.xroiSlider_1,'Value'))+1; %1:94
handles.x_roi_right = xMid+floor((xMid)*get(handles.xroiSlider_2,'Value')); % 94:188
handles.y_roi_left = floor((yMid-1)*get(handles.xroiSlider_3,'Value'))+1; %1:94
handles.y_roi_right = floor((yMid)+floor((yMid)*get(handles.xroiSlider_4,'Value')));
handles.z_roi_left = floor((zMid)*get(handles.yroiSlider_1,'Value'))+1; % 1:74
handles.z_roi_right = floor(zMid-1)+floor((zMid)*get(handles.yroiSlider_2,'Value'));
handles.activeExperiment.ROI_xRange = [handles.x_roi_left,handles.x_roi_right]
handles.activeExperiment.ROI_yRange = [handles.y_roi_left,handles.y_roi_right]
handles.activeExperiment.ROI_zRange = [handles.z_roi_left,handles.z_roi_right]
guidata(hObject, handles);


function rectInput = drawRect(p1,p2,p3,p4)
    pWidth = p3-p1;
    pHeight = p4-p2;
    rectInput = [p1,p2,pWidth,pHeight];
```

APPENDIX C OTHER CODE

C.1 ROI R CODE

```
#!/usr/local/bin/rscript
library("R.matlab")
library('ROCR')
# parse input commands
args <- commandArgs(TRUE)
mat_filename <- args[1]
# Import Data
imported_mat_data <- readMat(mat_filename)
decorrArr <- imported_mat_data[1]
labelArr <- imported_mat_data[2]
# logistic regression
#myModel <- glm(labelArr ~ decorrArr, data = imported_mat_data, family= binomial)
#decorrPredict <- predict(myModel, type = 'response')
# ROC creation
#ROCDecorr <- prediction(decorrPredict, imported_mat_data$labelArr)
ROCDecorr <- prediction(imported_mat_data$decorrArr, imported_mat_data$labelArr)
ROCPerformance <- performance(ROCDecorr, 'tpr','fpr')
auc.perf = performance(ROCDecorr, measure = "auc")
# plot
png('rocPlot2.png',width=3.25,height=3.25,units="in",res=500)
plot(ROCPerformance)
aucVal <- unlist(auc.perf@y.values)
dev.off()
writeMat("ROCOutput.mat", aucVal = aucVal, ROC_fpr = unlist(ROCPerformance@x.values),ROC
    _tpr = unlist(ROCPerformance@y.values ), ROC_cutoff = unlist(ROCPerformance@alpha.
    values) )
```

C.2 ARDUINO CODE

C.3 SIEMENS SCANNER AUTOMATION CODE