

# TI PROJECT

## ACOUSTOELASTIC EVALUATION OF TISSUE MATERIAL PROPERTIES

### TEAM MEMBERS

BOGDAN DZYUBAK

JOE HELFENBERGER

JONATHAN MEYER

MATTHEW PARLATO

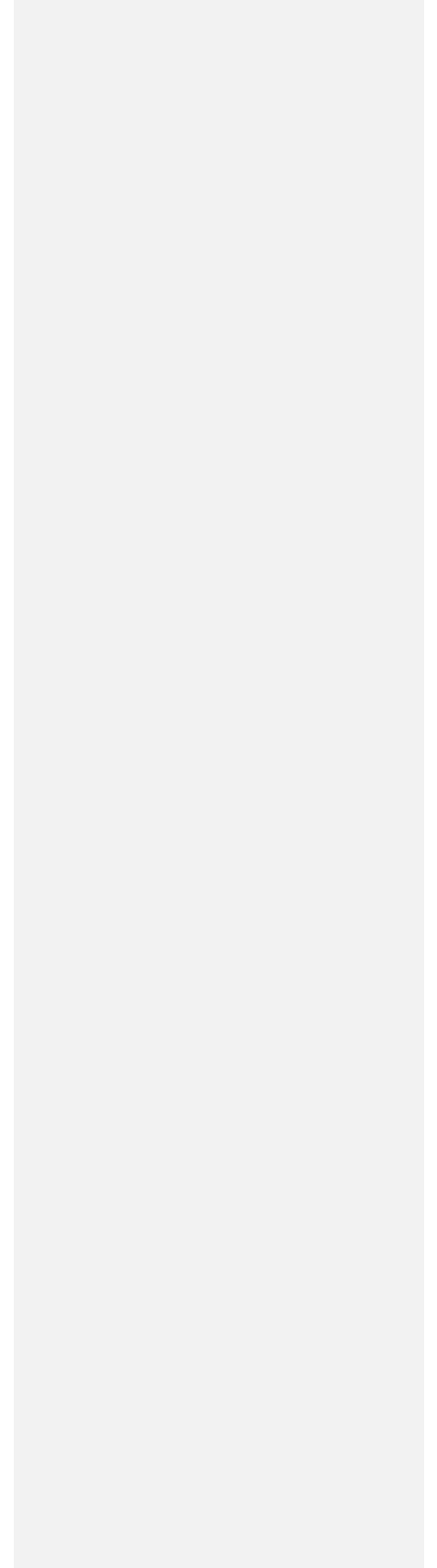
### ADVISOR

DR. WALTER BLOCK

### CLIENTS

DR. RAY VANDERBY

DR. HIROHITO KOBAYASHI



## TABLE OF CONTENTS

Table of Contents .....	Error! Bookmark not defined.2
Abstract .....	4
Introduction .....	4
Background .....	5
Ultrasound .....	5
Acoustoelasticity .....	6
Goals .....	8
Hardware .....	8
Final Algorithm .....	10
Image Normalization .....	10
Object Isolation .....	11
Convolution and Downsampling .....	12
Stiffness vs. Strain .....	12
Mapping .....	14
Alterations to the Original Algorithms .....	15
Image Normalization .....	15
Object Isolation .....	16
Filtration and Downsampling .....	16
Stiffness vs. Strain Calculation .....	16
Results .....	16
Hardware .....	16

## TI DSP Project: Final Report

Page 3

mapping.....	17
Processing.....	17
Future Work.....	19
Conclusion .....	20
Appendices .....	21
Appendix 1—Original Clipping Algorithm.....	21
Appendix 2—Original Stiffness vs. Strain Algorithm.....	25
Appendix 3—Revised and Combined Algorithm.....	30
Appendix 4—Calculations and Equations.....	37
Appendix 5—Product Design Specifications.....	40
References.....	44

### ABSTRACT

Ultrasound, a common imaging technique, is used to provide information on tissue structure. With additional processing, this data can yield the change in a tissue's stiffness as it is strained. Physicians and researchers could use this information to identify damaged locations in tendons and other structures.

The final goal of this project is to implement a program on a DSP chip which will perform this processing quickly enough for a physician to conduct the analysis during a patient visit. This semester, our goals were twofold:

1. Combine and revise a set of existing MATLAB ultrasound analysis algorithms to decrease their execution time
2. Develop an interface between a computer and a DSP chip

We successfully combined the algorithms and reduced their processing time by 87% while introducing a 5.9% difference in the final pixel calculations. The revised algorithm also processed a 30-frame, 434×532-pixel ultrasound video, a task which the original algorithm could not perform. We also developed an interface between a computer and DSP chip that was capable of taking a movie file from a computer, transcoding it, and displaying it on an output device.

Future work will include reducing the percent difference between the algorithm output and the original algorithm's output, transcoding the algorithm into C, and expanding the interface to send more parameters to the chip from the computer as well as send multiple frames to the chip at the same time.

### INTRODUCTION

Ultrasound is a widespread, non-invasive imaging technique with no reported side effects, used to evaluate internal organs and tissue structures by measuring their sound reflection properties (Brown 2003; Ophir 1998). This technique works by sending a high frequency sound pulse throughout the tissue and measuring the properties and timing of the returning signal. Data from the returning signal is interpreted and displayed as a picture, which is used to find tissue abnormalities so that proper treatment may be administered. Abnormalities generally result in changes in the stiffness and structure of tissues and can be seen in an ultrasound image. However, only major changes in the stiffness or structure of a tissue will be apparent in a standard ultrasound image.

It is possible to improve the diagnostic capability of ultrasound by using additional data already contained in the ultrasound data set. One way of doing this utilizes the theory of acoustoelasticity, in that tissue stiffness changes as it is deformed. The stiffness of a damaged tissue will change differently with deformation than the stiffness of a healthy tissue, thus allowing medical practitioners to utilize ultrasound technology to diagnose a variety of conditions ranging from tendon damage to cancer.

Incorporating the properties of acoustoelasticity into ultrasound technology can be especially useful in the diagnosis of conditions involving low-level tissue damage. The detection and assessment of low-level tissue damage is very important; if left untreated, it can lead to a loss

of tissue function. Although changes in tissue stiffness during deformation ~~are is~~ not tested by standard ultrasound, research by Dr. Vanderby and Dr. Kobayashi (2008) has shown that the relationship between a tissue's stiffness and the amount of deformation it undergoes can be used to help diagnose low-level tissue damage. As tissue stiffness can be related to the *intensity* of the ultrasound image, an algorithm to analyze ultrasound data could provide information about how tissue stiffness changes with deformation. Dr. Vanderby and Dr. Kobayashi have used two MATLAB algorithms to analyze a series of ultrasound images. Their research is aimed at determining the feasibility and accuracy of using ultrasound technology to diagnose pathologies based on the tissues' changes in stiffness at a constant rate of strain.

The algorithms that carry out the stiffness-strain analysis on ultrasound media are currently being run on a computer, with only the capability to analyze stored images. Analysis using these algorithms on a large data set ( $\geq 50$  MB) takes more than a few hours to run. This time delay limits the practicality of using this analysis in a clinical or research setting.

To increase the speed with which these algorithms execute, we propose using these algorithms in combination with a Digital Signal Processing (DSP) chip. This could potentially allow for data processing in near real time. The increase in speed would allow for:

- Analyzing data on-site in a clinical or research setting
- Allowing the user to immediately verify the quality and accuracy of the data
- Allowing the data to be available quickly for diagnosis

Dr. Vanderby's and Dr. Kobayashi's research offers advancement to ultrasound imaging that could greatly improve current diagnostic tools. The chip will greatly speed up data analysis because of its ability to perform simple calculations quickly. Texas Instruments has ~~provided~~ donated a DSP chip and development kit for use in this project. For detailed design specifications of our proposed device, see Appendix 1.

## BACKGROUND

### ULTRASOUND

Ultrasound is a powerful imaging technique used to evaluate internal tissue structure. Because of its price and lack of side effects, ultrasound is one of the most popular imaging techniques (Vanderby 2008). It can be used for analysis of internal tissue structure as well as functional and structural defects. Because ultrasound waves pass through low density tissues more easily, it is useful for examining fluid-filled organs such as the uterus in pregnancy, and soft organs such as the liver. Special applications of ultrasound can detect congenital heart disease, cysts, and tumors.

An ultrasound machine operates by a transducer sending a pulse of high-frequency sound through an area of interest in the body and measuring the fraction of energy reflected back as well as the timing of the returning signal. The timing is used to calculate distance to the object. The fraction of reflected energy is proportional to the tissue density and velocity of sound in the tissue.

The velocity of sound in the tissue is proportional to the stiffness of the tissue (Webster 1998). Research by Pourcelot, *et al.*, found that a correlation exists between the ultrasonic wave propagation velocity of the equine superficial digital flexor tendon and the force applied to this tendon along its main axis (2005). The tissue's wave propagation velocity was shown to be directly related to its increasing stiffness. Thus, the intensity in ultrasound images is proportional to the stiffness of the tissue:

$$\text{Intensity} \sim \text{Sound Velocity} \sim \text{Stiffness}$$

Analyzing changes in intensity as a structure is being stretched can yield information about its changing stiffness.

Formatted: Indent: First line: 0"

### ACOUSTOELASTICITY

Acoustoelasticity is the relationship between the velocity of sound in a material and the material's stiffness and stress. The material's stress is proportional to the strain (normalized change in length) applied to it. Stiffness is defined as the force required for a given amount of tissue deformation (Figure 1). Studies have shown a positive correlation between stiffness and strain in tissues. As a material is stretched, it becomes stiffer, causing more sound to be reflected.

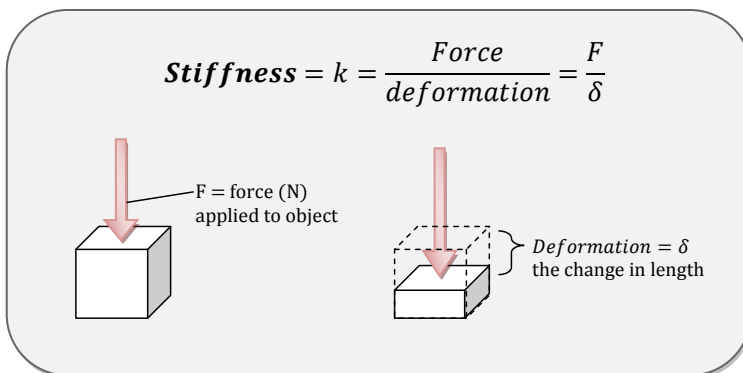


Figure 1: A schematic diagram representing how stiffness,  $k$  is defined and calculated. Block arrows represent a downward force applied to a specific object, with relative deformation occurring only in the vertical direction. The change in length  $\delta$  is calculated by subtracting final length from original length,  $L_{final} - L_{initial}$ .

One of the techniques using this property is elastography. By using the acoustoelastic properties of tumors, ultrasound images before and after tumor compression can be analyzed to better resolve tumors than using regular ultrasound. Though a tumor ~~or a suspicious cancerous~~

growth can be 5-28 times stiffer than normal soft tissue, the rate at which its stiffness increases with strain is also significantly different and can further aid in detection (Garra, *et al* 1997; Figure 2).

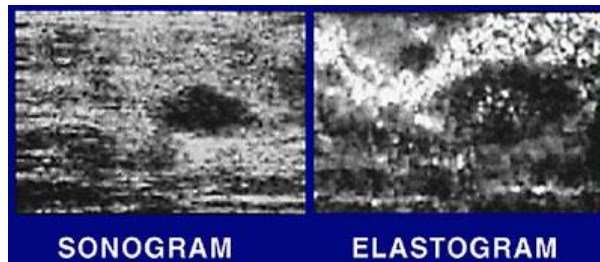


Figure 2: A sonogram and an elastogram of the same tissue area. The elastogram increases the recognition of the large tumor and also detects another, smaller tumor far more clearly than the sonogram. Obtained from Ophir (1998).

The acoustoelastic properties of tissues can also differentiate *damaged* tissues based on changing stiffness (Vanderby and Kobayashi, 2008). By plotting mechanically calculated strain and the force required for ~~is this~~ deformation, a relationship between stiffness and strain can be found (Figure 3; Vanderby and Kobayashi 2008). The difference between the  $\frac{\text{stiffness}}{\text{strain}}$  slopes can be used to differentiate the tissues. This relationship difference shows ~~how~~ that although damaged tendons are generally less stiff than ~~healthy damaged~~ tendons, their stiffness increases at a greater rate.

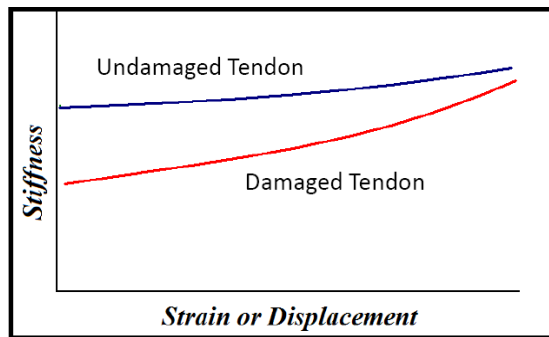


Figure 3: Graph of stiffness versus strain relationship in damaged (red) and undamaged (blue) tendons. Stiffness and strain forces calculated in vitro. Borrowed from Kobayashi and Vanderby (2008).

As damaged tendons have been found to have different ~~positive correlation~~ stiffness vs. strain slopes than healthy tissue in vitro, the use of ultrasound can be used to detect stiffness in vivo. By setting pre-specified strain conditions, ultrasound video can be analyzed to determine the

stiffness-strain relationship. An analysis algorithm was developed by Dr. Vanderby and Dr. Kobayashi to determine the stiffness-strain slope of the a tissue in question. ~~As development continues~~Further research hopes to determine; the stiffness-strain relationship ~~can be accurately found~~ in a non-invasive way.

## GOALS

The ultimate goal of this project is to create a diagnostic device that measures and displays the distribution of stiffness vs. strain in a tissue within minutes of data acquisition. This goal will be reached in three stages, which span over the course of several semesters.

### ➤ Goal 1—This Semester

- Optimize the client's algorithm so that ~~it they is are~~ capable of analyzing data quickly and efficiently
- Focus on the speed and accuracy of the revised algorithm compared to that of the original algorithm
- Process 30 frames of 434 x 532 pixels (~~half of the~~ total size of ~~provided ultrasound files from~~ our client's ultrasound files) in under 5 minutes
- Interface the DSP chip with a computer

### ➤ Goal 2—Next Few Semesters

- Implement the revised algorithm on a DSP chip
- ~~This device will function as a~~Develop a working prototype
- ~~Device will I~~ interface the prototype with a computer and analyze ultrasound data soon after it has been gathered

### ➤ Goal 3—Future

- ~~Creating~~ a device that is robust enough to be used in clinical and laboratory settings
- Device will perform analysis on the ultrasound data and generate output within minutes

## HARDWARE

Electronically, an image consists of a series of pixels, which are stored as numbers representing either the pixel's shade of gray or its mix of red, blue, and green. Image processing involves performing a series of basic mathematical operations on each of those numbers. Thus, the speed of an image processor relies on its ability to quickly perform many iterations of relatively simple calculations.

Personal computers are designed to handle data manipulation effectively and are not optimized for performing many basic calculations necessary for image processing. An alternative to a personal computer is a digital signal processor (DSP). DSPs are designed specifically to perform simple mathematical operations quickly (see Figure 4). Also, DSPs have the capability for parallel processing similar to that of multi-core computer processors. Furthermore, DSPs are hardwired with separate memory components for their programs and their data, which makes memory management simpler and reduces the amount of resources spent on it. The data on a DSP chip is stored in RAM which is about 1000 times faster than a hard drive used in a computer. Because of all of these factors, a DSP chip was chosen for this project as one of the ways to decrease processing time and allow on site data analysis.



	Data Manipulation	Math Calculation
Typical Applications	Word processing, database management, spread sheets, operating systems, etc.	Digital Signal Processing, motion control, scientific and engineering simulations, etc.
Main Operations	data movement ( $A \rightarrow B$ ) value testing ( <i>If <math>A=B</math> then ...</i> )	addition ( $A+B=C$ ) multiplication ( $A \times B=C$ )

**Figure 4.** Explanation of data manipulation and mathematical calculation; both of which are capabilities expected of typical processors.

The increase in processing speed with a DSP chip over that with a computer is difficult to predict. Due to the high rate at which the chip operates and the comparatively low amount of RAM it has (128 MB compared to a PC's 1-2 GB), the speed with which a program runs on the DSP chip depends on how it is written (particularly the way it addresses memory). Additionally, when large sets of data such as ultrasound video need to be analyzed, they cannot be stored on the chip because the chip uses its RAM for data storage. Thus, they must be transferred to the chip piece by piece, analyzed, and transferred back. The transfer of files occurs over the Ethernet interface and is slower than the processing. Therefore, the number of data transfers that need to be performed directly affects the processing time for a data set.

Two programs will be necessary for the chip's operation – the ultrasound stiffness vs. strain calculation program itself and the interface program that controls the transfer of files between the chip and the computer. Programs on the chip are stored in flash memory in machine code. A compiler called Code Composer Studio (v. 3.3), provided by Texas Instruments, can compile C code into an executable for the chip. It also contains a driver that is used to load programs onto the DSP chip's flash memory via USB. However, the Ethernet interface needs to be set up for data transfer. Texas Instruments provides example source code for a program that transfers a movie file onto the chip, decodes it, and outputs it to a display device. This program can be recompiled and altered so as to become the template for the DSP chip interface needed for this project.

The interface program will involve a component that runs on the computer and allows the user to input parameters that are used in the on-chip processing. Once the values are selected, the ultrasound sound video will be sent to the chip frame by frame and processed with the stiffness vs. strain algorithm. The result of the processing, a map of the rate of change of stiffness over strain between frames, will be sent back to the computer. The data flow is outlined below in Figure 5.

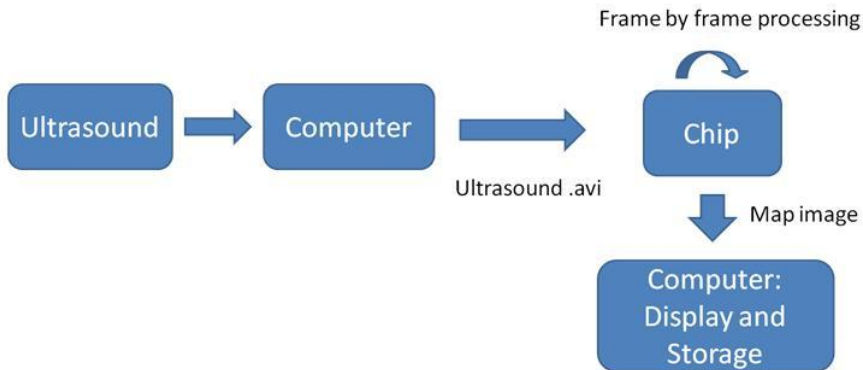


FIGURE 5: DATA FLOW FOR ULTRASOUND DATA. PROCESSING WITH THE STIFFNESS VS. STRAIN ALGORITHM IS PERFORMED ON THE CHIP.

A Texas Instruments DSP chip and development kit was made available by the UW-Madison BME Department to be used in this project. The chip, a TMS320C6437, has 600 MHz of processing power. Its development board has 128 Mb RAM for data storage and 64 Mb Flash memory for program storage. The chip is optimized for multimedia processing and contains video and audio output ports for display.

## FINAL ALGORITHM

The algorithm used in this project loads an ultrasound movie, allows the program operator to select an area of interest, calculates the relative rate of change of intensity (i.e. stiffness) of small blocks of the area of interest, and outputs a frame from the ultrasound video with the area of interest color-coded to indicate which regions of it have a higher rate of stiffness relative to the other regions. The sections of the algorithm are described below in more detail.

### IMAGE NORMALIZATION

Since image intensity is proportional to stiffness and stiffness is proportional to tissue density, if the density of a tendon is known, its stiffness can be calculated from the intensity of the ultrasound pixel.

Unfortunately, there are currently no reliable methods of determining absolute tendon density from ultrasound alone, so the exact stiffness of the tendon cannot be calculated. Instead, the algorithm uses relative stiffnesses to compare different parts of the tendon. The algorithm normalizes the intensity values of each frame in the ultrasound video to values between 0 and 1 with the following steps:

- The images are originally in RGB format - a three-dimensional matrix format in which each pixel has three numbers (relative amounts of red, green, and blue color associated with it).

They are converted to intensity format - a two-dimensional matrix with values ranging between 0 and 1.

- The images are then normalized to allow calculation of relative stiffness. The maximum intensity value of each frame is identified, and all of the intensity values of the frame are divided by it. The algorithm then stores each normalized frame of the movie as an array of matrices.

### OBJECT ISOLATION

The purpose of the object isolation algorithm is to prevent unnecessary processing and thus save time. It does this by isolating the region which the user is interested in so that the subsequent algorithms can limit their processing to the isolated region, rather than the entire frame. The mask is calculated from the first frame, and then used for all of the subsequent frames in the video.

The object isolation accepts a single normalized intensity image as its input, and outputs a binary matrix (the “binary mask”) with the same dimensions as the image, in which the region of interest pixels are 1’s, and the other values are 0’s. The binary mask then serves as a key for the subsequent algorithms to determine if a given pixel should be processed or ignored.

The algorithm consists of the following general steps:

- The binary mask is initialized as a matrix of zeros, with the same dimensions as the input image. Each value in this output matrix corresponds to a pixel in the input image.
- The input image is displayed so that the user can click on the object to be isolated, which should consist of a region of pixels with a higher intensity than their surrounding pixels (i.e. a “bright” region).
- The intensity of the pixel that the user selected, with the pixels three layers above, below, and to either side of it (a square containing 49 pixels total), are averaged. This “threshold average” then becomes the basis for determining whether or not surrounding pixels belong to the area of interest.
- The algorithm then processes the pixels in a series of concentric shells radiating outward from the selected area. If a pixel has a greater intensity than the threshold average, or falls within a specified range below the threshold average, it is added to the area of interest (i.e. its corresponding “0” in the binary mask is replaced with a “1”).

The intensity of a tissue can vary significantly throughout the region; often, tendons have a bright spot, and fade to a lower intensity near their edges. Thus, the threshold is constantly adjusted so that if a gradual fading occurs, the edges of the tissue can still be included.

When a pixel with an intensity below the threshold average is added to the area of interest, the threshold average is recalculated to include it. However, if the same process were used for pixels above the threshold, the threshold average could potentially increase during processing of a bright spot, preventing a dim region just beyond it from being included in the area of interest. Thus, when a pixel with an intensity value at or above the threshold average is added to the area of interest, a value equal to the threshold average is averaged in with the threshold average. In summary, when the algorithm includes a pixel dimmer than the threshold average, the threshold average decreases. When the algorithm includes a pixel brighter than the threshold average, the threshold average becomes more difficult to change.

- When the algorithm reaches a concentric shell which doesn't include any pixels which can be added to the area of interest, it stops; the area of interest is complete.
- The area of interest is then outlined on the input image, so that the user can confirm that the correct region of interest was selected.

### CONVOLUTION AND DOWNSAMPLING

Unfortunately, ultrasound images are noisy; their pixels are often randomly lighter or darker than the appropriate intensity for the pixel. Thus, if the change in intensity for regions of the area of interest were calculated on a pixel-by-pixel basis, the results would be aberrant.

To avoid this problem, the area of interest is divided into a series of blocks, and the average intensity of each block is used instead of the intensities of the individual pixels. This is done using a matrix operation called a convolution. The process is as follows:

- For each frame of the ultrasound video, a small matrix is created containing the pixel intensities of the area of interest and its surrounding region (the dimensions of the small matrix are determined by the height and width of the area of interest).
- A matrix of 1's, called the "kernel", is created with dimensions specified by the user. The dimensions of this matrix determine the resolution of the final output of the algorithm; the larger the kernel, the lower the resolution.
- The small matrix containing the area of interest is then convolved with the kernel. In this matrix operation, the kernel is centered, sequentially, over each element of the matrix, the average values are placed into a separate matrix. Thus, each element in the new matrix becomes the average of its equivalent, as well as the pixels neighboring its equivalent, in the old matrix.
- The image is then downsampled to reduce the number of calculations performed by the subsequent portion of the algorithm. This is done in proportion to the dimensions of the kernel; if the kernel is  $w$  columns wide and  $h$  rows tall, then every  $w$ 'th pixel in every  $h$ 'th row of the small matrix is selected and added to a new downsampled matrix, which looks like a miniature version of the original video frame.
- The process is then repeated for each frame in the video.

### STIFFNESS VS. STRAIN

The intensity of the downsampled matrix is then analyzed and the stiffness vs. strain distribution is calculated. For every pixel of the downsampled matrix, two values are calculated, called *top* and *base*:

$$Base = \frac{(1 + pixel's\ intensity\ value\ in\ the\ first\ frame)^2}{(1 - pixel's\ intensity\ value\ in\ the\ first\ frame)^2}$$

$$Top = \frac{(1 + pixel's\ intensity\ value\ in\ the\ current\ frame)^2}{(1 - pixel's\ intensity\ value\ in\ the\ current\ frame)^2}$$

By then dividing *top* by *base* one can find a ratio that indicates how much the stiffness of the material in question has changed compared to its stiffness in the first frame. This ratio is calculated for every pixel in every downsampled frame and is stored in a matrix.

$$Stiffness = \begin{bmatrix} Top_{frame\ 1} \div Base \\ \vdots \\ Top_{frame\ n} \div Base \end{bmatrix}$$

Next, a linear system is set up as shown below:

$$Stiffness = \begin{bmatrix} slope \\ y - intercept \end{bmatrix} * Strain\ data$$

Stiffness and strain data are both known values; slope and y-intercept are the unknown values. A graphical analogue to the current data setup would look like this:

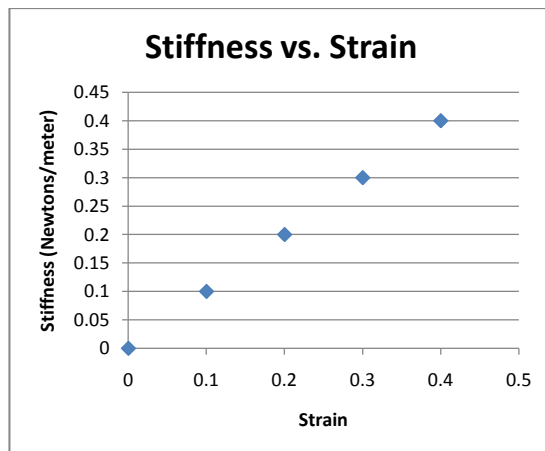


FIGURE 6. GRAPHICAL ANALOGUE TO THE CURRENT DATA SETUP. ONE CAN SEE THAT STIFFNESS IS THE DEPENDENT VARIABLE AND STRAIN IS THE INDEPENDENT VARIABLE. THE DATA POINTS ARE THE STIFFNESS VALUES AT GIVEN STRAIN. WHAT WE WISH TO OBTAIN IS THE SLOPE AND Y-INTERCEPT OF THIS DATA AND THIS CAN BE DONE VIA LINEAR ALGEBRA.

As with any linear system, simple algebra can be used to any unknowns. Using some basic algebra, one can find the values of the unknown values:

$$\frac{Stiffness}{Strain\ data} = \begin{bmatrix} slope \\ y - intercept \end{bmatrix}$$

This operation is carried out for every pixel in the downsampled image. Only the slope value is stored as the y-intercept value is not used in any further parts of the algorithm. We then end up with a matrix of the following form:

$$Distribution\ of\ slopes = \begin{bmatrix} Slope\ of\ pixel\ 1 & \dots \\ \vdots & \ddots \end{bmatrix}$$

#### MAPPING

The mapping algorithm takes the final output of the stiffness vs. strain algorithm which is a matrix containing the rate of change of each pixel of the downsampled matrices, color-codes the rates of stiffness change, and then overlays the color-coded image of the area of interest onto a frame of the ultrasound video. The output of this algorithm is one image which “points out” where the areas of greatest stiffness change are.

This process utilizes the following steps:

- The matrix from the stiffness vs. strain algorithm is converted to an indexed image format; its values are normalized so that they range from 0 to 255 (256 color format).
- A new color scheme is applied to the indexed image using MATLAB’s “colormap” command, in which different matrix values between 0 and 255 reference different *rgb* colors. We used MATLAB’s “hot” color scheme for the recoloring, in which progressively increasing matrix values reference progressively brighter shades of red and yellow, so that the highest values of the matrix look as if they are white hot (see Appendix 5). The color scheme, however, is easily changeable, and could easily be altered to suit a physician or researcher’s preference.
- The re-colored image is then converted to *rgb* format, and its size is increased to its original dimensions (i.e. to match the height and width of the area of interest). The conversion is done so that the color map is preserved when it is overlaid over the initial image.
- The area of interest in the first frame of the ultrasound video is replaced by its corresponding pixel in the resized, recolored image. This is done with a for loop which checks to see if a pixel in the resized, recolored image corresponds to a “1” in the binary mask, in which case it is copied into the original ultrasound frame.

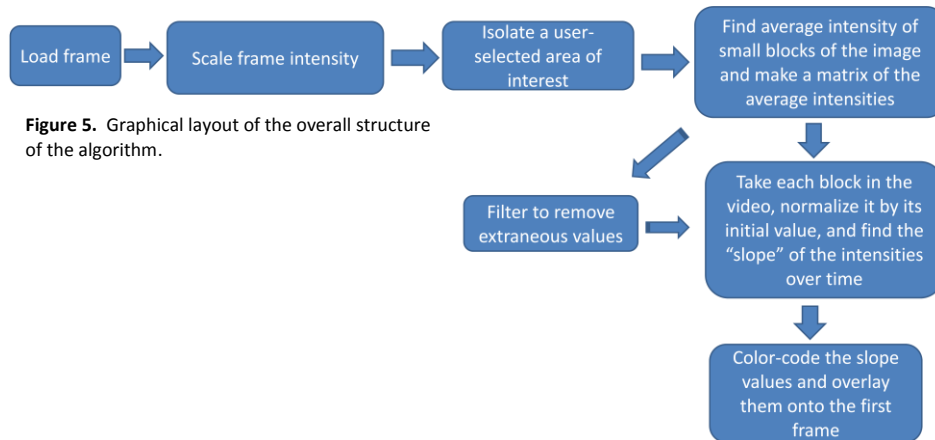


Figure 5. Graphical layout of the overall structure of the algorithm.



Figure 6. Ultrasound image at every step of the algorithm. From left to right: Original image, region of interest clipped out, stiffness vs. strain calculations carried out, and the stiffness vs. strain distribution overlaid on the original image. Ultrasound images provided by Dr. Kobayashi.

### ALTERATIONS TO THE ORIGINAL ALGORITHMS

We made a variety of alterations to the algorithms we received from our client, eliminating extraneous calculations where possible to allow the algorithm to run more efficiently. The original code that our clients were using is provided in Appendices 3 and 4, with our code provided in Appendix 5. The most significant changes we made are described below.

#### IMAGE NORMALIZATION

The original algorithm converted each ultrasound video frame from *rgb* to intensity images with MATLAB’s “*rgb2hsv*” command, which converts the three-dimensional *rgb* matrix into a three-dimension *hsv* matrix, in which each pixel has three numbers associated with it (hue, saturation, and value). The “value” matrix was then used as the intensity matrix. This method was inefficient because two-thirds of the calculated values (hue and saturation) were not used in the subsequent algorithms unnecessarily consuming memory. Thus, we replaced this command with the series of commands described earlier, which calculate intensity without using hue and saturation.

### OBJECT ISOLATION

Our client provided us with separate object isolation algorithm and stiffness vs. strain algorithms. The clipping algorithm output an image with the area of interest outlined, as well as the coordinates of the boundary pixels for the area of interest. Since the stiffness vs. strain algorithm was not designed to run with the clipping algorithm, it required the user to manually enter coordinates for a rectangular area of interest. The algorithms were combined making memory management more efficient and removing this step.

We simplified the object isolation algorithm, removing several unnecessary image manipulation commands, and changing its output to the binary mask. The binary mask was more useful than boundary coordinates because it facilitated the matrix operations which we conducted in subsequent subroutines.

### FILTRATION AND DOWNSAMPLING

The original stiffness algorithm utilized a series of *for* loops which divided the rectangular area of interest into a series of blocks, and took the average of each block for future calculations. We simplified this process by incorporating the convolution and downsampling algorithms, which use more matrix operations and fewer “*for*” loops. This achieved a similar result while reducing the execution time.

### STIFFNESS VS. STRAIN CALCULATION

Multiple changes were made to the stiffness vs. strain algorithm, and these changes greatly increased the speed. First of all, minor changes were made throughout the algorithm optimize memory assignments (i.e. new variables were assigned only when necessary, etc.). Originally, the algorithm read the ultrasound movie from the computer’s hard drive one frame at a time. Hard drive reading is slow and should be done as little as possible, so instead of having the algorithm perform as many hard drive reads as there are movie frames, a change was implemented so that the algorithm read the hard drive only once, reading the entire movie into RAM memory at one time.

Another important improvement was the removal of extraneous *for* loops and replacing them with MATLAB functions optimized the specific purpose that the loop was used for. This significantly reduced processing time (Appendix 5).

## RESULTS

### HARDWARE

The source code provided by Texas Instruments was combined to build an Ethernet interface program. Code Composer Studio was used to compile the program and convert it to machine code. The program can be loaded onto the chip using the USB interface drivers associated with Code Composer Studio. Currently, the program can perform the basic function of decoding a video file and transferring it to the chip frame by frame. There is also a user interface program that allows one to set a limited number of parameters such as the file name and codec.



A DSP chip better suited for the project was requested and received from Texas Instruments. The new chip is a TMS320C6455-1200 and has 1200 MHz processing power and 2 Mb on chip cache memory. Its development board has 256 Mb RAM and 8 MB flash memory. The new chip is intended specifically for image processing and is set up to return data to the computer.

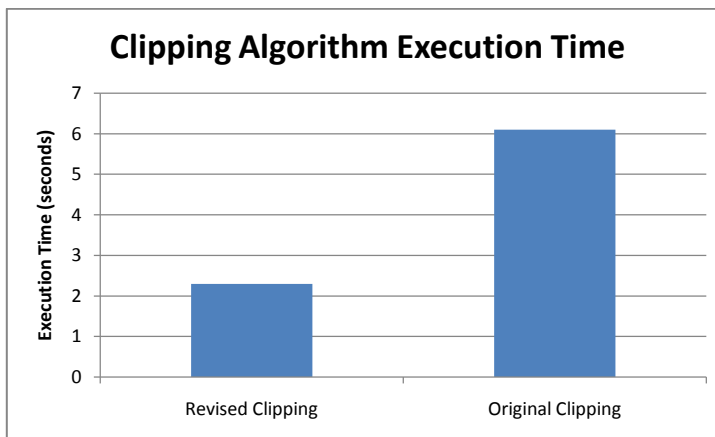
**MAPPING**

The mapping feature, overlaying the stiffness vs. strain distribution onto one of the frames was added to the algorithm. This allowed for better readability of results.

**PROCESSING**

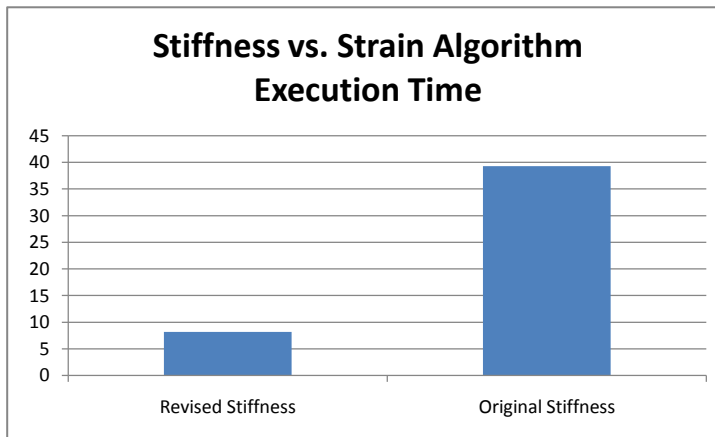
Testing was performed on both the original algorithms and on our finished algorithm. A data set with 434x532 pixels per frame and 5 was used. The implemented changes to the algorithm caused it to execute significantly faster and generate only marginal data deviations (an average of  $\pm 5.9\%$  intensity variation per pixel) from the original algorithms (the formulae used to calculate the percent intensity variation per pixel are included in Appendix 2).

The clipping part of our algorithm was tested against the original clipping algorithm separately from the stiffness vs. strain portion. Not only did our algorithm clip out the same region of interest as the original when given a specific seed point, but it also executed much faster (see Figure 7).



**Figure 7.** Run time of the clipping algorithm tested on a Dell Inspiron Desktop Computer (2.31 GHz Dual-Core AMD Processor). For a small data set, (434x532 pixels by 5 frames), the revised clipping algorithm executed significantly faster than the original while still returning results identical to that of the original.

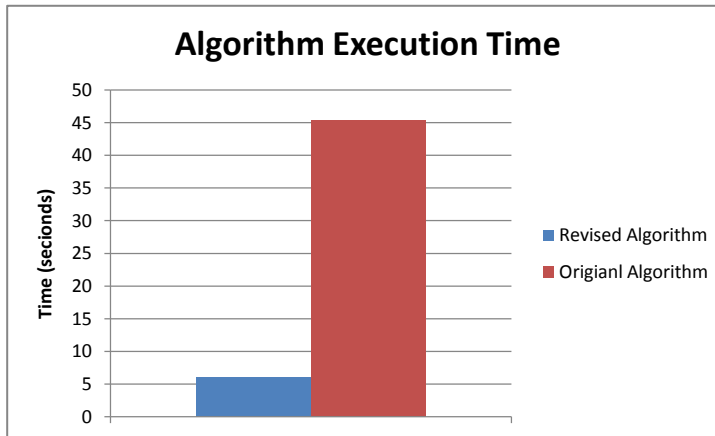
The stiffness vs. strain part of our algorithm was also tested against the original stiffness vs. strain algorithm separately from the clipping portion. Significant reduction in execution time was observed (see Figure 8) with an average data deviation of  $\pm 5.9\%$  intensity variation per pixel. Since this deviation is small, it is acceptable due to the time savings. The reason for the deviation is discussed in the future work.



**FIGURE 8.** Run-time for the revised stiffness vs. strain algorithm vs. original stiffness vs. strain algorithm. These times were recorded for a small data set (434x532 pixels by 5 frames) on a Dell Inspiron Desktop Computer (2.31 GHz Dual-Core AMD Processor).

We also compared the performance of the stiffness vs. strain portion of the two algorithms with a larger data set, 434x532 pixels by 30 frames. The original algorithm failed to process the data set, returning a matrix of zeros. The revised algorithm took 13.59 seconds yielding predicted results.

Combining these two algorithms allowed even further revision to be made so that the end result was an even faster algorithm that performed both clipping and the stiffness vs. strain analysis. This revision consisted mostly of making the memory usage more efficient. The execution time of this algorithm was compared against the original and, as one would predict, ours afforded excellent time savings with only minimal data deviation (see Figure 9).



**Figure 9.** Sum of run-times for the original two algorithms vs.the run time for the revised algorithm. These times were recorded for a small data set (434\*532 pixels by 5 frames) on a Dell Inspiron Desktop Computer (2.31 GHz Dual-Core AMD Processor).

## FUTURE WORK

The stiffness vs. strain algorithm is currently written in MATLAB. It will need to be converted to C so that the Code Composer Studio can further convert it to machine code so that it can be run on the DSP chip. This can be done by either rewriting the algorithm by hand or using a specialized program such as the Embedded IDE Link CC addition to Simulink. Additionally, changes should be made to the interface program to enable it to load multiple frames simultaneously to the chip. This would reduce the number of hard drive reads, reducing the execution time.

The user interface program will need to be expanded to allow processing parameters, such as the seed point for the clipping algorithms and filter levels, to be set. The stiffness vs. strain algorithm will need to be adjusted to process the dataset one image at a time and receive input parameters. The former also means that effective temporary storage of processed data will need to be set up.

Besides adjusting the algorithm to run on the chip and converting it to C, the algorithm must continue to be revised. A three dimensional array can be used to further improve the efficiency. This can be achieved by indexing all data by frame number, x-coordinate, and y-coordinate on the image. The current method uses 2-dimensional arrays (indexing all data by frame number and pixel number), which requires several *for* loops which would be unnecessary with 3-dimensional arrays. Additionally, the small deviation from the original algorithm in the output must be reduced. This is probably due to edge effects that are occurring with the convolution function as this function averages values that are inside the region of interest with values that are outside of the region of

interest. Some modification to the code must be made to correct for this. Finally, more robust filters should be incorporated into the algorithm to further reduce noise in the data.

## CONCLUSION

In conclusion, ultrasound data is currently underutilized; it currently provides information on tissue structure, but with additional processing, it can yield information on tissue material properties, specifically, the change in a tissue's stiffness as it is strained. This information will be valuable for physicians and researchers, as it can identify damaged locations in a tendon or other tissue. The final goal of this project is to implement a program on a DSP chip which will conduct this processing on a 60-frame 700×700-pixel ultrasound video in under 5 minutes. This semester, we acquired an algorithm from Dr. Vanderby and Dr. Kobayashi, which calculates a tissue's stiffness vs. strain, and reduced its processing time by 87% while introducing a 5.9% difference in the final pixel calculations. We also modified it so that it allows a user to select an area of interest, and overlays a color-coded image of the stiffness change distribution onto a frame of the ultrasound video. Additionally, we set up an interface between a computer and a DSP chip, which can be modified in future semesters to manage data flow between these two devices.

Future work will include reducing the percent difference between the algorithm output and the original algorithm's output, transcoding the algorithm from MATLAB into C, and expanding the interface to send multiple ultrasound video frames simultaneously to the chip. Also, the interface will need to be expanded to send parameters (such as the image isolation seed point), to the chip.

Thus, though the device is not ready for clinical applications, the algorithms have been significantly improved, and can now be more effectively used for research.

## APPENDICES

## APPENDIX 1—ORIGINAL CLIPPING ALGORITHM

The following code was provided by our client and is written in MATLAB. This code clips out an area of interest from a single picture. Note that all text colored green is a comment and is not executed by the computer.

```
function [boundaryPixel, finalImage, lFamily, time] = EdgeDetection(imIn, dob)
% This function is used to detect the edge of the possible tumor, ligament,
% or tendon. The user needs to specify the file name of the image, and the
% X and Y of the starting pixel.
% Written by: Mon-Ju Wu

% imIn: The file name of the image file.
% dob: This is used to tell if the target image is darker or brighter than
% its surrounding areas. If the target is brighter than the surrounding
% area, enter "1". Otherwise, enter "0".
% boundaryPixel: The matrix which list the [row, column] of all the
% boundary pixels.
% finalImage: The final image.
% lFamily: The total number of pixels in the target regions.

% Read in the image. Check if the images are in gray scale. If not, turn
% them into gray scale. Also, scale the image to 8-bit image.
% image: The read-in image (in 8-bit gray scale).
% iRow: The number of rows of the image.
% iColumn: The number of columns of the image.
% maxImage: The max intensity of the image.
% brightness: The variable used to tell if the target image is darker or
% brighter than its surrounding areas. It takes the value from "dob".
% RGB = imread(imIn); % Read in the image in RGB format
RGB=imIn; %%matrix input for testing
HSI = rgb2hsv(RGB); % Convert the image from RGB format to HSI format.
image = HSI(:,:,3); % Pass the intensity matrix of the image to "image"
[iRow, iColumn] = size(image);
maxImage = max(max(image));
image = image*double(1/maxImage); % Re-adjust the max intensity value to make
sure the intensity falls between 0 and 1.
brightness = dob;
% If the target image is darker than its surrounding area, take the
% inverse intensity value of the image so that the target image would
% look brighter than surrounding areas.
if brightness==0
    image = 1 - image;
end
imshow(image); % Plot the image.

% inOut: A matrix (with the same size as the image) used to keep in track
```

```

% if a pixel is processed. It is initiated as zero. If a pixel processed,
% its corresponding pixels in "inOut" becomes 1.
% outImage: A matrix (with the same size as the image) used to keep in track
% which pixels belong to the specified region. It is initiated as zero. If
% a pixel is considered as part of the region, its corresponding pixel in
% "outImage" becomes 1.
% eMask: A 3x3 matrix used to perform erosion.
% family: A matrix which used to keep in track the boundary pixels of the
% region. It is initiated with the boundary of the starting pixel
% (indicated by user) and its surrounding 7*7 matrix. It is in the format
% of [row, column].
% X: The column number of the starting pixel.
% Y: The row number of the starting pixel.
% bRow: The rows of the boundary pixels.
% bCol: The columns of the boundary pixels.
inOut = zeros(size(image));
outImage = zeros(size(image));
beginColumn=229;
beginRow = 129;           %get rid of ginput command for testing purposes % Use
the command "ginput" to let the user select the starting point with cursor.
inOut(beginRow-3:beginRow+3,beginColumn-3:beginColumn+3) = 1;           % Set
the 7*7 matrix centered at the starting point as processed.
outImage(beginRow-3:beginRow+3,beginColumn-3:beginColumn+3) = 1;           % Set
the 7*7 matrix centered at the starting point as part of the target region.
eMask = ones(3,3);
boundaryImage = outImage - imerode(outImage, eMask);           % Extract the
boundary of the target region.
[bRow bCol] = find(boundaryImage);           % Find the pixel location of the
boundary.
family = [bRow bCol];

% Start the loop to add pixels into "outImage".
% pSize: Previous number of pixels of the region in "outImage".
% cSize: Current number of pixels of the region in "outImage".
% pixelIndex: The list of the pixel number that's within the region.
% family: The
% fRow: The number of rows of the "family".
% fCol: The number of columns of the "family".
% fAverage: The previous average intensity of region. It is initialized as
% the average intensity of the first 7x7 matrix.
% average: The current average intensity of region.
iRun = 0;
pSize = 0;
cSize = 0;
while iRun<2
    if iRun==0
        pSize = 1;
        iRun = iRun + 1;
        average = sum(sum(image(beginRow-3:beginRow+3,beginColumn-
3:beginColumn+3)))/49;           % Calculate the average intensity of the target
region.
    end
    % Find the total number of pixels within the region.

```

```

    outImage = imfill(outImage, 'hole');           % Use the command "imfill"
to turn every pixel within the region into part of target region.
    pixelIndex = find(outImage);                 % Find every pixel location of the
target region.
    cSize = length(pixelIndex);                 % Calculate the totla number of
pixels (belong to the region).
    % Find the boundary pixels of the region.
    boundaryImage = outImage - imerode(outImage, eMask); % Calculate
the boundary.
    [bRow bCol] = find(boundaryImage);
    family = [bRow bCol];                       % Create the "family" matrix as which
record the locations of the boundary pixel.
    [fRow fCol] = size(family);
    if cSize==pSize                             % If the size of target region doesn't change
any more, that means it probably already
        break                                     % hits the pixel whose intensity is sharply less
the average intensity of the target
    else                                         % region. That means it probablyhtis the
boundary. In this case, the the loop stops.
        fAverage = average;
        % Calculate the average intensity of the family.
        average = 0;
        for z=1:cSize
            if (image(pixelIndex(z))-fAverage)>0.1
                average = average + fAverage;
            else
                average = average + image(pixelIndex(z));
            end
        end
        average = average/cSize;
        % Caculate the intensity of the neighboring pixel. If it falls
        % within the threshold, add it to the target region. Afterwards,
        % mark the pixel as processed (no matter if it's in the target
        % region or not).
        for i=1:fRow
            pSize = cSize;
            for p=family(i,1)-1:family(i,1)+1
                for q=family(i,2)-1:family(i,2)+1
                    if (p>1)&&(p<iRow)&&(q>1)&&(q<iColumn)
                        if inOut(p,q)==0
                            inOut(p,q) = 1;           % Mark the pixel as
processed.

                            sumNeighbor = 0;
                            tempMask = zeros(3,3);
                            % Calculate the intensity of the neighboring
                            % pixel.
                            for m=p-1:p+1
                                for n=q-1:q+1
                                    if image(m,n)-fAverage>0.02
                                        tempMask(m,n) = fAverage;
                                    else
                                        tempMask(m,n) = image(m,n);
                                    end
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end

```

```

sumNeighbor = sumNeighbor + sum(sum(tempMask));
if (fAverage-image(p,q)<0.13)&&(fAverage-
(sumNeighbor/9)<0.05)
    outImage(p,q) = 1;          % Mark the pixel
as processed.
end
end
end
end
end
end
end

% Create a 3x3 mask to perform erosion on "inOut" to extract the boundary
% of the image.
% eMask: A 3x3 matrix used to perform erosion.
% boundaryImage: The output image boundary.
% boundaryPixel: A n*2 matrix which lists all the boundary pixels.
% finalImage: The final image which is composed of the original image with
% the detected boundary.
outImage = imfill(outImage, 'hole');          % Use the command "imfill" to
turn every pixel within the region into part of target region.
boundaryImage = outImage - imerode(outImage, eMask);          % Extract the
boundary of the target region.
% If the image is inversed, convert it back.
if brightness==0
    image = 1 - image;
end
finalImage = image;
[tempRow tempCol] = find(boundaryImage);          % Find the pixel location of
the boundary.
boundaryPixel = [tempRow tempCol];          % Create the "boundaryPixel"
matrix as which record the locations of the boundary pixel.
[bpRow bpCol] = size(boundaryPixel);          % Get the size of the
"boundaryPixel".
for s=1:bpRow
    finalImage(boundaryPixel(s,1), boundaryPixel(s,2)) = 1;
end
imshow(finalImage)
lFamily = length(find(outImage))

```



## APPENDIX 2—ORIGINAL STIFFNESS VS. STRAIN ALGORITHM

The following code was provided by our client and is written in MATLAB. This code calculates the stiffness vs. strain distribution for a user-defined rectangular area in each frame of an ultrasound video. Note that all text in green is a comment and is not executable code. A few lines of code had to be modified to resolve various compiler errors; these lines are marked with the comment

`%modification necessary here for code to execute.`

```
% avi2d06d.m
% H. Kobayahsi
% 06_15_08
% This m file will import AVI format file (infile) into Matlab and freeze each
% frame and convert
% into color_echo map. Finally combine all images into one animation AVI
% file and output (outfile).
% Segment chosen block into multiple blocks in HORIZONTAL direction complare
% combination of avi_h01 and avi_slope01 to evalaute regional slope
% destribution
% automated version of avi2d01.m
% adopting avi2d05 format

infile=input(' INPUT_File name ?','s')

amp=input('amplification? ')

mov=aviread(infile);
[dummy,FrameNum]=size(mov);
loc=[-60,-60,0,0];
dmov=aviread(infile,25); % frame number that filter is set up off of has been
changed to be appropriate for testing purposes from frame 60 to frame 25
[X,Map] = frame2im(dmov) ;
HSI = rgb2hsv(X); %modification necessary here for code to execute
%H = HSI(:,:,1);
%S = HSI(:,:,2);
I = HSI(:,:,3); %modification necessary here for code to execute

I=double(I)*amp; %modification necessary here for code to execute

I_max=max(max(I));

FI=flipdim(I,1)/I_max;

[nraw,ncol]=size(FI);

contour(FI(1:nraw-50,:),30);
grid

FrameNum

% evaluation of mea echo in ROI
checker=input(' Define range of ROI YES(1) NO(0)');
```

```
xstart_ROI=input('x_START');
xend_ROI=input('x_END');
ystart_ROI=input('y_START');
yend_ROI=input('y_END');

sum=0.0;
pixel_count=0;
for ix=xstart_ROI:xend_ROI
    for jy=ystart_ROI:yend_ROI
        sum=sum+FI(ix,jy);
        pixel_count=pixel_count+1;
    end
end

MEAN_ROI=sum/pixel_count
Proc_range=input('% range of process magnitude range ')/100;
up_bound=MEAN_ROI*(1+Proc_range);
lw_bound=MEAN_ROI*(1-Proc_range);

checker=input(' Define range of viewing area YES(1) NO(0)');

if checker == 1

    xstart_total=input('x_START');
    xend_total=input('x_END');
    total_point_V=xend_total-xstart_total
    nlayer=input('Number layers in V-direction? ');
    ystart0=input('y_START');
    yend0=input('y_END');
    total_point_H=yend0-ystart0
    iblock=input(' Number of lblocks in H-drection = ');

    vblock=nlayer;
    v_width= round((xend_total- xstart_total)/nlayer);
end

start_frame=input('starting frame nummber ?')
end_frame=input('ending frame number ?')

xstart0=xstart_total;
xend0=xstart0+v_width;
for nl=1:nlayer

Layer_Number=nl

% Checking the condition of block number
checkb=ceil((yend0-ystart0)/iblock)-floor((yend0-ystart0)/iblock);
```

```
% corrected block number
nblock=checkb+iblock;
% evlaaution of block width
blockw=floor((yend0-ystart0)/iblock);
% block width of final blcok
if checkb == 1
flockw=yend0-(iblock*blockw+ystart0);
else
flockw=blockw;
end
%FrameNum=1;

for i=1:FrameNum
%Frame_number=i
dmov=aviread(infile,i);
[X,Map] = frame2im(dmov);
HSI = rgb2hsv(X);%modification necessary here for code to execute
%H = HSI(:,:,1);
%S = HSI(:,:,2);
I = HSI(:,:,3); %modification necessary here for code to execute

I=double(I)*amp; %modification necessary here for code to execute

I_max=max(max(I));

FI=flipdim(I,1)/I_max;

% block loop
ystart=ystart0;
yend=yend0;
for nb=1:nblock
%block_numner=nb
if nb <= nblock-1
width=blockw;
else
width=flockw;
end

ystart;
yend=ystart+width;

%checkpt=input(' ystart and yend looks OK? ');

if checker ==1
SI=FI(xstart0:xend0,ystart:yend);
else
SI=FI;
end
[nrow, ncol]=size(SI);

ncount=1;
sum=0.0;
```

```
for j=1:nrow
    for k=1:ncol
        temp=SI(j,k);

        % suppressing loop
        if temp > up_bound;
            temp=temp*0.01;
        end

        %if temp < lw_bound;
        %    temp=temp*0.01;
        %end

        % summation loop
        if temp > 0.05
            sum=sum+temp;
            ncount=ncount+1;
        end
    end
    end
    temp=sum/ncount;
    msi(i,nb)=temp;
    %contourf(SI);
    %check=input('checker ? ');
    %F=getframe(gcf);
    %aviobj = addframe(aviobj,F);
    ystart=width+ystart;
end

end

%aviobj = close(aviobj);

for nb=1:nblock

    base=((1+msi(1,nb))^2/((1-msi(1,nb)))^2);
    for i=1:FrameNum

        top=((1+msi(i,nb))^2/((1-msi(i,nb)))^2);
        nstiff(i,nb)=top/base;

    end
end

% contouring nstiff_frame
% contourf(nstiff,20)
%checker=input('Contour OK ?');
%plot(nstiff)
%checker=input(' Stiff_Frame plot OK ?');
% slope evalaution
%start_frame=1;
%end_frame=20;
```

```
for nb=1:nblock
    for i=start_frame:end_frame
        y_value=nstiff(i,nb);
        x_value=i*0.01;
        mtx(i,1)=x_value;
        mtx(i,2)=1;
        vec(i)=y_value;
    end
    MX=mtx'*mtx;
    VC=mtx'*vec';
    sol=inv(MX)*VC;

    slp(nl,nb)=sol(1);
    tg(nl,nb)=sol(2);
end

xstart0=xend0;
xend0=xstart0+v_width;
end
```

## APPENDIX 3—REVISED AND COMBINED ALGORITHM

This is the final copy of our algorithm that combines both the clipping and stiffness vs. strain algorithms. Significant revisions were also made to both algorithms to facilitate combining them and to allow them to execute faster.

```
%Main Ultrasound Stiffness Analysis File
%Version 1.4
%Last Edited by Jonathan Meyer
%Authors: Bogdan Dzyubak, Joe Helfenberger, Matthew Parlato, and Jonathan Meyer
% Based on source code from Dr. Hirhito Kobayashi and Mon-Ju Wu
%This code calls a variety of subroutines to determine the changes in
% stiffness in a tendon or ligament under a constant strain rate.
clear all; close all; clc;
%% Step 1: Load an avi and normalize it
infile=input('Enter the file path of the file to analyze: ','s');
mov=aviread(infile);
FrameNum=length(mov); % Get the number of frames in the avi
Images=cell(FrameNum,1);
for i=1:FrameNum
    if i==1
        X = frame2im(mov(i));
        indexedGrayImage = double(rgb2gray(X));
        intensityGrayImage=indexedGrayImage./255;
        I_max=max(max(intensityGrayImage));
        Images{i}=intensityGrayImage./I_max;
        finalImage=ind2rgb(indexedGrayImage, colormap(gray(255)));
    else
        X = frame2im(mov(i));
        indexedGrayImage = double(rgb2gray(X));
        intensityGrayImage=indexedGrayImage./255;
        I_max=max(max(intensityGrayImage));
        Images{i}=intensityGrayImage./I_max;
    end
end
imageSize=size(Images{1});

clear X I_max infile indexedGrayImage
%% Step 2: get area of interest
% and have the user select the area of interest.

Confirmation=0;
while Confirmation==0
    areaOfInterest=ObjectIsolation1(Images{1},1);
    Confirmation=input('Is the selected area acceptable? (Enter=yes, 0=no) ');
end

%% Step 3: Convolve and downsample images
hblock=input('kernel height: ');
wblock=input('kernel width: ');
kernel=ones(wblock,hblock);
%Cut out the area of interest
```

```

[AreaOfInterestRows,
AreaOfInterestColumns]=find(areaOfInterest);%AreaOfInterestRows is a list of
%row numbers in which there are non-zero values
maxRow=max(AreaOfInterestRows);
maxColumn=max(AreaOfInterestColumns);
minRow=min(AreaOfInterestRows);
minColumn=min(AreaOfInterestColumns);
areaOfInterestHeight=maxRow-minRow;
areaOfInterestWidth=maxColumn-minColumn;

downsampledImage=cell(FrameNum,1);
for i=1:FrameNum; % Crop the area of interest, and then filter and downsample it
    %Downsample the area of interest
    if i==1 %only the first downsampled image needs to be multiplied by the
        % areaOfInterest

downsampledImage{i}=downsample(downsample(conv2(Images{i}(minRow:maxRow,minColumn:
n:maxColumn),kernel,'same')./9.*areaOfInterest(minRow:maxRow,minColumn:maxColumn
),hblock)',wblock)');
%This command takes the block containing the areaOfInterest, convolves it,
isolates the areaOfInterest in it (to allow the pixels of the downsampled area
of interest to be found later) and downsamples it using the blocks specified by
the convolution kernel.
    else %The remaining frames don't need to be multiplied by the
        %areaOfInterest

downsampledImage{i}=downsample(downsample(conv2(Images{i}(minRow:maxRow,minColumn:
n:maxColumn),kernel,'same')./9,hblock)',wblock)');
%This command does the same thing without multiplying it by the area of
%interest
    end
end
[downsampledRowNumber,downsampledColumnNumber]=size(downsampledImage{1});
clear kernel hblock wblock

disp(num2str(FrameNum))
yes=input('Do you want to specify a start and end frame (1=yes, Enter=no)?');
if yes==1
    start_frame=input('starting frame number?');
    end_frame=input('ending frame number?');
else
    start_frame=1;
    end_frame=FrameNum;
end
summation=downsampledImage{25};
[up_bound,low_bound,size_of_summation_matrix]=Filtering(summation);
%% Actually Reading in the Frames and such
%% The following two lines allocate memory to the arrays msi and nstiff
msi=zeros(FrameNum,downsampledRowNumber*downsampledColumnNumber);
nstiff=msi;

for i=1:FrameNum;

    pixel_counter=1; %This is used in assigning a number to every pixel used %in
calculation. This will make later matrix operations easier.

```

```

    for j=1:downsampledRowNumber;
        for k=1:downsampledColumnNumber;
            msi(i,pixel_counter)=downsampledImage{i}(j,k); %msi is a matrix %of
average pixel values, with framenumbers as rows and block averages as %columns
            if msi(i,pixel_counter) > up_bound;
                msi(i,pixel_counter)=msi(i,pixel_counter)*0.01;
            end
            base=((1+msi(1,pixel_counter)))^2/((1-msi(1,pixel_counter)))^2;

            top=((1+msi(i,pixel_counter)))^2/((1-msi(i,pixel_counter)))^2;

            nstiff(i,pixel_counter)=top/base;

            pixel_counter=pixel_counter+1; %here we increment the pixel %count
        by 1
    end
end
end

%% Carrying out the Calculations for Stiffness vs. Strain
for p=1:pixel_counter-1;
    for i=start_frame:end_frame;
        y_value=nstiff(i,p);
        x_value=i*0.01;
        mtx(i,1)=x_value;
        mtx(i,2)=1;
        vec(i)=y_value;
    end
    MX=mtx'*mtx;
    VC=mtx'*vec';
    sol=inv(MX)*VC;

    slp(p)=sol(1);
end
p=0;
final_answer=zeros(downsampledRowNumber,downsampledColumnNumber);
for x=1:downsampledRowNumber
    for y=1:downsampledColumnNumber
        p=p+1;
        final_answer(x,y)=slp(p);
    end
end

%% create rgb image of manipulated data
% change everything back to rgb, with the stress vs. Strain image
% given a different colormap
clear imageSize
resizedImage=imresize(final_answer,[areaOfInterestHeight,areaOfInterestWidth]);
indexedImage=grayslice(resizedImage,255);%convert the intensity image to an
%indexed image
rgbMap=colormap(hot(255)); %Set a colormap that isn't grayscale
rgbImage=ind2rgb(indexedImage,rgbMap);
%% Add images

```



```

for x=1:areaOfInterestWidth
    for y=1:areaOfInterestHeight
        if areaOfInterest(y+minRow-1,x+minColumn-1)==1
            finalImage(y+minRow-1,x+minColumn-1,1)=rgbImage(y,x,1);
            finalImage(y+minRow-1,x+minColumn-1,2)=rgbImage(y,x,2);
            finalImage(y+minRow-1,x+minColumn-1,3)=rgbImage(y,x,3);
        end
    end
end
image(finalImage);

```

Two functions that we wrote are called from this main code. They are called filtering and and objectisolation1. Their code is included here as well for the sake of completeness.

### Filtering

```

function [up_bound, lw_bound, size_of_summation_matrix]=Filtering(summation)
total_sum=sum(sum(summation));
size_of_summation_matrix=size(summation);
pixel_count=size_of_summation_matrix(1).*size_of_summation_matrix(2);
MEAN_ROI=total_sum/pixel_count;
Proc_range=input('Specify Error Tolerance Percentage')/100;
up_bound=MEAN_ROI*(1+Proc_range);
lw_bound=MEAN_ROI*(1-Proc_range);
end

```

### ObjectIsolation1

```

function outImage = ObjectIsolation1(image, brightness)
% This function is used to detect the edge of the possible tumor, ligament,
% or tendon. The user needs to specify the file name of the image, and the
% X and Y of the starting pixel.
% Written by: Mon-Ju Wu
% Revised by: Jonathan Meyer

% image: The file name of the image file.
% brightness: This is used to tell if the target image is darker or brighter %
than
% its surrounding areas. If the target is brighter than the surrounding
% area, enter "1". Otherwise, enter "0".
% boundaryPixel: The matrix which list the [row, column] of all the
% boundary pixels.
% image: The read-in image.
% iRow: The number of rows of the image.
% iColumn: The number of columns of the image.

[iRow, iColumn] = size(image);

% If the target image is darker than its surrounding area, take the
% inverse intensity value of the image so that the target image would
% look brighter than surrounding areas.
if brightness==0
    image = 1 - image;
end

```

```

imshow(image);           % Plot the image.

% inOut: A matrix (with the same size as the image) used to keep track of %
% which pixels are processed. It is initiated as zero. If a pixel processed,
% its corresponding element in inOut becomes 1.
% outImage: A matrix (with the same size as the image) used to keep track of
% which pixels belong to the specified region. It is initiated as zero. If
% a pixel is considered as part of the region, its corresponding pixel in
% outImage becomes 1.
% eMask: A 3x3 matrix used to perform erosion.
% family: A matrix which used to keep track of the boundary pixels of the
% region. It is initiated with the boundary of the starting pixel
% (indicated by user) and its surrounding 7*7 matrix. It is in the format
% of [row, column].
% X: The column number of the starting pixel.
% Y: The row number of the starting pixel.
% bRow: The rows of the boundary pixels.
% bCol: The columns of the boundary pixels.
inOut = zeros(size(image));

[beginColumn, beginRow] = ginput(1); % Use the command "ginput" to let the
%user select the starting point with cursor.
inOut(beginRow-3:beginRow+3,beginColumn-3:beginColumn+3) = 1;           %
Set the 7*7 matrix centered at the starting point as processed.
outImage = inOut;               % Set the 7*7 matrix centered at the starting point
%as part of the target region.
eMask = ones(3,3);

% Start the loop to add pixels into "outImage".
% pSize: Previous number of pixels of the region in "outImage".
% cSize: Current number of pixels of the region in "outImage".
% pixelIndex: The indices of the pixels within the region.
% fRow: The number of rows of the "family".
% fCol: The number of columns of the "family".
% fAverage: The previous average intensity of the region. It is initialized %
% as the average intensity of the first 7x7 matrix.
% average: The current average intensity of region.
iRun = 0;
pSize = 0;
cSize = 0;
while iRun<2
    if iRun==0
        pSize = 1;
        iRun = iRun + 1;
        fAverage = sum(sum(image(beginRow-3:beginRow+3,beginColumn-
3:beginColumn+3)))/49;% Calculate the average intensity of the target region.
    end
    % Find the total number of pixels within the region.
    pixelIndex = find(outImage);% Find every pixel location of the target region.
    cSize = length(pixelIndex); % Calculate the total number of pixels
%that belong to the region.
    % Find the boundary pixels of the region.

```

```

boundaryImage = outImage - imerode(outImage, eMask); % Calculate the
%boundary.
[bRow bCol] = find(boundaryImage);
family = [bRow bCol]; % Create the "family" matrix which records the
%locations of the boundary pixel.
fRow=length(bRow);
if cSize==pSize % If the size of target region doesn't change anymore,
%that means it probably already hits the pixel whose intensity is sharply %less
the average intensity of the target region. That means it probably is %the
boundary. In this case, the loop stops.
    break
else

    % Calculate the average intensity of the family.
    average = 0;
    for z=1:cSize
        if (image(pixelIndex(z))-fAverage)>0.1
            average = average + fAverage;
        else
            average = average + image(pixelIndex(z));
        end
    end
    % Calculate the intensity of the neighboring pixel. If it falls
    % within the threshold, add it to the target region. Afterwards,
    % mark the pixel as processed (no matter if it's in the target
    % region or not).
    for i=1:fRow
        pSize = cSize;
        for p=family(i,1)-1:family(i,1)+1
            for q=family(i,2)-1:family(i,2)+1
                if (p>1)&&(p<iRow)&&(q>1)&&(q<iColumn)
                    if inOut(p,q)==0
                        inOut(p,q) = 1; % Mark the pixel as processed.
                        sumNeighbor = 0;
                        tempMask = zeros(3,3);
                        % Calculate the intensity of the neighboring
                        % pixel.
                        for m=p-1:p+1
                            for n=q-1:q+1
                                if image(m,n)-fAverage>0.02
                                    tempMask(m,n) = fAverage;
                                else
                                    tempMask(m,n) = image(m,n);
                                end
                            end
                        end
                        sumNeighbor = sumNeighbor + sum(sum(tempMask));
                        if (fAverage-image(p,q)<0.13)&&(fAverage-
(sumNeighbor/9)<0.05)
                            outImage(p,q) = 1; % Mark the pixel as
%processed.
                        end
                    end
                end
            end
        end
    end
end
end
end

```

```
        end
    end
end

% Create a 3x3 mask to perform erosion on inOut to extract the boundary
% of the image.
% eMask: A 3x3 matrix used to perform erosion.
% boundaryImage: The output image boundary.
% boundaryPixel: A n*2 matrix which lists all the boundary pixels.
boundaryImage = outImage - imerode(outImage, eMask);% Extract the boundary of
%the target region.
% If the image is inversed, convert it back.
if brightness==0
    image = 1 - image;
end
[tempRow tempCol] = find(boundaryImage); % Find the pixel location of the
%boundary.
boundaryPixel = [tempRow tempCol];% Create the boundaryPixel matrix as %which
record the locations of the boundary pixel.
[bpRow bpCol] = size(boundaryPixel);% Get the size of the boundaryPixel.
for s=1:bpRow
    image(boundaryPixel(s,1), boundaryPixel(s,2)) = 1;
end

imshow(image);
end
```

APPENDIX 4—CALCULATIONS AND EQUATIONS

**Stiffness vs. Strain Calculation**

The method used to find the slope of the stiffness vs. strain curve simply solves a linear system that is set up like this for a slope and y-intercept value:

$$Stiffness_{pixel} = \begin{bmatrix} slope \\ y - intercept \end{bmatrix} * strain\ data$$

$Stiffness_{pixel}$ , an  $n \times 1$  matrix, is the stiffness data for the pixel or block in question for every frame. As strain is assumed to be constant throughout the video, the strain at any point can be thought of as a proportional to the frame in question. Since an ultrasound machine operates at a set frame rate, the passage of time can be found by the number of frames that have been viewed. Using this logic, the following relation becomes clear:

$$\Delta strain \approx Constant\ rate = C$$

$$strain = C * \Delta time; \Delta time = time\ ellapsed$$

$$**Essentially:** strain \sim \Delta time$$

$$\Delta time = current\ frame\ number * frame\ rate$$

$$**Therefore:** strain \sim current\ frame\ number * frame\ rate$$

This relation indicates that strain is directly proportional to the passage of time, and, alternatively, the number of the frame in question multiplied by the frame rate is indicative of the amount strain that has occurred. Therefore, the client’s algorithm and our algorithm used the following formula to calculate the strain data for a video that is “n” frames long. The value of 0.01 is the frame rate of the client’s ultrasound machine. The right-hand column containing all ones is included so that when the matrix containing slope and y-intercept is multiplied by this matrix, y-intercept is added to the final value.

$$Strain\ data = \begin{bmatrix} (frame\ 1) * 0.01 & 1 \\ (frame\ 2) * 0.01 & 1 \\ \vdots & \vdots \\ (frame\ n) * 0.01 & 1 \end{bmatrix}$$

As this is a linear system, one can see that simple algebra can be used to find any unknowns. A system, for an ultrasound video that is “n” frames long, is set up like this:

$$Stiffness_{pixel} = \begin{bmatrix} stiffness\ value\ in\ frame\ 1 \\ \vdots \\ stiffness\ in\ frame\ n \end{bmatrix} = \begin{bmatrix} slope \\ y - intercept \end{bmatrix} * \begin{bmatrix} (frame\ 1) * 0.01 & 1 \\ (frame\ 2) * 0.01 & 1 \\ \vdots & \vdots \\ (frame\ n) * 0.01 & 1 \end{bmatrix}$$

The rules of fundamental algebra still apply, so to solve for the unknowns one must rearrange the equation to get the slope and y-intercept variables by themselves on one side of the equation. This can be done like this:

$$\begin{bmatrix} \text{stiffness value in frame 1} \\ \vdots \\ \text{stiffness in frame n} \end{bmatrix} * \begin{bmatrix} (\text{frame 1}) * 0.01 & 1 \\ (\text{frame 2}) * 0.01 & 1 \\ \vdots & \vdots \\ (\text{frame n}) * 0.01 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \text{slope} \\ \text{y-intercept} \end{bmatrix}$$

A problem occurs here, however, since the matrix dimensions of the stiffness matrix (an  $n \times 1$  matrix), the strain data matrix (an  $n \times 2$  matrix), and the slope and y-intercept matrix (a  $2 \times 1$  matrix) do not agree. Therefore, a common matrix technique must be applied to get the dimensions to agree so that this operation can be carried out. This technique is carried out as follows:

$$\begin{bmatrix} (\text{frame 1}) * 0.01 & 1 \\ (\text{frame 2}) * 0.01 & 1 \\ \vdots & \vdots \\ (\text{frame n}) * 0.01 & 1 \end{bmatrix}^T = \begin{bmatrix} 1 & \dots & 1 \\ (\text{frame 1}) * 0.01 & \dots & (\text{frame n}) * 0.01 \end{bmatrix}$$

$$\begin{bmatrix} (\text{frame 1}) * 0.01 & 1 \\ (\text{frame 2}) * 0.01 & 1 \\ \vdots & \vdots \\ (\text{frame n}) * 0.01 & 1 \end{bmatrix} * \begin{bmatrix} (\text{frame 1}) * 0.01 & 1 \\ (\text{frame 2}) * 0.01 & 1 \\ \vdots & \vdots \\ (\text{frame n}) * 0.01 & 1 \end{bmatrix}^T = \text{a matrix of dimensions } 2 \times 2$$

So now one can see that a matrix of finite size will be obtained. If this transform is applied to side of the equation, then it must be applied to the other side, as well (fundamental rules of algebra).

$$\begin{bmatrix} (\text{frame 1}) * 0.01 & 1 \\ (\text{frame 2}) * 0.01 & 1 \\ \vdots & \vdots \\ (\text{frame n}) * 0.01 & 1 \end{bmatrix}^T * \begin{bmatrix} \text{stiffness value in frame 1} \\ \vdots \\ \text{stiffness in frame n} \end{bmatrix} = \text{a matrix of dimension } 2 \times 1$$

Once again, a matrix of finite dimensions results from this operation. Also, one can now see that the matrix dimension problems are now no longer an issue:

$$(2 \times 2)^{-1} * (2 \times 1) = (2 \times 1)$$

After applying this transform and rearranging the equation so that the unknowns slope and y-intercept can be found, one obtains this formula:

$$\begin{bmatrix} (\text{frame 1}) * 0.01 & 1 \\ (\text{frame 2}) * 0.01 & 1 \\ \vdots & \vdots \\ (\text{frame n}) * 0.01 & 1 \end{bmatrix}^T * \begin{bmatrix} \text{stiffness value in frame 1} \\ \vdots \\ \text{stiffness in frame n} \end{bmatrix} = A$$

$$\begin{bmatrix} (\text{frame 1}) * 0.01 & 1 \\ (\text{frame 2}) * 0.01 & 1 \\ \vdots & \vdots \\ (\text{frame n}) * 0.01 & 1 \end{bmatrix} * \begin{bmatrix} (\text{frame 1}) * 0.01 & 1 \\ (\text{frame 2}) * 0.01 & 1 \\ \vdots & \vdots \\ (\text{frame n}) * 0.01 & 1 \end{bmatrix}^{-1} = B$$

$$A * B = \begin{bmatrix} slope \\ y - intercept \end{bmatrix}$$

This formula is used on every pixel or block in the original algorithm and in our algorithm.

**Data Deviation Calculation**

To determine if the data sets produced by the original algorithm and our algorithm were the same, some type of comparison had to be performed. We found the average percent difference of the pixels of our results vs. the original results by the following formula:

$$average\ percent\ difference = \frac{\sum_{i=1}^{n=\# \text{ of pixels}} \frac{|p_i - l_i|}{\frac{p_i + l_i}{2}}}{n}$$

Where p=our results and l=original results

## APPENDIX 5—PRODUCT DESIGN SPECIFICATIONS

**Date:** December 12, 2008

**Clients:** Dr. Ray Vanderby, Dr. Hirohito Kobayashi, and Texas Instruments

**Advisors:** Dr. Tom Yen and Dr. Walter Block

**Function:** Texas Instruments (TI) is interested in finding applications in the medical field for their Digital Signal Processing Chips (DSP's). DSP's specialize in simple mathematical calculations and are able to collect information and process it in real-time (unlike other common micro-processors). The guidelines for this project are that a TI DSP chip be used and that the final product solves a meaningful medical problem. We have chosen to apply the DSP chip in the field of medical imaging, where it would function as a part of a portable ultrasound medical imaging system that would identify and analyze stiffness distribution in tissues on-the-fly. Though significant work can be done on such a project, we have decided on three goals that we would like our device to satisfy:

1. Implement an algorithm that processes stiffness vs. strain in tissues using stored data
2. Optimize this algorithm for near-real time processing
3. Execute this algorithm from a DSP chip

In the future, this device could be used for surgical or clinical diagnostic purposes. However, currently our client just requires a device that can quickly process this data for research purposes.

### **Client Requirements:**

➤ **Device that processes data quickly:** The performance that the client requires from this device is that it must be able to process  $434 \times 532$  pixels in 30 frames of ultrasound video data in less than 5 minutes. As this is a research device, immediate feedback after data gathering is not necessary, but it is helpful. Also, as this device could one day be used in a clinical setting, this standard of 30 frames in less than 5 minutes should be upheld.

A TI C6000 High Performance DSP Chip model \*\*\*\*\* will be used. As this chip will take some time to arrive, for this semester, Dr. Yen has made available a TMS320DM6437 DaVinci DSP chip. Though not ideal for the type of processing we wish to do, it should serve as an excellent template to get started from.

### ➤ **Interface**

The interface should be relatively easy to use, as any person in the medical field (someone not necessarily well versed in DSP technology) should be able to effectively use it. Currently, the interface is very straightforward as user input is limited to selecting the area of interest, specifying whether it is darker or lighter than the surrounding tissues, and choosing the size of the kernels that the convolution algorithm uses.



➤ **Compatability**

The device must be compatible with the client's computer as it will be interfacing with it for data storage and display purposes. Also, as this device may one day be used in a clinical setting, it must be compatible with most common computer platforms as it is likely that a computer will still act as this chip's display and storage device.

**Design Requirements:** As this device will primarily be used by the client to process ultrasound data being used in research, the device design has only a few constraints. It must be able to effectively identify tissues and analyze their properties in a matter of minutes, and it must have an effective interface that an expert in this field can easily use. The device must be able to interface with a computer as the ultrasound data will be stored and displayed on the computer. The size of the device should be convenient and weigh less than 10 lb. In the future, it is possible that the device may be used for clinical diagnosis or as a surgical tool, so the detector end should not be damaged by sterilization.

**Physical and Operational Characteristics**

**a. Performance requirements:** The ultimate goal of this device is to process data in a matter of minutes and send the processed data to an output and storage device (most likely a computer). It should be able to consistently identify tissues and analyze their distribution of stiffness change for a given strain rate. The device will initially be used for research purposes, perhaps once a day. The system needs to be portable - including a weight of less than 10 lb, a size smaller than 1 cubic foot, and should run, or have the capability to run, on battery power. Because the device may eventually be used in surgery or clinical diagnosis, the probe must not be damaged by sterilization.

This device must also perform its duties very quickly (on the order of processing 30 frames of 434 x 532 pixels in under 5 minutes). Precautions also need to be taken so that the chip does not run out of memory, as its limited RAM may fill up quickly during the course of the data processing. Another requirement is that it must be compatible with the client's computer (or any common computer platform, for that matter) as a computer will act as the primary data storage and display device. It is likely that the user would also use the computer to pass any necessary user inputs to the chip.

**b. Safety:** Ultrasound imaging has no known negative effects. If the device is used in surgical settings, the probe must be sterilized between uses.

**c. Accuracy and Reliability:** The device must be able to calculate stiffness vs. strain distributions in ultrasound video images accurately. It should be able to provide information about loading and damage of tissues. The techniques for the latter purpose are in the research stage, so final accuracy of the device is difficult to predict at this time.

This device also needs to be able to handle large data sets without running into data storage issues. Another requirement is that the device must be able to interface smoothly with a computer as it is likely that this device will not only be storing and displaying the ultrasound data on a computer, but it will be interfacing with a computer to receive necessary user inputs.

**d. Life in Service:** Due to the nature of its use, the device does not experience much stress. Its lifetime, thus, should be fairly extensive, on the order of several years or more. It will most likely become obsolete before it wears out.

**e. Operating Environment:** The device contains circuitry and plastics. It should not be exposed to very high or low temperatures, high humidity, strong magnetic fields, corrosive materials, physical stresses, or electric shock. If handled with care, the device should fair very well in both clinical and research settings. Care must be taken with it, however, if sterilized for use in a clinical setting, however, to ensure that the sterilization process does not destroy its sensitive electronics.

**f. Materials:** A DSP chip from Texas instruments, the appropriate ultrasound data, a desktop computer, and a device for the DSP Chip to display and store the edited data on.

- Dr. Yen has provided a TMS320DM6437 DaVinci DSP chip for us to work with for this semester. Though this is not ideal for our purpose, this chip should allow us to gain an adequate understanding of how to work with DSP chips.
- We have requested a \*\*\*\*\* High Performance DSP chip, which is better suited to our project.
- Sample ultrasound data has been provided by our clients Dr. Kobayashi and Dr. Vanderby.
- The output device that is currently being used is a standard TV. The TV can easily display the device's output, but it can not store any outputted information.
- Dr. Yen provided us with a desktop computer located in the Bioinstrumentation Lab in the Engineering Centers Building.

#### Production Characteristics

**a. Quantity:** Only one is currently needed for our clients. In the future, if the device is useful for research, clinical, or surgical applications, the quantity that will need to be produced is quite large.

**b. Target Product Cost:** The cost of the final device would be cost of the DSP plus accessories such as RAM and a circuit board. Therefore, the target cost will be below \$3,000.

#### Miscellaneous Requirements

##### A. FDA Approval:

The device will not need full FDA approval while it is being clinically tested. It will, however, be subject to the Clinical Trials & Investigational Device Exemption, which requires the following (Quoted from FDA, 2003):

- An IDE approved by an institutional review board (IRB). If the study involves a significant risk device, the IDE must also be approved by FDA.
- Informed consent from all patients,
- Labeling for investigational use only,
- Monitoring of the study, and

- Required records and reports.

**B. Customer:** We have two clients, whose expectations are as follows:

- Texas Instruments has agreed to donate “all resources necessary to develop new applications including hardware and software development tools,” including a DSP developer’s kit, supporting hardware, and consultation with their engineers. In exchange, they expect that their resources will be used to solve “a meaningful medical instrumentation or medical imaging design problem” (BME, 2008)
- Dr. Vanderby and Dr. Kobayahsi are overseeing this ultrasound project. They expect a device that can work in conjunction with a computer to quickly process stored ultrasound data to be developed by the end of the semester. In the future, they expect this device to be able to process 60 frames that are 700 x 700 pixels in size in under 5 minutes and to interface with a computer for data storage and display purposes.

**C. Research and Patient-related Concerns:**

**Sterility** The device will need to be sterilized between uses if used for anything beyond medical research. Due to the sensitivity of the device’s electronics to heat, it will probably be sterilized by chemical means.

**Privacy** Clinical testing of the device will involve storage of patient data. This data will need to be stored securely, to protect the privacy of research participants.

**D. Competition** To our knowledge and to the knowledge of our client, no device which measures and displays the stiffness vs. strain distribution in a tissues in real-time exists.

## REFERENCES

- Brown, Darla. 2003. "New Imaging Technology Deciphers Tumors." Distinctions. The University of Texas Health Science Center at Houston. Accessed December 2008 at <http://distinctions.uthouston.edu/archive/2003/oct/tumor.html>
- Crisco, J J., et al. 1998. "Stress Wave Velocities in Bovine Patellar Tendon." J Biomed Engr. **120**: 321-26.
- Foucher, J., Chanteloup, E., Vergniol, J., Castéra, L., Le Bail, B., Adhoute, X., Bertet, J., Couzigou, P., de Lédighen, V. 2006. "Diagnosis of cirrhosis by transient elastography (FibroScan): a prospective study." Gut. **55(3)**: 403-8
- Garra, Cespedes, Ophir, Spratt, Zuurbier, Magnant and Pennanen. 1997. "Elastography of breast lesions: initial clinical results." Radiology. **202**:79-86.
- Kobayashi, H. Vanderby, R. 2005. "New Strain Energy function for Acoustoelasticity Analysis of Dilatational Waves in Nearly Incompressible Hyper-Elastic Materials," J. Applied Mech., **72**: 843-851
- Kobayashi, H., Vanderby, R., 2007. "Acoustoelastic analysis of reflected waves in nearly incompressible, hyperelastic materials: Forward and inverse problems," J. Acoust. Soc. Am. **121 (2)**: 879-887
- Pourcelot, P., et al. 2005. "A non-invasive method of tendon force measurement." J of Biomech **38(10)**: 2124-29.
- Ophir, Jonathan, MD. 1998. "New Medical Imaging Technique Improves Chances of Early Cancer Detection." Algor. Accessed December 2008 at [http://www.algor.com/news\\_pub/cust\\_app/cancer/cancer.asp](http://www.algor.com/news_pub/cust_app/cancer/cancer.asp)
- Pislaru, C., Bruce, C., Anagnostopoulos, P., Allen, J., Seward, J., Pellikka, P., Ritman, E., Greenleaf, J. 2004. "Ultrasound Strain Imaging of Altered Myocardial Stiffness." Circulation. **109**:2905-2910
- Shmulsky, R., Seale, P., Snow, E. Analysis of acoustic velocity as a predictor of stiffness and strength in 5-inch-diameter pine dowels. Forest Products Journal. 56, 9:53-55.
- Smith, Steven W. The Scientists and Engineer's Guide to Digital Signal Processing. Chapter 28. 2007. California Technical Publishing.
- "Reflection and Transmission Coefficients (Pressure)" NDT Resource Center. Accessed December 11, 2008, at <http://www.ndt->

[ed.org/EducationResources/CommunityCollege/Ultrasonics/Physics/reflectiontransmission.htm](http://ed.org/EducationResources/CommunityCollege/Ultrasonics/Physics/reflectiontransmission.htm)

Vanderby and Kobayashi. 2008. Tissue Stiffness Evaluation with a 2-D Ultrasound System Using the Acoustoelastic Strain Gauge Method.

Vanderby, R. Personal interview: November 2008.