Using Doppler Ultrasound for Diagnosing and Staging of Rhinosinusitis

Implementation on a Clinical Scanner

Daniel Andersson & Peter Zandén

September 3, 2012

LUNDS UNIVERSITET
Lunds Tekniska Högskola

Master's Thesis
Faculty of Engineering, LTH
Department of Measurement Technology and
Industrial Electrical Engineering
Division of Electrical Measurements
Supervisor: Tomas Jansson

Abstract

Rhinosinusitis is the fifth most common diagnosis for which antibiotics are prescribed- but it is often done in vain. Currently, the only way to fully determine whether or not such treatment is actually needed is an invasive procedure. In order to spare patients from the discomfort that it would involve, this essential step is often skipped and instead, antibiotics are prescribed regardless. Using Doppler ultrasound, it is possible to induce and measure acoustic streaming inside a cavity. This can be used in order to determine the nature of the sinus infection and hence avoid unnecessary prescriptions of antibiotics. In this master's thesis, a program was created which simplifies and, to a degree, automates this new method of diagnosis. Using energy levels below the safety limits, set by the FDA, trial results were successful in inducing acoustic streaming velocities of 2 cm/s inside a phantom model. The method is ready to be used clinically using a pulse repetition frequency of 3.3 kHz.

Acknowledgements

We would like to give special thanks to Tomas Jansson for his help and guidance. To Hans W. Persson for his valuable comments and to Stephane at Ultrasonix for his help with the Ulterius SDK.

Contents

1 Introduction **6**

2 Theory **7**
2.1 Ultrasound 7
2.1.1 Acoustic Impedance 8
2.1.2 Pulses 9
2.1.3 Non-Linear Propagation 9
2.1.4 Acoustic Streaming 11
2.1.5 Transducer 11
2.1.6 Beam Forming 12
2.1.7 B-Mode Images 13
2.1.8 Doppler 14
2.2 Safety 15
2.3 Paranasal Sinuses and Rhinosinusitis 16
2.4 Sonix RP 18
2.5 Programming 20
2.5.1 C++ vs Java 20
2.5.2 Ulterius 20
2.5.3 Qt 21
2.5.4 Mutex 23

3 Method **25**
3.1 Approach 25
3.2 Sonix RP Machine 25
3.3 Modeling the Paranasal Sinuses 27
3.4 Programming 27
3.5 Safety 28

4 Results **32**
4.1 Sonix RP 32
4.2 Programming 32
4.2.1 Coding process 32
4.2.2 Graphical User Interface 33
4.2.3 Class Layout 35
4.3 Safety 37
4.3.1 Scale Test 37
4.3.2 Hydrophone Test 37
4.4 Transducer Glitch 42

4.5 Trial Results 42
4.6 True Frequencies 44

5 Discussion **45**
5.1 Sonix RP 45
5.2 Transducer Glitch 46
5.3 Programming 46
5.4 Safety 48
5.5 Milk Trials 49

6 Conclusion **50**

A Snotalyzer User Guide **53**

B Source Code **58**
B.1 main.cpp 58
B.2 mainwindow.h 58
B.3 mainwindow.cpp 59
B.4 mainwindow.ui 68
B.5 ushandler.h 75
B.6 ushandler.cpp 77
B.7 patientdialog.h 92
B.8 patientdialog.cpp 93
B.9 settingsdialog.ui 94
B.10 settingsdialog.h 98
B.11 settingsdialog.cpp 98
B.12 settingsdialog.ui 100
B.13 keypresswidget.h 104
B.14 keypresswidget.cpp 105
B.15 worker.h 105
B.16 worker.cpp 106

1 Introduction

Today, many patients diagnosed with rhinosinusitis are being administered antibiotics in vain. Previous studies have shown that the need for such treatments is predicated on the nature of the infection. Rhinosinusitis is caused by either bacteria or virus, and one way to distinguish between the two is to examine the viscosity of the mucus, or fluid, which is formed inside the maxillary paranasal sinuses during infection. Since antibiotics are only truly needed if the infection is caused by bacteria, determining the viscosity of the fluid could prevent extraneous treatments.

It has been found that acoustic streaming can be induced into a fluid through the use of Doppler ultrasound. Depending on the viscosity of the fluid, more or less energy is needed in order to generate a flow that is detectable by a Doppler measurement. If the difference in streaming velocity between the two types of fluids is large enough, it should be possible to determine the nature of the infection using Doppler ultrasound. The theoretical background for this master's thesis has been established, but an implementation for use in actual diagnosis has yet to be made.

The main goal of this master's thesis is to simplify and, to a degree, automate the diagnosis of paranasal sinus infections. Ultrasound is already being used in order to detect the presence of fluid inside the paranasal sinuses, but in order to, for certain, determine whether or not antibiotics are needed- an invasive procedure is still required. If it is possible determine the nature of the fluid inside the cavities using a non-invasive procedure, it would lessen the strain on the patients as well as lower the unnecessary use of antibiotics. In order to evaluate Doppler ultrasound as a diagnosis method, as developed by Tomas Jansson and Pernilla Sahlstrand Johnson, and decide whether or not it can be used in diagnosis, a physical implementation needs to be created. The aim of this master's thesis is to develop a program which can be used on a clinical ultrasound scanner and evaluate its validity in the diagnosis of paranasal sinus infections.

2 Theory

2.1 Ultrasound

Sound waves are longitudinal waves that propagate by oscillating the particles of the medium back and forth in the direction of the wave. By setting a surface in movement, the surface collides with particles of the medium at the face of the surface. These particles then do the same thing to whatever lies beyond them, effectively starting a chain reaction making the particles of the medium oscillate in the direction of the longitudinal wave. Where particles in adjacent regions have moved towards each other, there is a region of higher pressure. Vice versa in regions where particles have moved away from each other there is a region of lower pressure.[1] Normally, the net movement of the medium is zero but when something called acoustic streaming occurs, this is not the case. This will be explained later. The two most common ways of displaying a sound wave are with a function in a Time-Amplitude diagram, *i.e.* a sine wave or pulse, or with a frequency-amplitude diagram *i.e.* the frequency spectrum.

Sound is generally split into three categories depending on their frequencies. Infrasound, which refers to sound with frequencies lower than 20 Hz, Audible sounds ranging from 20 Hz to 20 kHz and Ultrasounds which have frequencies above 20 kHz. The relation between frequency (f) and the wavelength (λ) is determined by equation 1, where c is the speed of sound in the medium. It should be noted that the speed of sound in a medium is also dependant on temperature. At 37°C the speed of sound in soft tissue is usually approximated to 1540 m/s, and 3360 m/s in skull can bone. In relation, the speed of sound in water at 20°C is about 1480 m/s [1]. The frequencies used in this master's thesis range from 4-10 MHz, which corresponds to wavelengths of 0.385-0.154 mm in water.

$$f = \frac{c}{\lambda} \tag{1}$$

When a sound wave encounters an irregularity in the medium, or the surface of a new medium all together, part of the wave is reflected back towards its source and a part of the wave is scattered. Some of the energy is absorbed as heat, while the rest propagates further into the medium. The amount of energy which is reflected is determined by equation 2.

$$\frac{p_r}{p_i} = \frac{z_2 - z_1}{z_2 + z_1} \tag{2}$$

Where p_i and p_r are the amplitudes of the incident and reflected waves respectively, while z_1 and z_2 are the mediums' acoustic impedance.

2.1.1 Acoustic Impedance

Acoustic impedance is a measure of how the particles of a medium respond to a sound wave of a given pressure. The formula for calculating the acoustic impedance is found in equation 3.

$$z = \frac{p}{v} \tag{3}$$

Where p is the local pressure and v is the particle velocity. z can also be determined by the mediums density(ρ) and stiffness (k) as: $z = \sqrt{\rho k}$. To give a perspective of different acoustic impedances, see Table 1.

Table 1: Acoustic Impedances [1]

Material	z($kgm^{-2}s^{-1}$)
Liver	$1.66{*}10^6$
Fat	$1.33 {*}10^6$
Air	430
Bone	$6.47 {*}10^6$

As bone and soft tissue have very different properties, most of the energy is reflected back when the wave encounters bone inside tissue. However, if the thickness of a membrane/slab/structure corresponds to a half-wavelength, this reflection has been found to be minimized. This is due to what is called the 'intensity transmission coefficient' (α_T). At normal incidence of a membrane, if the impedance of the membrane is z in a surrounding medium of impedance Z, the transmission coefficient is determined by equation 4.

$$\alpha_T(f) = \frac{4Z^2}{4Z^2 cos^2(2\pi fl/c) + (Z + Z^2/z)^2 sin^2(2\pi fl/c)} \tag{4}$$

Where c is the speed of sound in the membrane, f the frequency, and l is the thickness of the membrane. So if $\lambda/2 = l$ or equivalently $f_0 = c/2l$, the transmission should be maximized [13].

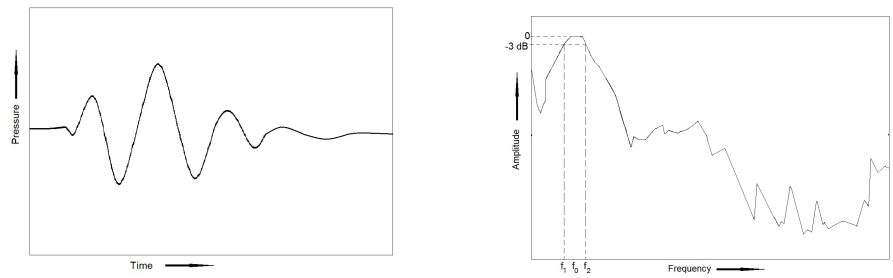


Figure 1: A pulse (left) and its frequency spectrum. The pulse contains a range of different frequencies, dominated by a center frequency, or nominal frequency f_0. [1]

An ultrasound image is built up from the echoes of larger interfaces and scattering from smaller targets. In order to get more distinct echoes, the ultrasound signal is sent in short bursts, or pulses. The longer the pulse, the stronger signal, but the shorter the pulse- the better distance resolution.

2.1.2 Pulses

Pulses can be represented as a wave in a pressure-time figure, or as amplitudes in a frequency spectrum. This is because pulses are comprised of many different frequencies, compared to continuous transmission which merely contain one single frequency. The frequencies included in a pulse are centered around the so called nominal frequency (sometimes referred to as the fundamental frequency). The nominal frequency is usually the one with the highest amplitude, and is hence the one mentioned as the transmitting frequency. This is illustrated in figure 1 [1].

2.1.3 Non-Linear Propagation

While the speed of sound is normally said to be the same for all frequencies, this is not completely true. At higher pressures, or amplitudes (>1 MPa in water [1]) this is no longer a good approximation. The speed at which sound travels through a medium is now related to the properties of the medium and the local particle velocity. In high pressure parts, *i.e.* where particles are highly compressed, there is a slight increase in speed, and vice versa slightly slower in low pressure parts of the wave. As the wave progresses into the medium, this results in the high pressure

parts of the wave gradually catching up to the low pressure parts. This effectively changes the shape of the wave. Compression parts becoming more narrow and increasing amplitude, while refractory parts become wider with lower amplitude. Hence it is possible, and quite easy, to spot a non-linear propagation. Another effect, as shown by Baker, 1989, is a sine wave's growing resemblance to a saw tooth wave with propagation distance, can be seen in Figure 2.

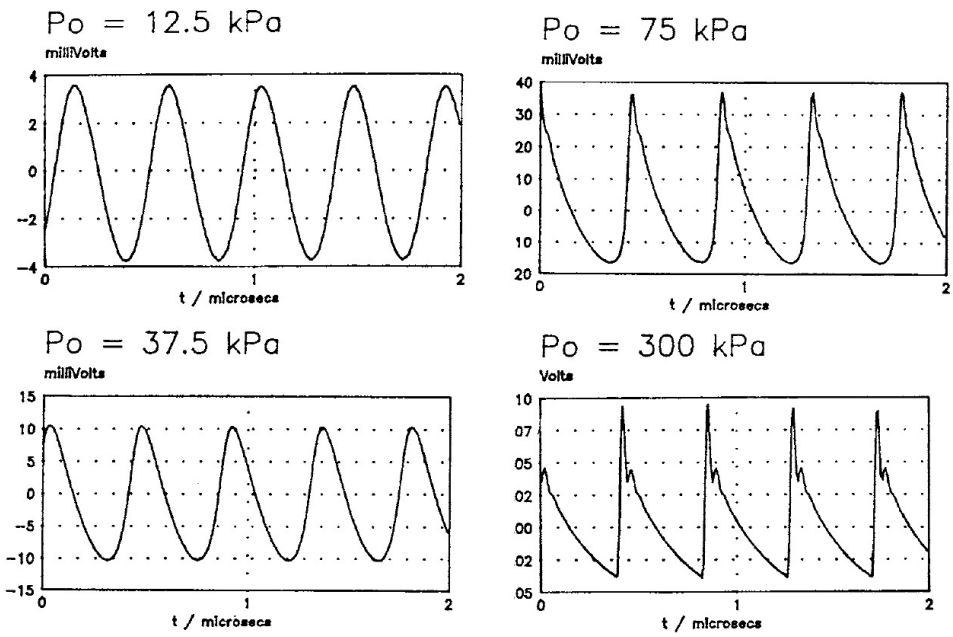


Figure 2: Hydrophone measurements of pressure at z = 700 mm, the far field of a piston source (a = 19 mm) operating CW at 2.25 MHz as the drive voltage to the transducer is increased (from Baker, 1989) [2].

When looking at a frequency spectrum, this narrowing of the compression parts of the wave is also seen as changes in the frequency components of the wave. The swift changes in pressure in the compression part of the wave appear as high frequency components in the frequency spectrum. These components are multiples of the original fundamental frequency, *i.e.* f_0, $2f_0$ etc. Essentially, this phenomenon shows that a portion of the energy of a wave is transferred to higher frequencies due to non-linear propagation. While commonly used in order to create "shock" fronts (*i.e.* almost instantaneous increases in pressure), non-linear propagation is also highly sought after when trying to induce

what is called acoustic streaming.

2.1.4 Acoustic Streaming

Acoustic streaming is where energy is absorbed in a medium to the point where it starts moving in the direction of the wave. Since non-linear propagation shifts the energy of the wave to higher frequencies, and higher frequencies are attenuated more than lower frequencies, it can be safely assumed that non-linear propagation means that more energy is being absorbed by the medium. Hence this is very useful when trying to induce streaming. Equations for inducing acoustic streaming, which were formulated by Nyberg [5] using a successive approximation method, are seen below.

$$\mu \bigtriangledown^2 u_2 - \bigtriangledown p_2 + F = 0 \tag{5}$$

$$F = -\rho_0 \langle (u_1 \bigtriangledown) u_1 + u_1 (\bigtriangledown u_1) \rangle \tag{6}$$

Where μ is dynamic viscosity, ρ_0 is constant equilibrium density, u_1 is oscillatory particle velocity, u_2 is acoustic streaming, P_2 is a steady state "dc" pressure, F is non-linear driving forcing term, and the brackets indicate a time averaging over a large number of cycles. What should be taken away is that a higher viscosity effectively means that more energy is required in order to induce acoustic streaming in the medium [5].

All absorption processes contribute to the acoustic streaming effect including shear and bulk viscosity, relaxation and excess absorption due to non-linear propagation. In liquids with very low dampening or high viscosity, it is hence more difficult to induce a detectable amount of streaming [15]. Since streaming is to be induced *in vivo* , the range at which acoustic streaming is achievable is quite narrow due to the tight energy regulations set by the FDA. These restrictions upon the energy used in master's thesis will be presented in the Safety section.

2.1.5 Transducer

A transducer, or probe, is the component which converts an electrical impulse into a acoustical pressure. This is most commonly done by the use of piezoelectric elements. Connecting piezoelectric materials, such as quartz, to a voltage changes their physical dimensions. Reciprocally,

changing such materials' physical dimensions generates electrical potential gradients. This means that if an alternating current, for instance a sinusoidal one, is transmitted through an element of this material- the same frequency is emitted by the material as sound waves. Since the the piezoelectric elements also work reciprocally, the transducer also has the ability to convert from sound waves to electrical signals. Putting many such elements in a matrix, while having control over delay circuits for each of the elements, one can control the sound waves with quite high precision. This control is called beam forming. However, to improve the transmitting capabilities, there is also a backing layer, a matching layer and a lens. The backing layer effectively cancels out one side of the elements, forcing most of the emitted energy to travel in one direction. The matching layer is, as the name suggests, a means to minimize the difference in acoustic impedances that the wave needs to travel through. The lens is used to focus the many different waves into one direction. The focusing, however, is often mostly accomplished by the use of beam forming. The basic structure of an ultrasound transducer can be seen in Figure 3.

2.1.6 Beam Forming

In order to build an image from echoes, it is preferred to build the image one column at a time. In order to accomplish this, as well as other creating a single wave front, beam forming is used. Beam forming is where the different elements in a transducer are controlled separately in order for their respective waves to interfere at certain points inside a medium. This control includes the power and timing of the pulses from each piezoelectric element, see Figure 4a. Using beam forming one can effectively look along one narrow direction at a time. An expansion of this method is dynamic focusing. When looking at the returning echoes from a pulse and/or looking down a single line, the focusing changes with time. The use of dynamic focusing yields much clearer images. Depending on the transducer and image analysis methods used, an ultrasound machine is able to create both two- and three-dimensional images in real time. For this master's thesis, only the two dimensional, or B-mode, images are of interest.

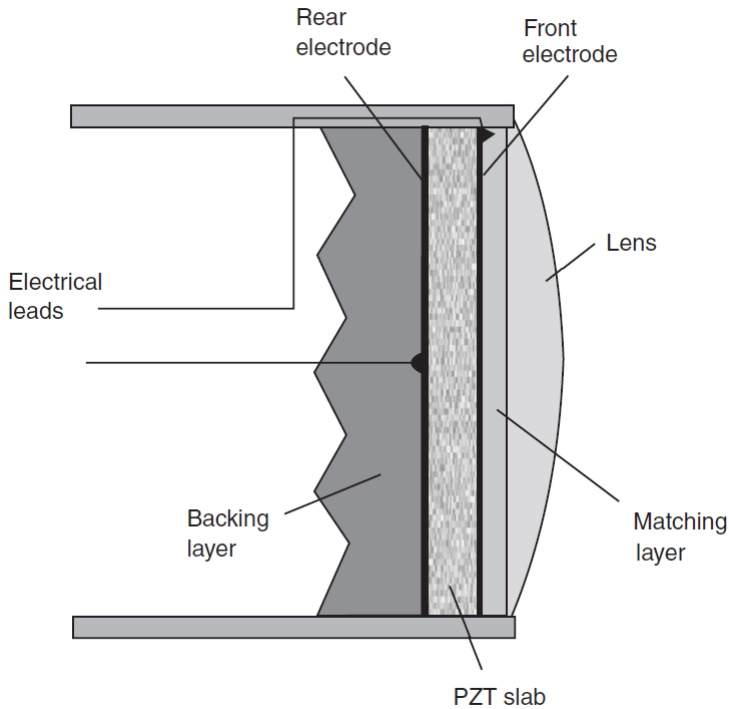
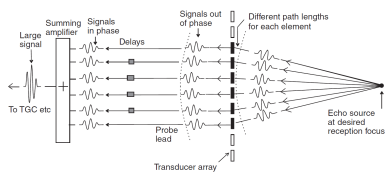
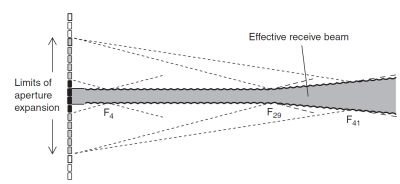


Figure 3: The basic Structure of a transducer [3].



(a) Signals to different elements are delayed in order to aim the combined sound wave [3].



(b) Dynamic focusing and beam forming [3].

2.1.7 B-Mode Images

In order to understand how an image is created, it is easy to first look at how the common A-Mode (amplitude modulation) works. In A-mode, a pulse is sent out from the transducer in one direction, or line. Directly after emitting the pulse, the transducer is switched to listening mode. During this phase the signals go in the opposite direction, and instead of creating a pulses, the amplitude of the returning echoes are registered. These echoes are then displayed along a time axis, depending

on how long after the pulse was sent out, until the echo was received. If this process is repeated along many lines, one can build up a matrix of amplitudes. Where each column represents one direction, and each row represents a return time, or distance. The amplitudes are in turn mapped to pixel values. An example of this sort of mapping can be seen in figure 4. The time-to-distance mapping is determined by the time resolution, which depends on the shape and length of the sent pulse.

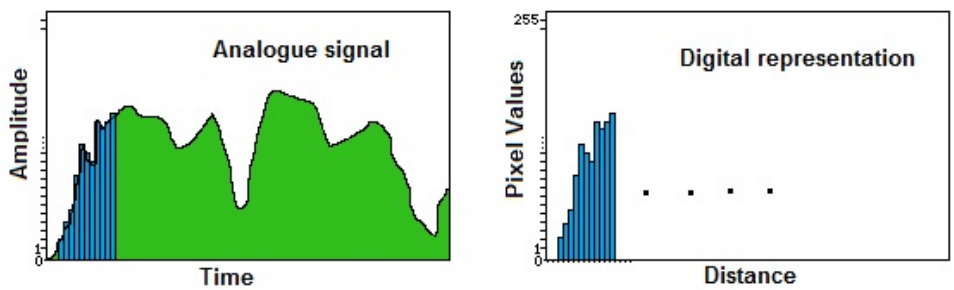


Figure 4: The actual signal/time (green) is integrated over regular time intervals. These integrated values (blue) are then mapped to pixel values, and the time intervals are converted to distance.

As each pixel in an image has finite range, the amplitude matrix is mapped onto this range, effectively losing some information. In the case of the Sonix RP, each pixel is a single number between 0 and 255. These numbers represent a gray scale where 0 is completely black and 255 is completely white. This means, that if B-mode data is used to differentiate between two echoes - this is only always possible if the difference in amplitude between them is greater than 0.39% (1/256).

2.1.8 Doppler

The principle of Doppler measurements are based on the change in observed frequency when, for instance, a sound wave is emitted from a source that is moving relative to the receiver. If the source and the receiver move relatively towards each other, the observed frequency is higher than the emitted. The change in frequency is directly proportional to the velocity of the sound, and most importantly: the relative velocity between the source and the receiver. This relation is expressed in the Doppler equation (7).

$$f_d = f_r - f_t = \frac{2v f_t cos(\theta)}{c} \tag{7}$$

Where f_d is the change in frequency, f_r and f_t the frequency of the received and transmitted signals, and v is the velocity. The speed of sound is c, and θ is the angle between the direction of the beam and the direction of the movement of the moving medium.

Hence, by sending a pulse with a known frequency towards a moving medium, and looking at the new frequency of the received echo pulse, one can determine the velocity of the moving medium relative to the transducer.

Pulsed Doppler is when the transducer sends a pulse and waits for the return of said pulse before sending a new one. This repetition time is called 'PRF' (pulse repetition frequency). Another way to describe this is 'PRP' or pulse repetition period. The relation between PRF and PRP is found in equation 8. The PRF ranges usually between 1-10 kHz in diagnostic ultrasound machines. The reason for mentioning PRP is because this is what Ultrasonix uses in their settings, while PRF is what is usually displayed and mentioned.

$$PRF = \frac{1}{PRP} \tag{8}$$

The PRF determines the maximum velocity that is possible to detect, as well as the depth at which the Doppler can be used. The equation used in order to determine the maximum velocity (v_{max}) that can be measured, using the pulse repetition frequency f_{PRF}, and the pulse nominal frequency f_0 in equation 9[11].

$$v_{max} = \frac{c}{2}\frac{f_{PRF}}{2f_0} \tag{9}$$

The depth at which is possible to use the pulsed Doppler with a given PRF is determined by equation 10.

$$depth_{max} = \frac{c}{2PRF} \tag{10}$$

2.2 Safety

Since ultrasound is absorbed in tissue, and tissue is vulnerable to temperature changes, the Food and Drug Administration (FDA) has set limits regarding the energy and effect allowed in clinical ultrasound diagnosis. The maximum pressure limit, as set by the FDA, is 50 mW/cm^2 for ophthalmic regions (directly on and inside the eye) as measured in water. This limit is lower than for other regions, where it is 720

mW/cm^2, because of the dangers of high pressure waves damaging the sensitive parts of the eye (the maxillary paranasal sinuses are however not considered part of the ophthalmic region despite being very closely situated). Pulsed Doppler ultrasound have been found to produce significant thermal effects, particularly near bone. And recommendations using pulsed Doppler are as follows[10]:

> Care should be exercised to ensure that the examinations are performed prudently using as low as reasonably achievable (ALARA) acoustic output and dwell time. Users should take notice of exposure information provided by the manufacturer and minimise exposures to tissue structures containing bone and/or gas.[10]

For the thermal effects of ultrasound, the FDA has set the limit to 720 mW/cm^2 I_{SPTA} (spatial peak temporal average intensity), and the maximum increase in temperature may not exceed 2°C.

As a reference, values for equipment in clinical use can be seen in Table 2, where intensity is derated by 0.3 dB cm⁻1 MHz⁻1 to compensate for attenuation by the tissue-path.

Table 2: Values for the UK survey of equipment in clinical use. Intensity measured in water.[10]

	Intensity limit I_{SPTA} [mW/cm^2]	
Application	Mean	Maximum
B-Mode	200	1000
Pulsed Doppler	1700	9000
Colour flow imaging	450	2000

2.3 Paranasal Sinuses and Rhinosinusitis

The most prominent, and the sinuses that are the focus in this report, are the maxillary paranasal sinuses. These are air filled pockets located on either side of the nasal cavity approximately 4-6 cm deep (et al. Jannert, Malmö 1982).

In front of the maxillary paranasal sinuses is the Maxilla facial bone, or canine fossa. Through measurements using CT by (et al. Sahlstrand Johnson 2011) this bone has an approximate thickness of 1.1 +-0.4 mm.

Rhinosinusitis is an infection, caused by either bacteria or virus, in the paranasal sinuses. Acute rhinosinusitis, or ARS, is the fifth most common diagnosis for which antibiotics are prescribed, according to the US National Ambulatory Medical Care Survey. [4] It is, according to European Position Paper on Rhinosinusitis and Nasal Polyps, or EP3OS, defined as (Fokkens et al. 2007):

> Inflammation of the nose and the paranasal sinuses characterized by two or more symptoms, one of which should be either nasal blockage/obstruction/congestion or nasal discharge (anterior/posterior nasal drip): +- facial pain/pressure, +- reduction or loss of sense of smell and either:
>
> endoscopic signs of: -polyps and/or mucopurulent discharge primarily from the middle meatus; and/or -oedema/mucosal obstruction primarily in the middle meatus,
>
> and/or -changes seen in computed tomography (CT) images: mucosal changes within the ostiomeatal complex and/or sinuses.

At the University Hospital in Skania, Sweden, (Skånes Universitetssjukhus) or SUS, the diagnosis for rhinosinusitis commonly involves ultrasound. An ultrasound probe is used in order to detect whether or not there is fluid inside the patients' paranasal sinuses. This is accomplished by simply looking for an echo from the back wall of the paranasal sinus cavity. If such an echo is detected, it can be determined that there is fluid present inside the cavity.

Studies indicate that ARS with serious sinus fluid, with low viscosity, is caused by viruses, while mucopurulent sinus secretion, with high viscosity, is caused by bacteria (Carenfelt and Lundberg 1977; Carenfelt et al. 1978). Since antibiotics are only needed when the infection is caused by bacteria, determining the viscosity of the fluid inside the cavity would potentially decrease the unnecessary use of antibiotics in rhinosinusitis patients.

However, antibiotics are still being administered in vain, because the method using ultrasound (as well as the another method using CT) only shows the presence of fluid, while offering no information of the fluid's properties.

The only way to determine the viscosity is currently to perform a sinus puncture. This involves penetrating the maxillary sinus cavity

with a needle via the nose through a bone wall, and is an uncomfortable invasive procedure.

Using ultrasound for the staging of rhinosinusitis may be a way to spare patients this procedure. Since the maxillary paranasal sinuses are not included in the ophthalmic region, the maximum allowed energy is 720 mW/cm^2 [12]. Hence, the energy should be sufficient to induce a detectable amount of acoustic streaming in the low viscosity fluid. It may not, however, be enough for the high viscosity fluid and thus the way to distinguish between the two types of rhinosinusitis could be the detection of a velocity >0.

2.4 Sonix RP

Figure 5: The Sonix RP

The clinical ultrasound scanner used in this master's thesis is the Sonix RP, made by Ultrasonix. It is basically a personal computer with hardware added to support ultrasound processing. It uses Windows XP Professional as its base operating system. The motherboard is a 'Asus

P4P800-VM Intel 865G 4xDIMM Socket 478 mATX Bulk' and, as the name of the motherboard suggests, it uses an Intel P4 processor. The Sonix RP has 1GB RAM. The graphics card used is an AGP Nvidia GTX 6600.

There are also three PCI expansion cards. One allows more serial(RS232) connections to the computer and another is a modem to allow a dial-up connection. The last expansion card is a custom card from Ultrasonix that enables use of the ultrasound transducer as well as powering the console. The console is a physical interface with a small touch screen, a keyboard and trackball, but also ultrasound specific controls, like Time Gain Control and PWD/B-mode switches. The touch screen enables the user to change mode specific settings, like frame rate and focus settings in B-mode.

Ultrasonix also provide software to be able to record and display the received ultrasound. A screenshot of the software can be seen in figure 6.

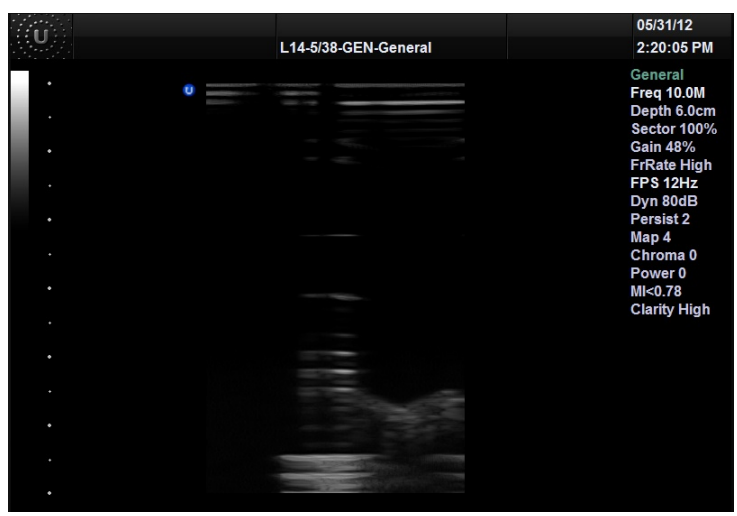


Figure 6: The Exam software provided by Ultrasonix

There is a button on the console called "Research Mode". When in Research Mode the user is able to change different parameters, for example voltage for a range of frequencies in Pulsed Wave Doppler. There are also xml files that need to be altered to get some of the research parameters to work with the exam software. Each probe type has its own xml file where probe specific settings are stored, for example frequency range for different modes.

The transducer used in this master's thesis is L-14/38mm. It has 128 elements and a center frequency of 7.2 MHz.

2.5 Programming

2.5.1 C++ vs Java

The implementation of the diagnosis method requires programming in C++, and as many who read this master's thesis will be unfamiliar with this programming language, while quite experienced in Java- a short introduction to C++ follows. Compared to the programming language Java, C++ offers the user more control over data and memory usage, while being less user friendly. C++ is, in other words, a lower level programming language than Java. Most of the syntax is very similar, but the low level control over the execution of a program that C++ offers, puts more pressure on the programmer. This control, and possibly the biggest difference between them, comes from C++ making use of pointers.

Pointers are essentially addresses to where a certain object is located in the memory, which are sent instead of all the actual data contained at the address location. This is done to minimize the memory usage of a given function. Using pointers may however result in 'NULL pointers' and 'pointer exceptions'. These occur if the address provided to a function or action is currently occupied by some other process, if it is empty, or contains something other than what the pointer suggests. While in Java, the process of sending pointers and handling addresses are highly regulated, they are not in C++. Hence this is the most common problem facing many programmers who use C++ or other lower level languages.

In order to simplify using a low-level language, such as C++, one often makes use of a software development kit, or SDK. An SDK generally provides the user with functions and objects (or classes) that handle certain common, but complicated, actions. They can for instance provide means to interact with hardware or machines, as well as the means to display images and create a user interface.

2.5.2 Ulterius

Ulterius is a software development kit (SDK) developed by Ultrasonix. In essence, it is an interface to the Sonix RP machine which simplifies

access to certain capabilities and settings of the machine. Ulterius allows full control over the imaging parameters of the ultrasound machine and the ability to remotely connect to it. Once connected, it is possible to both set and check certain parameters, as well as collect the data gathered by the ultrasound machine. Such data include, but are not limited to, B-mode images, RF-data (raw format data) and FFT (fast fourier transform) sequences acquired from the PW-Doppler (pulsed wave Doppler) mode.

At the time of this master's thesis, Ulterius uses something called 'callbacks'. A callback is an interrupt signal that triggers the use of a 'callback function'. Once the callback function is done operating, it can be called upon again. This enables a program to be put into a wait-for-update state and effectively update or stream images continuously from the ultrasound machine. At the time of this master's thesis, Ulterius needs to be compiled using a Microsoft Visual C++ compiler. Ultrasonix are moving towards using Qt Creator in their demo programs but, at the time of this master's thesis, this is not yet available in Ulterius(Version 5.7.3).

Some restrictions to Ulterius are controlled by '.xml' files on the Sonix RP system. These files contain parameters which govern the way that the imaging parameters can be adjusted, both inside of the main program as well as through the use of programming environments developed by Ultrasonix (including Ulterius). Hence when changing parameters, one must check and/or change the '.xml' file of the probe used in order to be sure the new value is viable.

2.5.3 Qt

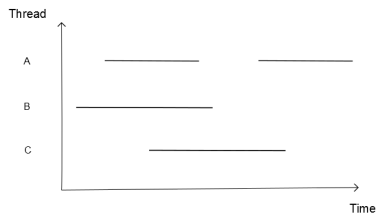
The Qt SDK is commonly used through Qt Creator, which is an open source cross-platform integrated development environment (IDE) developed by NOKIA. It is used by over 450,000 developers in more than 70 industries to build advanced applications and devices[6]. This is due to its many utilities, and rather quick learning curve. For this master's thesis, Qt Creator has two main strengths that are needed. The first one being that Qt Creator simplifies creating a graphical user interface (GUI). The second that it is also possible to utilize the Microsoft Visual C++ compiler. It does not, however, use callback functions. Instead, Qt Creator utilizes signals and slots.

Signals and slots are quite similar to the callback functions. Just as with the callback functions, a signal is emitted when a certain event

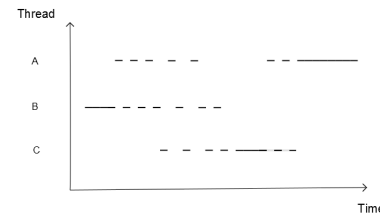
occurs. If said signal has been connected to a slot, the code in that slot is handled much like the callback function. The key difference is that the signals used in Qt are type safe. This means that a slot, unlike callback functions, can never be called with incorrect arguments. A callback function may even crash if called with incorrect arguments. This can never happen with signals, as slots only respond to signals with the correct signature. A slot can also ignore extra arguments, making them even safer to use [6]

2.5.4 Mutex

Most operating systems switch between processes/threads/programs many times a second to give the impression that they are concurrently running. The threads are however run one at a time with each process given a timeslot of the processor to run in.



(a) Apparent processor usage



(b) Actual processor usage

Problems occur when two or more threads share resources, for example one part of the memory or a variable storing data. An illustration is the best way to explain one of the problems that may occur. Imagine that you have 5000 SEK on your bank account. You are at an automatic teller machine(ATM) and you are about to withdraw 1000 SEK but at the same time a friend is about to deposit 7000 SEK into your account.

A	B
balance = getBalance();	
	balance = getBalance();
balance = balance - 1000;	
	balance = balance + 7000;
setBalance(balance);	
	setBalance(balance);

(c) The resulting balance is 12000 SEK

A	B
balance = getBalance();	
	balance = getBalance();
	balance = balance + 7000;
	setBalance(balance);
balance = balance - 1000;	
setBalance(balance);	

(d) The resulting balance is 4000 SEK

The account balance is of course supposed to be 11000 SEK in both cases. Neither you nor the bank wants the concurrency errors above to occur. What is needed is mutual exclusion, that is, only one process may access shared resources, or critical sections of code, at a time.

A mutex can be seen as a padlock with only one key. Only the keyholder can open the padlock and use what it protects. That is how the mutex works. If a thread claims the mutex, no other thread can claim the mutex. When other threads try to claim the mutex they are put to sleep until the mutex is released by the thread currently holding it.

It could get rather complicated however if several mutexes are needed. One issue one has to think about while using mutexes is deadlocks. A deadlock is when two or more threads are waiting for the other/others to finish and because they are waiting- neither of them ever do. Below is an example to show a deadlock.

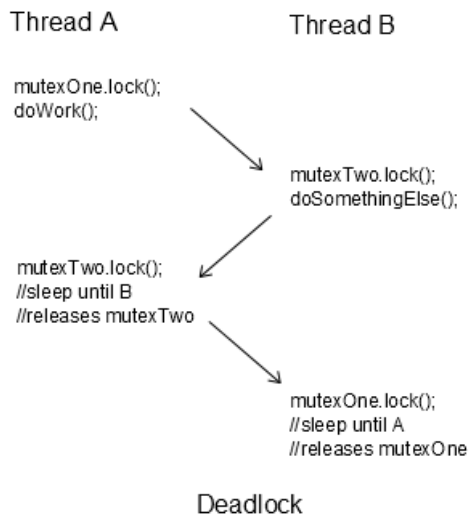


Figure 7: A simple example showing a deadlock

There are other tools to ensure concurrency correctness like semaphores and monitors, but this master's thesis does not make use of them.

3 Method

3.1 Approach

The implementation of the diagnosis program will be created specifically for the ultrasound machine Sonix RP, developed by Ultrasonix corporation. This is due to one being available at the Department of Electrical Measurements, Lund Institute of Technology, and the possibility of creating custom applications for said machine.

Diagnosis should begin by establishing the presence of fluid inside the paranasal sinuses. In order to accomplish this task, the doctor uses B-mode to find an echo originating from the back wall of the maxillary paranasal sinus cavity. Hence the program needs to start in B-mode. Once the back wall is found- the program should find the frequency for which the back wall echo is maximized. This is to ensure that as much energy as possible is transmitted through the front wall into the fluid. The reader should note that, as stated earlier, the bone wall in front of the paranasal sinuses vary between patients and the optimum frequency thereby varies as well.

After finding the optimum frequency, the program should change to a Doppler setting. When the Doppler is running, the program should display the velocity of the measured flow inside the cavity. Finally data needs to be collected in order to establish some sort of threshold that would indicate the viscosity, or the nature of the fluid. If the viscosity is low, a higher velocity should be detected compared to if the viscosity of the fluid is high. Meaning that if the detected velocity of the acoustic streaming is higher than a (yet to be) established threshold, there will be no need for administering antibiotics to the patient.

In order to be viable for use in diagnosis, the program should be easy to use and yield consistent results which are easy to interpret. After an initial walk-through of the program, the user should have no difficulty to use the program to make a valid diagnosis.

3.2 Sonix RP Machine

At the beginning of this master's thesis, the ultrasound scanner (Sonix RP) did not start properly. Upon attempting to start, it would go into an endless boot-loop. The unit would reboot just as Windows was about to load. At first the hard drive was thought to be broken but when Windows was started in safe mode a few files would load,

indicating that it might just have been a corrupt system file.

A recovery CD was used to format the hard drive and reinstall Windows. The reinstall was successful but, for whatever reason, the Ultrasonix software did not accept the license provided by the manufacturer. After several sessions with tech support it was decided that it was likely some part of the hardware, probably the company's custom PCI card, causing the licensing error. The Sonix RP was dismantled and the modulo(computer chassis) was removed to get access to the hardware. A PCI Modem was removed completely as it no longer served any purpose. The custom PCI card was then put in the PCI Modem's slot in order to see if it was simply the PCI slot on the motherboard that was faulty. After putting the machine back together it started but gave an error saying it could not connect to the Console. Parts of the console had power as some of the buttons were lit and the display on the console had power.

The Console is connected by three serial(R323) cables, which were swapped around to eliminate broken connectors. There seemed to be no difference changing the serial cables, since still only parts of the Console were lit up. After another reseating of the PCI cards the Sonix RP would no longer manage to get through BIOS. It only showed a black screen. A number of graphics card were tried to no avail. The BIOS battery was removed in order to reset the BIOS. The BIOS settings could have been corrupted, but just resetting the BIOS was not enough to solve the issue. The motherboard was likely broken and an identical board was found and bought.

After the motherboard was changed, Windows and the Ultrasonix software was still not operating properly and another reinstall was attempted. The console still did not receive any power, and so the CD/DVD reader inside the console did not work. Several IDE DVD readers were tried but none of them worked. At first the DVD was thought to be scratched and unusable. Tech support thereby offered an ftp with the Recovery DVD as an ISO. Something was wrong with the routing or the file on their server because it would stop downloading at 41.3 MB every time. By chance the DVD was brought home and tried on a SATA DVD reader, and the DVD could then be read. That reader was then connected to the Sonix RP and windows could be reinstalled once again. After the reinstall the Sonix RP worked again.

3.3 Modeling the Paranasal Sinuses

A 6x4 cm (outside measurement) acrylic glass box, with 0.4 cm thick walls, is used to model the walls of the paranasal sinuses. Milk with 3% fat is used to approximate the fluid in the cavity. The milk should be a fairly good representation of the low viscosity fluid that is secreted during bacterial rhinosinusitis. Trials using cream to increase the fat percentage further was tried, but failed to induce any kind of streaming. Water was also found to not work as well, since the absorption coefficient in water is very low. The results only include the trials where the 3% milk was used as those were the most representative of the low viscosity fluid. In the back of the acrylic glass box, there is a piece of sound dampening material. The same material that covers the walls of the glass container in the hydrophone tests. Due to the quite small echo reflected off said surface, this created a more accurate representation of how a back-wall echo would appear in actual patients. The maxillary sinus cavity is rarely square in form, and so the non-flat shape of the material further improved the model.

Figure 8: The model of the paranasal sinuses

3.4 Programming

Initial trials were made using Visual Studio 2010 Express. However as, for some reason, the GUI demo by Ultrasonix was unable compile,

it was decided to switch from Visual Studio. Qt Creator was chosen since it has powerful tools for creating user interfaces plus the ability to utilize the Microsoft Visual C++ compiler which is needed in order for Ulterius to work.

In order to create and display images, Qt Creator has numerous available options. After some failed attempts and a bit of forum research, 'QLabel' seemed like the most promising option to use. During these trials it was discovered that the image displayed in the QLabel could only be updated from within the 'MainWindow' class, which needed some sort of work-around. It was concluded that this issue could be solved by the use of signals. The MainWindow class is the main user interface class used in Qt Creator.

The callback functions used are triggered every time a new image has been processed by the ultrasound machine. The callbacks provide the data from the image, certain characteristics of the image itself, and parameters describing the data type.

3.5 Safety

As the power setting on the Sonix RP is maxed out (15V) and the probe is to be used on an area close to the eye of a patient - it is important that the energy emitted does not exceed safety limits.

In order to test the energy emitted from the transducer probe two types of measurements were made. The first test involved a highly sensitive scale. A small container filled with water, with an absorbing target on the bottom, was placed on the scale and the transducer was suspended above the water surface with its tip submerged. The scale was calibrated to zero while the transducer was silent. The transducer was then switched on and the resulting change on the scale was noted.

Radiation pressure is force per unit area at a target, and so the total force acting on a target, or radiation force, is obtained by integrating the radiation pressure over the target. Provided that the whole beam is intercepted, the radiation force F is the determined by equation 11. Where W is total power, c the speed of sound, and h is a factor depending on the type of target used. As stated before the target is an absorbing one, hence h= 1 [14].

$$F = \frac{hW}{c} \tag{11}$$

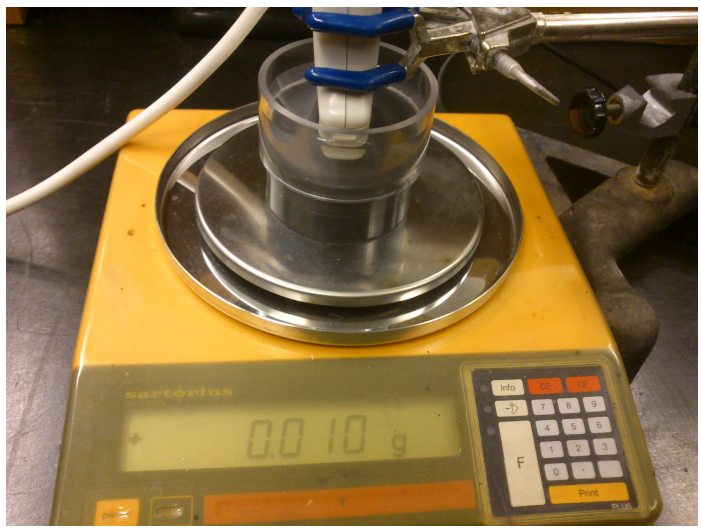


Figure 9: The transducer is suspended above and into a container with water and an absorbing target, which is placed on the scale.

The second test was done in a water tank by sending pulsed Doppler wave signals through water and analysing the pulses a few centimetres away with a sensitive hydrophone. Since the Doppler mode uses beam-forming, the detected energy of the pulsed waves are very different depending on how the hydrophone is positioned compared to the ultrasound probe. In order to find the optimum distance, a program developed at the Department of Electrical Measurements, at LTH, was used. This program controls very precise motors which enables the user to move an object with high precision in x, y, and z direction while it also has the ability to fetch and analyse data from an oscilloscope. By combining these capabilities it is possible find the optimum position for the hydrophone as well as actually measuring the energy of the pulses from the pulsed Doppler.

The walls of the water-filled tank is laced with rubber walls designed to dampen any residual sound waves. The water in the tank is left to sit for many hours to make sure that there are as few bubbles as possible. While measuring, removing bubbles from the transducer and hydrophone is vital for accurate measurements.

The hydrophone used is a Precision Acoustics 0.5 mm interchangeable probe made by Percision Acoustics, for which the calibration has expired. It has a capacitance of 24 pF$\pm$3 pF and is correctly terminated through 50 Ω to an HPI submersible pre-amplifier. The signal

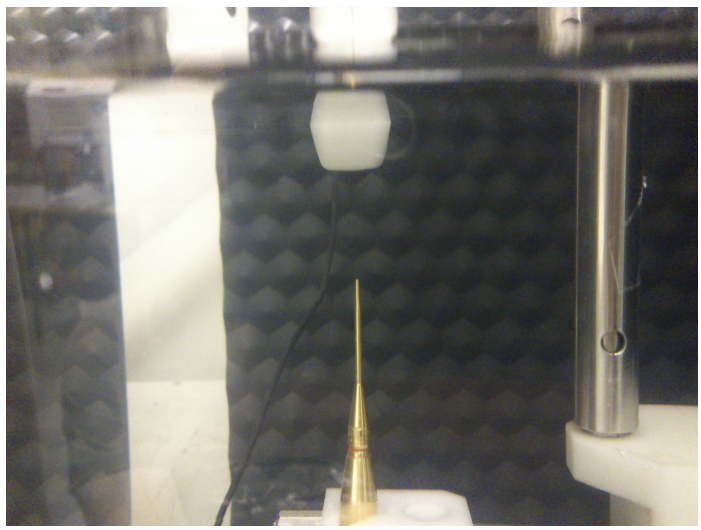


Figure 10: The transducer is positioned just under the surface of the water. The position of the hydrophone is then adjusted for maximum amplitude.

goes to an analogue oscilloscope (model: Tektronix TDS 360, made in Beaverton, Oregon) which is connected to the computer. The computer runs on Windows XP Professional and a Labview VI developed at the Department of Electrical Measurements, LTH. While the hydrophone's calibration time has expired, it is mainly used to get approximate figures.

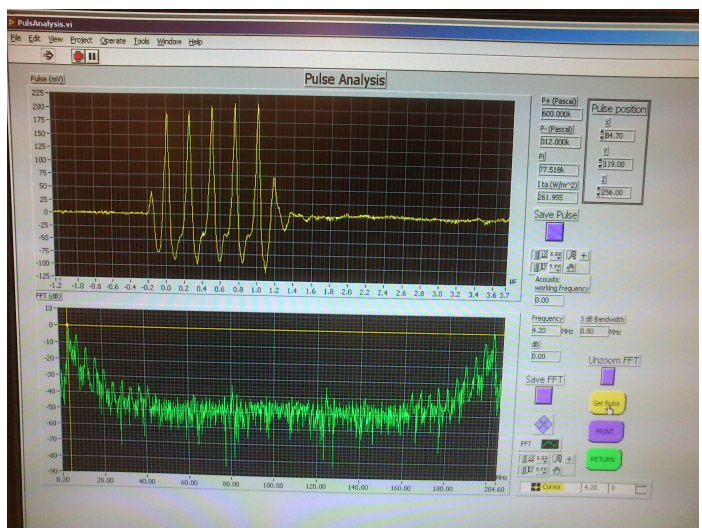


Figure 11: The Pulse Analysis program. It displays one of the pulses emitted from the transducer, with the spectrum underneath. Some calculated characteristics can be seen to the right.

4 Results

4.1 Sonix RP

The Sonix RP is working again. However, the probe used during the project is not fully functional. What needs to be done is either to replace the probe or reattach/replace the silicon layer of the probe. It should be noted that while B-Mode is functional, both energy output and the detection capabilities of the probe may be lowered to some extent. While the difference in results between the probe used, and what would be acquired with a fully functional one, may be insignificant- it should be kept in mind when looking at the results from both the milk trials and energy measurements. The process of fixing the Sonix RP can be found in section 3.2. A section concerning the probe is found in section 4.4.

4.2 Programming

4.2.1 Coding process

The first version of the program was made before the Sonix RP was fully operational. The focus was to get the Graphical User Interface(GUI) as complete as possible so focus could be put on algorithms and collecting data from the Sonix RP. Creating a GUI is usually quite cumbersome in C++, but working with Qt helped things along. Integrating Ulterius with the first version was troublesome however. This was mostly due to Ulterius requiring its callback function to be declared static while at the same time displaying widgets in Qt had to be non-static.

Objects declared static are always in memory. They are created the first time their declaration is encountered, and are removed only when the program terminates. In turn this means that any static functions are only able to call upon other static functions [9].

There are two ways of connecting to the Ultrasonix software through Ulterius. Either connecting through the internet using the Sonix RP internal IP address, or running an Ulterius application on the Sonix RP machine and connecting to local host *e.g.* itself. Running the program locally on the Sonix RP machine was found to run much more smoothly than a remotely connected application. This is of course due to the latency which comes with a connection through the network. Such a latency also increases the interval between each callback. The need for quick calculation from image data is thereby much less pressing, as the

computer has more time before the images are updated. This was made clear when the application was run locally on the Sonix RP machine.

The second version was running everything in a single thread, due to simplicity. Once the application was running locally, the computer could not keep up with the repetition frequency of the callback being called. The function 'QImage::Pixel()' proved to be too computationally heavy for the Sonix RP machine's processor to manage all calculations in time for a new callback. Therefore a version of the application using multiple threads was needed. The final version's class structure can be found in section 4.2.3

A B-mode image uses the amplitude of the returning echoes in order to create an image. Making use of Time Gain Control and continuous focusing in order to determine the origin of the echo, the calculated amplitude value is then mapped to an XRGB value. These values range from 0 to 255, which means that the data contained in the image is already compressed.

While the more exact amplitude of the echo is lost in a B-mode image, it is still quite possible to use the information contained in order to compare the penetration of different frequencies. By selecting an area of interest and then comparing the pixel values of the areas for different frequencies, it is possible to evaluate their respective echoes and, consequently, penetration through a medium. Given that the difference in amplitude between the frequencies is bigger than 0.4 percent, the distinction is easily made.

4.2.2 Graphical User Interface

The program, called Snotalyzer has had many capabilities which are not present in the final version. For instance, it contained the code to display as well as save RF-data to a file, which could be used together with Matlab. In order to simplify use, a low number of buttons have been prioritized. A few functions are accessible through the menus, but are not essential, or even recommended for novice users. In the settings menu, the user has the ability to change the IP, PRF, and Doppler measurement time.

The final version of the front panel only includes B-Mode, and Doppler buttons, as well as TGC slider controls. All these are essential controls which without, the user would be ill-equipped to make a diagnosis. To help the user, there is also a text display which tells the user what is happening. The upper large display shows the B-Mode

image, while the lower one displays the Doppler shift signal acquired in Doppler mode. An image of how the user sees the program can be seen in figure 12. A more detailed description as well as a user guide can be found in the appendix.

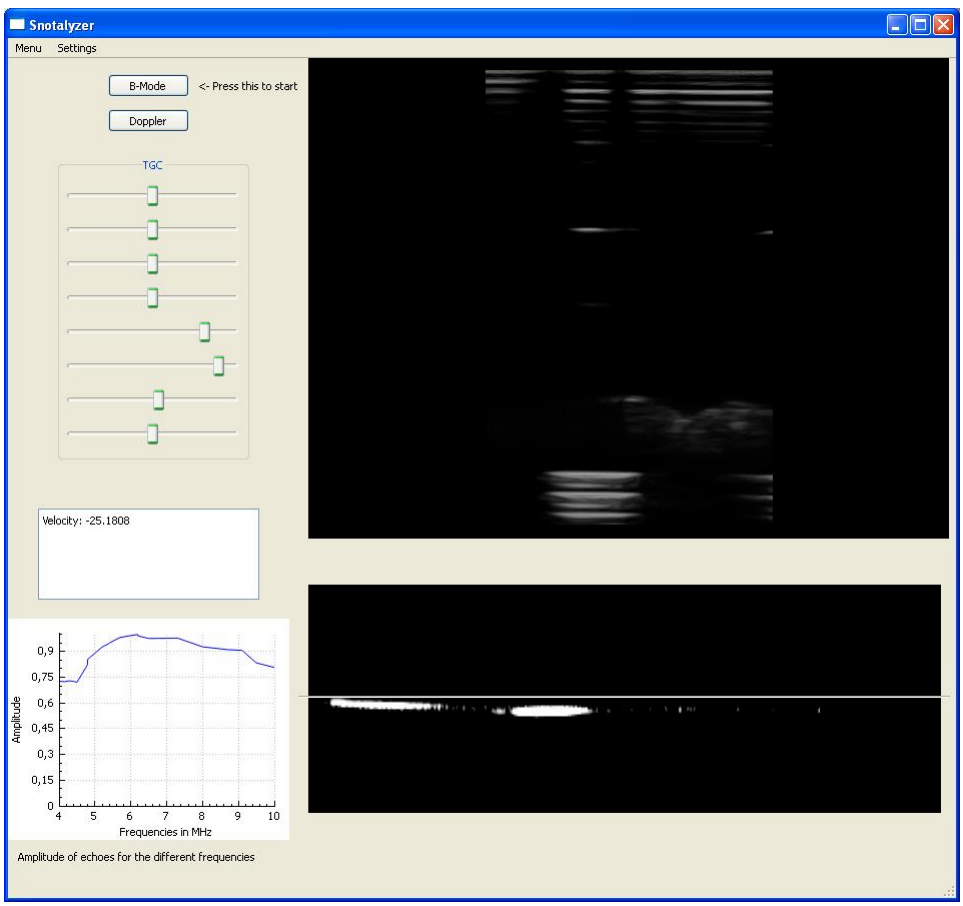


Figure 12: The User Interface Panel of Snotalyzer

4.2.3 Class Layout

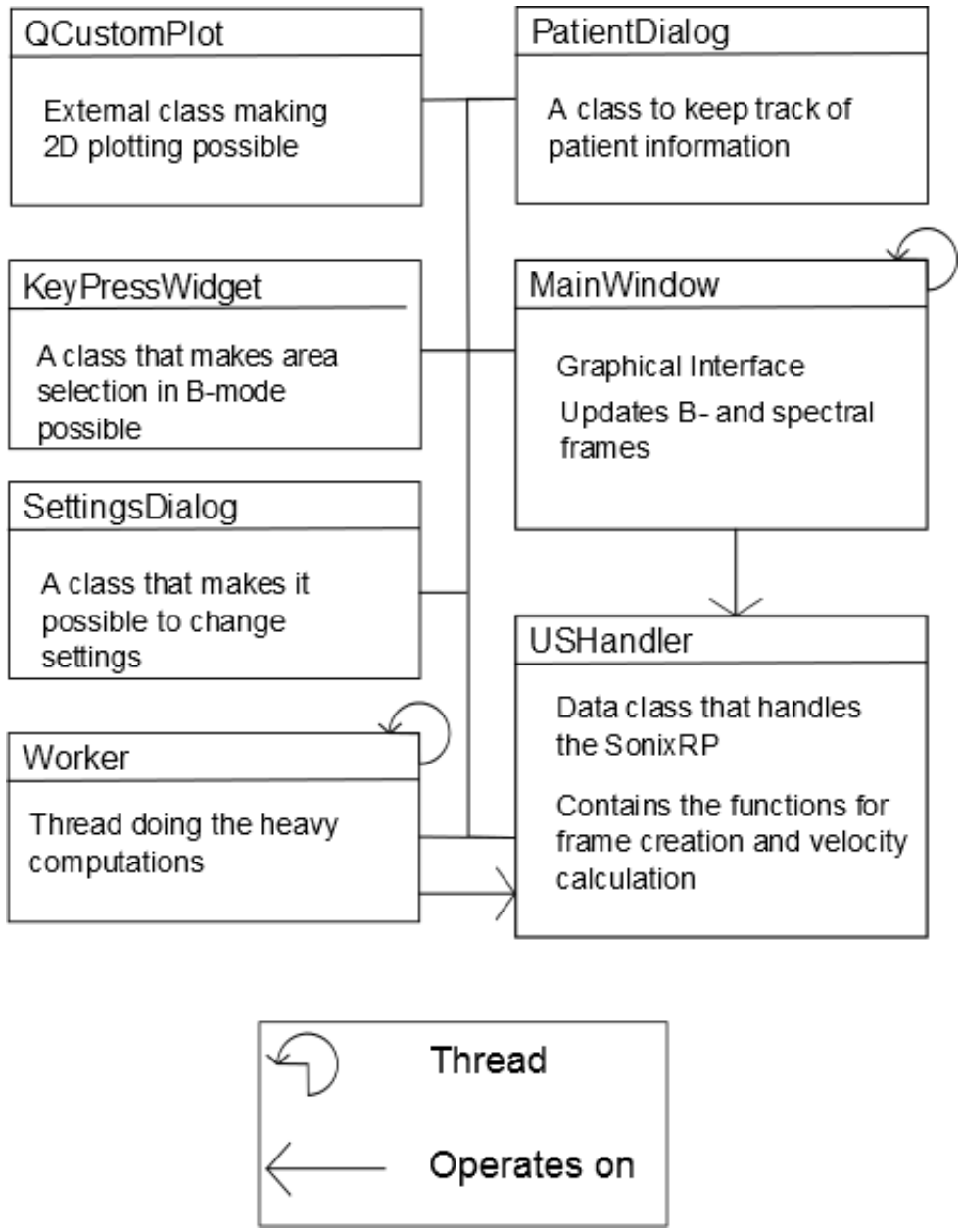


Figure 13: Layout of the Classes in the final program.

- MainWindow is the main class of the program. It is a thread handling the user interface. A few images of the Graphical User Interface can be found in section 4.2.2.

- QCustomPlot is an external class written by DerManu [7]. QCustomPlot is an easy way to plot data. Qt does not have an internal tool for 2-D or 3-D plotting. There are other external libraries like qwt, but they are overkill for a simple 2-D plot.

- KeyPressWidget is a class/widget that makes it possible to select an area of a QLabel. The user can select an area in the B-mode image and let the program maximize the intensity by going through a range of frequencies.

- USHandler is a class that handles everything having to do with Ulterius. It will set up a connection or disconnect from the Sonix RP. The class has callback functions that copy image data to a buffer from memory. The class also has two computational heavy functions that runs in another thread, Worker. The shared resources in USHandler are protected by a mutex.

- The worker thread is a thread that operates on USHandler when it is signalled from the MainWindow thread.

- SettingsDialog is a class that extends QDialog. It enables the user to choose which IP to connect to. The Pulse Repetition Frequency (PRF) in Doppler mode can also be changed with a slider. It is also possible to change for how long each Doppler measurement run.

- PatientDialog is another class that extends QDialog. It is used to enter patient specific data, for example social security number.

For the full code, see Appendix B.

4.3 Safety

4.3.1 Scale Test

The first transducer test was done with a sensitive scale. Where the Doppler signal was sent through water onto the scale. This should effectively measure the continuous impact caused by the Doppler signal, and then quite easily translate into energy transferral. By measuring the difference between the signal being sent, and the transducer being silent.

The scale was very sensitive. By only touching the table that the scale was on with a finger, the scale would alter the value presented on the display. However, the only time we could get a proper reading on the scale was with the highest pulse repetition frequency possible, 12.5 kHz. At these settings, the scale showed 10 mg, which translates to 151 W/m^2 through equation 11. This is well below 720 W/m^2 which is the maximum limit for diagnostic ultrasound.

4.3.2 Hydrophone Test

From looking at the pulse shapes, energies and number of periods in each of the pulses sent out, it was found that only 7 different frequencies were being emitted. See figure 14. The variations within each span is consistent with the margin of error for the measurements. After this was established, the following results were obtained from one frequency in each of the different frequency spans.

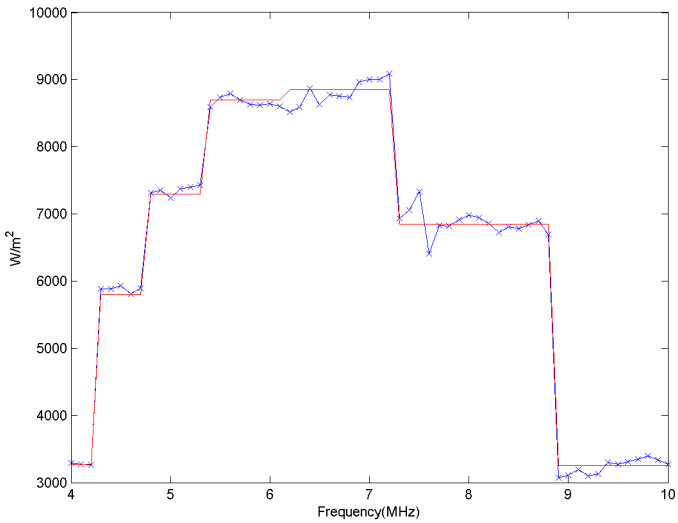


Figure 14: Pulse Energy Stages - Measured Energies

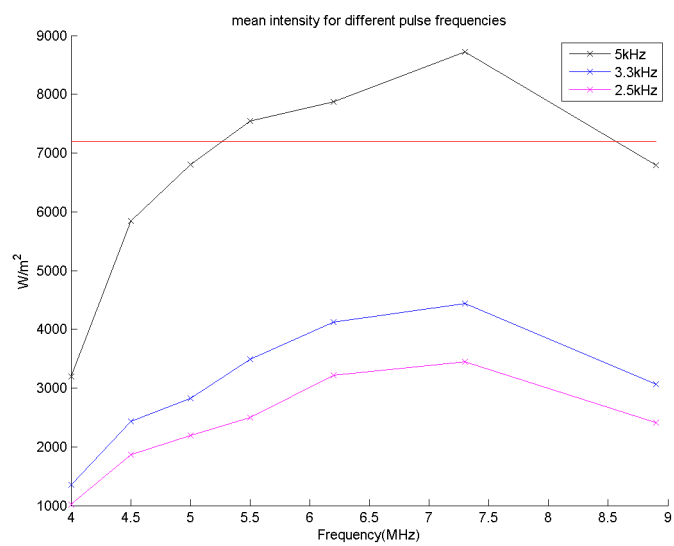


Figure 15: Mean intensity

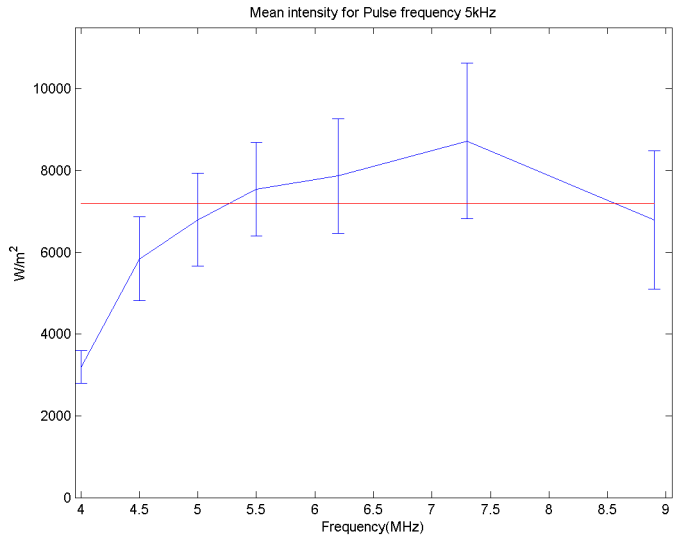


Figure 16: 5kHz pulse repetition frequency with standard deviations

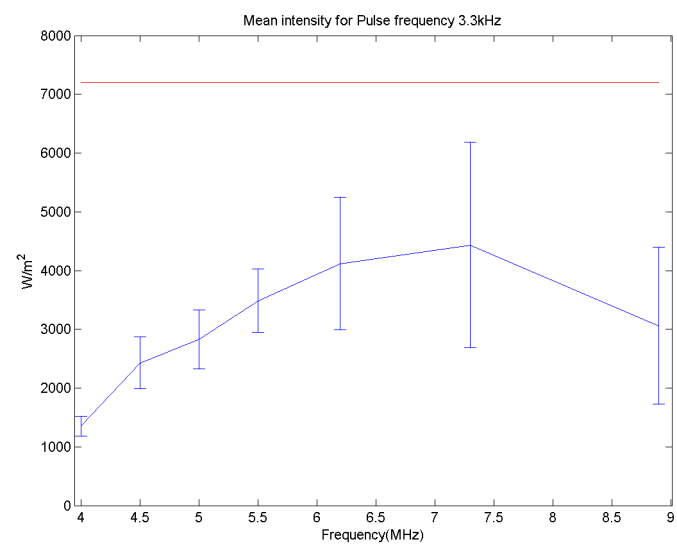


Figure 17: 3.3kHz pulse repetition frequency with standard deviations

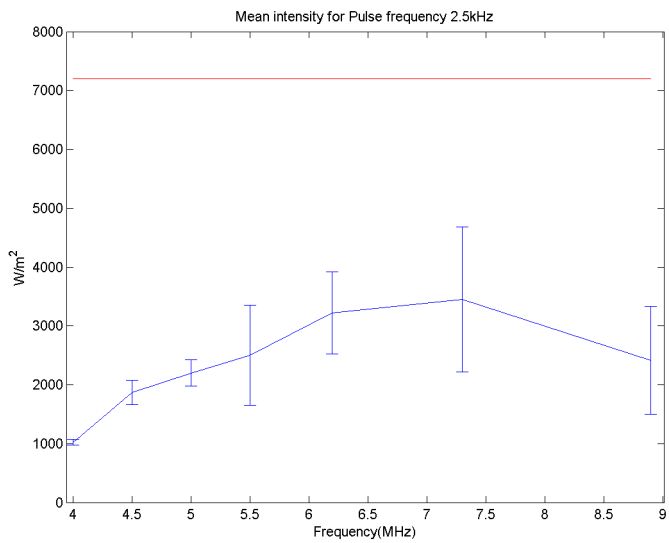


Figure 18: 2.5kHz pulse repetition frequency with standard deviations

As seen in figure 16, when using a pulse repetition frequency (or PRF) of 5kHz - the energy of the pulses exceed the safety limits set by the FDA. Lowering the PRF has a direct correlation with the energy induced into the medium. This can be seen in figure 15. Using a PRF of 3.3kHz puts the Doppler pulse energy safely under the limits, and would thereby preferably be the PRF used in a patient study.

In figure 19 is one of the largest pulses detected during the hydrophone measurements. It also illustrates the non-linear propagation mentioned in the theory section.

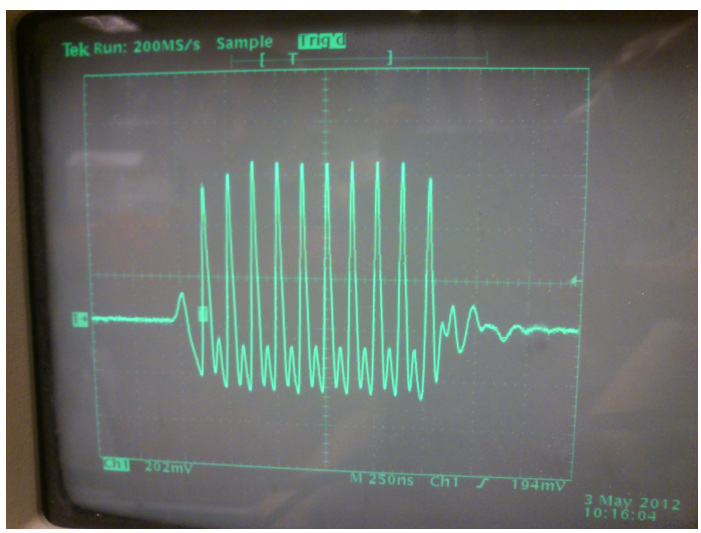


Figure 19: Pulse on oscilloscope. The triangle-like shape indicates non linear propagation.

4.4 Transducer Glitch

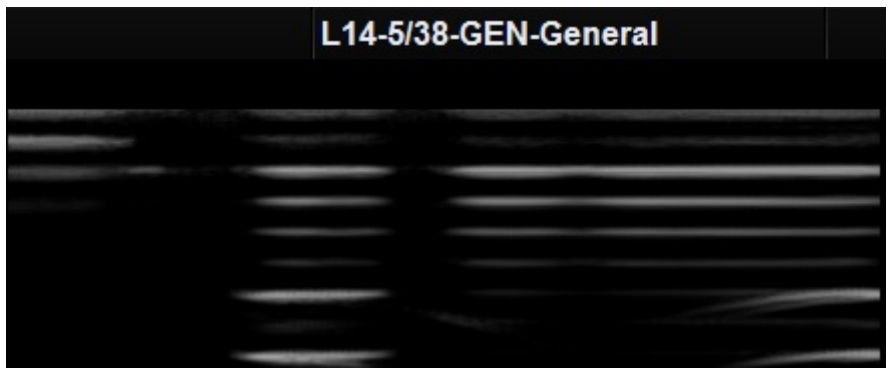


Figure 20: Cropped screenshot of the Exam software, note the black vertical areas

When using a pulse repetition frequency of 12.5kHz and a voltage of 15V across the elements during the scale tests(see section 3.5), parts of the image went black. Moving the area of interest, the pulsed wave Doppler gate, to another section yielded the same result: this section also went black.

The Doppler measurement is currently done using the lower half of the transducer where all elements appears to be fully functional.

4.5 Trial Results

The model of the maxillary paranasal sinuses used in the trials can be found in section 3.3. The whole procedure of finding the optimum frequency from the back-wall echo was done prior to each measurement. A measurement time of 10 seconds during which a pulsed Doppler of the calculated optimum frequency was being sent into the model with the transducer fixed.

The optimum frequency found for the acrylic glass box with milk was found to be 6.6MHz (which is within the span ranging from 6.2 to 7.2MHz). The mean value of the maximum velocities found during each test can be found in table 3, and illustrated in figure 21.

Table 3: Mean maximum velocity found during milk trials.

PRF [kHz]	Mean Velocity [mm/s]	Standard Deviation
5.00	-38.1296	2.5215
4.00	-23.9283	3.3224
3.33	-20.4266	3.7682
2.50	-21.1075	1.7830
1.67	-16.3413	0.6419
1.25	-15.3578	1.7060

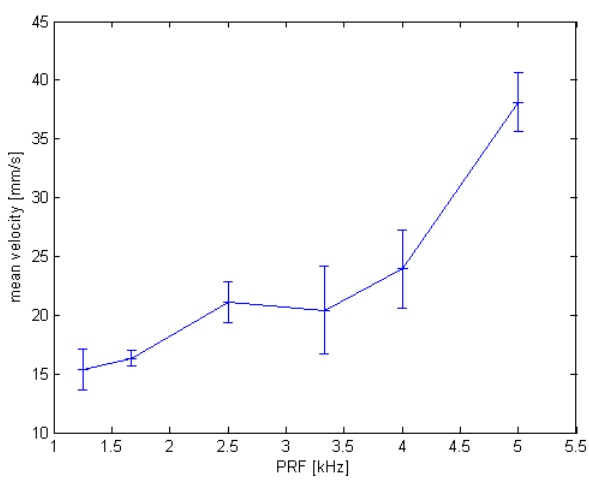


Figure 21: The maximum mean velocities found for each PRF.

4.6 True Frequencies

As stated in previous sections, the Sonix RP is only able to emit certain frequencies. That means that regardless of how accurate the settings in the .xml file are, and what the software tells you is being emitted, the actually transmitted frequency is often different. The actual frequencies in each span can be found in table 4. Upon contacting the manufacturer concerning this issue with our results, it was confirmed that the Sonix RP was only able to emit frequencies of 40/X MHz.

Table 4: The actual transmitted frequencies for each span.

Span [MHz]	Actually Transmitted Frequency [MHz] (40/X)
4.0-4.29	4
4.3-4.79	4.44
4.8-5.39	5
5.4-6.19	5.71
6.2-7.29	6.66
7.3-8.89	8
8.9-10	10

5 Discussion

5.1 Sonix RP

While the probe should be replaced/repaired eventually, it still works fine for the procedure since there is no real use for the entire length of the transducer anyway. If patient trials are successful, the development of a customized probe might be the next step.

When looking at the pulses more closely during the hydrophone test, it was found that the energy, and frequencies of the pulses did not appear to change between all frequencies. Merely 7 different pulses were recorded in the span from 4-10 MHz. Since the instructions from Ultrasonix implied that it was possible to change frequencies on a Hz scale, this seemed unlikely. However, upon contacting the manufacturer a second time concerning this issue, we were informed of the real case. As stated in the results, the Sonix RP we used can only emit frequencies of 40/X MHz. This limitation had staggering implications for the master's thesis as a whole. At first, the intention was to check the intensity every 100kHz which would have resulted in 61 different intensities and quite a high certainty. Instead, it is unlikely that we can find the truly best frequency when there are only 7 to choose from.

Information regarding the frequency limitations should be available to users, as other experiments using the Sonix RP machines and software may require quite specific frequencies. As this is not possible, the implications may be that studies have published incorrect results. Since this limitation was discovered during very extensive tests checking different frequencies, it is unlikely that others have found, or even thought of the possibility, that this limitation exists.

5.2 Transducer Glitch

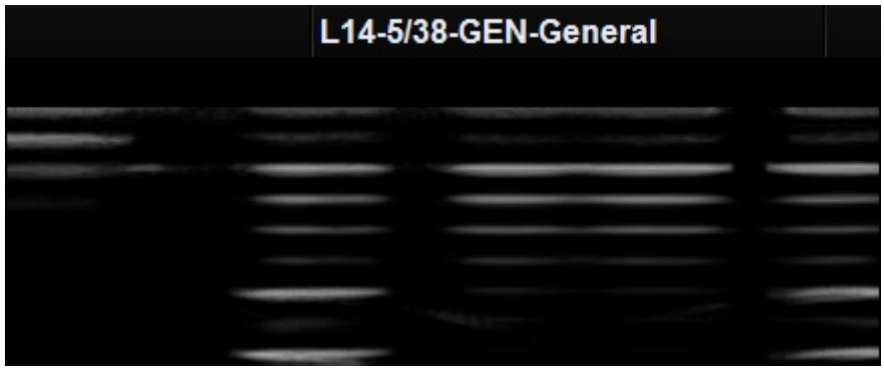


Figure 22: Cropped screenshot of the Exam software, note the extra black vertical area to the right

Ultrasonix's Exam software enables the user to disable individual transducer elements, which has been done in the figure 22 above. Ten elements in a row have been turned off to simulate the elements being dead.

The difference might not be clear in the printed version but the non-simulated areas are more diffuse, particularly the topmost horizontal line. It is still possible to discern a line in the middle and left black areas, whereas the right area is entirely black. Also, the shape is different. The simulated area is hourglass shaped while the others are more diffuse.

It is therefore likely that overheating occurred and that the glue melted, which holds the silicon layer in place (see figure 3). The glue in question is between the lens and matching layer. The energy absorbed in this layer was too high for extended use, perhaps in addition to the transducer being old and the glue subject to some dehydration. When the PRF was increased to 5kHz: this problem was no longer present, and the transducer is able to emit these frequencies and amplitude at extended periods of time without loss of function.

5.3 Programming

As we were both new to Qt Creator, and had not been working with C++ for a considerable amount of time, it took us a while to get Qt Creator to work together with Ulterius. Fixing include directories and libraries was difficult at first, as the documentation did not specify how

to do it correctly. Some '.dll' files were also needed in the build directory when the software was compiled. If the '.dll's were not there a cryptic "The program has finished unexpectedly." message would be all that was posted. Another initial unforeseen issue was that the Sonix RP had to be in Reasearch Mode to be allow remote connections from Ulterius.

It seemed as if every time anything new was being introduced into the program, a different error occurred. This led to hours upon hours of searching through forums for not only similar problems, but one that offered a solution let alone one that worked. One could almost argue that the developers need to hire inexperienced testers that try to follow their installation steps in order to understand how dreadfully non self-explanatory the steps are. In our searches we found that close to all explanations offered as solutions to errors and/or installation steps are incomplete and/or inaccurate. Despite it being true, admitting that approximately 50 percent of the time spent working, excluding this report, on this master's thesis, consisted of trying to get programs, libraries, and compilers to function properly, is almost embarrassing.

Even a seemingly simple task of adding a library in order to plot a simple graph can cause significant mental distress and anguish due to the installation steps being incomplete and non functioning. Hence we learned that one should never underestimate how much time might be needed for a given task, no matter how trivial it might appear.

In hindsight, it would probably have been a good idea to have used a version control system, like git, for this master's thesis. A version control system keeps track of the changes you make to your project. If you notice that you introduced a serious bug in your program you can just roll back to an earlier version with a version control system. It is also possible to fork your program, that is to make a new copy of the program, for testing purposes. Copying the folders manually worked well for a while, but when other things like the measurements or this report was the focus it got confusing which folder kept the newest version of the program.

The physics of the master's thesis should probably have been tested first. We first spent a few weeks fixing the Sonix RP and then started programming. What we probably should have done was to start by trying to induce acoustic streaming in milk. Simply assuming it would work was risky, considering that we spent quite a bit of time on first fixing the Sonix RP and then writing the program.

While finishing up the program- we found that disconnecting and

reconnecting to the Sonix RP caused our program to crash. We first though this was due to our own code, and so we tried creating scaled down versions of it to see where things went wrong. When these simple programs reproduced the same error, we decided to once again contact Ultrasonix's support forum, as we had done many times during the coding process. We were offered this explanation:

> Ulterius 5.7 has several known issues when streaming data. Access violations occur randomly depending on the sequence of actions performed. With the console demo I ran 10 tests [connect, set capture flag to BPost32, disconnect]. I got a crash in ntdll on the first try, then a few consecutive tests went fine, and then I got an access violation in utx_utils. These crashes are due to some race conditions, lack of thread safety and buffer management issues.
>
> The upcoming Ulterius fixes these issues. Fixing Ulterius 5.7 would require too much re-engineering so there is no plan to do it. I suggest that you call setDataToAcquire(0) before disconnecting. It seems to ease the issue but crashes may still occur.

It was surprising that such an established developer still had crashes in an application used in medical diagnosis. While the fault lies within 'Research Mode' (during which they renounce themselves of all responsibility for the machine's safety) it is still surprising.

As the response from Ultrasonix indicates, the exam software has several known issues. Since random occurring errors has forced us to search for faults in our own code, a simple post on their support forum of the known errors would have helped. The crashes that occur do not impose any danger to patients, or loss of data, but when using a program clinically, stability is important. If the program does not crash or malfunction it instils confidence to both patient and user. The hope is that future versions of the exam software handles their race conditions and Snotalyzer can be successfully updated and finally be really stable.

5.4 Safety

As can be seen in the figures on chapter in section 4.3.2 almost all of the measurements with a PRF of 5kHz were above FDA's limit, but during the measurements with the scale there was no measurable output for

5kHz. Only with 12.5kHz did we get a reading at all on the scale. The hydrophone has not been calibrated for a long time. Thus, our margin of error is quite large, but the measurement was mainly to get ballpark figures.

If a PRF of about 3.3kHz is used, there is little-to-no chance to ever exceed the safety limits set by the FDA. It has come to our attention that a small amount of heating of the sinus cavity might actually be a good thing for the patient suffering from rhinosinusitis, and so- this should be taken into consideration upon evaluating a patient study [16]

As the worst case scenario should always be considered when using equipment on patient, we always tried to find the absolute maximum values possible. The most extensive tests, using a PRF of 5kHz, was used to find out what the PRF should be in order to stay well below the safety limits. This was possible due to the total amount of energy being directly proportional to the PRF.

5.5 Milk Trials

During the milk trials, it was found that it is possible to induce detectable streaming inside the acrylic glass container even with a low PRF. The likelihood of the energy induced would be enough for actual patients is of course decreasing with lowering the PRF. So what should be sought after is the maximum PRF while under the safety limits set by the FDA. It should be noted that with increasing PRF, the minimum velocity that can be detected increases as well and thus it is not certain that increasing the PRF the answer. The results were promising for a possible future patient study using the program.

6 Conclusion

Using the Sonix RP as a base for this procedure is not completely optimal. It lacks the ability to go through enough different frequencies in order to obtain a certain point for optimum penetration through the front wall. However, the results from our phantom with milk suggests that it should be good enough to both induce and detect acoustic streaming in the low-viscosity fluid. And so it should not be completely dismissed.

A PRF of 3.3kHz should be used in order to be well below the safety limits set by the FDA. Whether or not this is enough to induce detectable streaming in all sorts of the fluid caused by bacteria is yet to be seen. It is, however, quite definitely not enough to induce anything detectable in the high viscosity fluid. Hence, it might be used as an initial guide at least. If there is a constant flow generated, antibiotics ought not be needed. This statement, however, is merely a speculation.

References

[1] P. Hoskins, A. Thrush, K, Martin, T. Whittingam, *Diagnostic Ultrasound Physics and Equipment,* Greenwith Medical Media Limited*, 2003.*

[2] Szabo, Thomas L. *12 - Nonlinear acoustics and imaging". Diagnostic Ultrasound Imaging,* Burlington: Academic Press, 2004. *http://www.sciencedirect.com/science/article/pii/B978012680145350013X.*

[3] Whittingham, T.A. *Medical diagnostic applications and sources.* Progress in Biophysics and Molecular Biology 93*, num. 1–3 (January 2007): 84–110.*

[4] P. Sahlstrand Johnson, *On Health-Relatade Quality of Life and Diagnostic Improvements in Rhinosinusitis,* Lund University - Faculty of Medicine*, 2011*

[5] W.L. Nyborg *Acoustic Streaming Near a Boundary,* Journal of the Acoustical Society of America*, May 30, 1958.*

[6] Nokia Corp. *Qt Creator,* NOKIA Corporation*, 2012.*

[7] DerManu, *Qt Plotting Widget, March 2012. http://www.workslikeclockwork.com/index.php/components/qt-plotting-widget/*

[8] C. Sturesson *Medical Laser-Induced Thermotherapy - Models and Applications,*Lund Reports on Atomic Physics*, 1998.*

[9] B. Stroustrup *The C++ Programming Language, 3rd Edition,* AT&T Labs in Murray Hill, New Jersey*, June 1997.*

[10] Barnett, Stanley B, Gail R Ter Haar, Marvin C Ziskin, Hans-Dieter Rott, Francis A Duck, and Kazuo Maeda. *International recommendations and guidelines for the safe use of diagnostic ultrasound in medicine.* Ultrasound in Medicine & Biology 26, num. 3 *(Mars 2000): 355–366.*

[11] J. A. Jensen. *Estimation of Blood Velocities using Ultrasound.* Campbridge University Press*, 1996.*

[12] F A Duck, G ter Haar. *The Safe Use of Ultrasound in Medical Diagnosis.* Brittish Institute of Radiology, London, UK*, 2000.*

[13] Jonathan Ophir, P.A. Narayana. *Effect of Half-Wavelength Membranes on the Axial Resolution of Real-Time Ultrasonic Scanners.* Department of Radiology, The University of Texas Medical School, Houston, Texas*, June 1983.*

[14] Roy C. Preston. *Output Measurements for Medical Ultrasound.* Springer-Verlag, Berlin*, 1991*

[15] Kathryn R. Nightingale, Phyllis J. Kornguth and Gregg E. Trahey. *The Use of Acoustic Streaming in Breast Lesion Diagnosis: A Clinical Study.* Ultrasound in Med. & Biol., Vol. 25, No. 1*, USA 1998*

[16] D. Young, R. Morton, J. Bartley. *Therapeutic ultrasound as treatment for chronic rhinosinusitis: Preliminary observations.* J Laryngnol Otol 2010;124:495-499.

A Snotalyzer User Guide

Turn on the Sonix RP machine. Once in Windows XP, start the Ultrasonix Exam Software. Make sure the Exam Software from Ultrasonix is running in Research Mode (the Research Mode button should be Orange on the console). If Snotalyzer is running on the Sonix RP itself, **minimize the Ultrasonix Exam Software**. This step is important, otherwise the Exam software will capture the mouse pointer during the Doppler Measurements.

Start Snotalyzer from the shortcut on the desktop or from the executable file in the Snotalyzer folder.

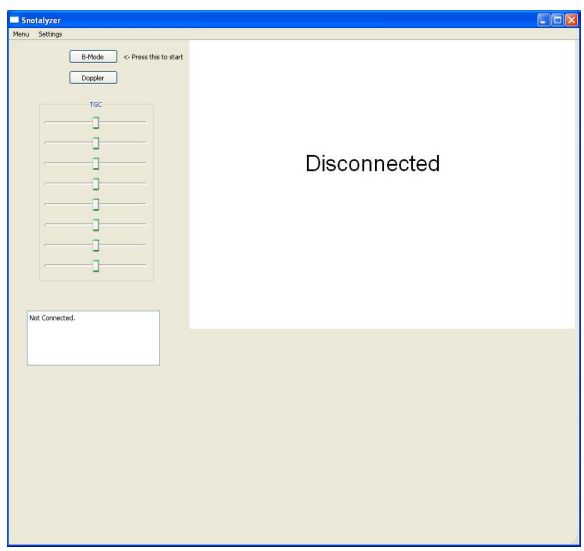


Figure 23: Startup. Press B-Mode button to connect. If fails, open settings window from the drop down menu.

To connect: Press B-Mode button. If this fails, make sure the Ultrasonix program is running in Research mode, and that the IP is set correctly by opening the 'Settings' window.

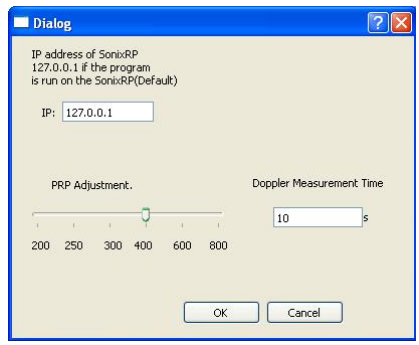


Figure 24: The 'Settings' options that appears when 'Settings' is pressed from the drop down menu.

Upon connecting, a pop-up appears where patient info should be entered. The 'First Name', 'Last Name' and 'Date' from this entry is used as the patient file name. These files are saved in the folder 'Patient Data' inside the 'Snotalyzer' directory.

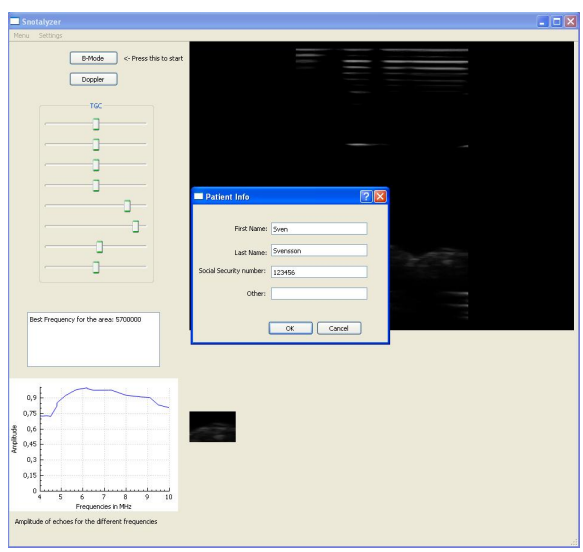


Figure 25: Pop-Up Patient Info. Enter the patient information here.

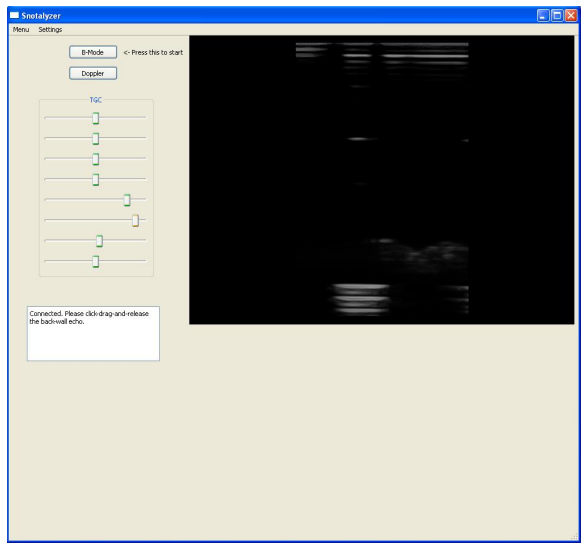


Figure 26: The B-Mode is running. Now is the time to use the Time Gain Control (TGC) sliders to improve the view of the back wall echo. Once satisfied, press-and-release diagonally over the area of interest.

B-Mode: Press-and-release diagonally over the back wall echo. The selection made is shown in the smaller display below. If unsatisfied, simply press-and-release a new one.

Once selection is made: Keep the transducer completely still until the optimum frequency is found.

Optimum Frequency Found: If satisfied with the graph showing the amplitudes of the returning echoes from all frequencies, press Doppler.

Doppler: Keep the transducer still and let it measure the acoustic streaming induced. Default time: 1+10 seconds. The first second is a simple calibration for the Sonix RP machine, the 10 following are the time of measure.

Pop-Up Measure Again: Press 'Yes' and go through another Doppler measurement. 'No' to finish. If finished: The program goes back into B-Mode.

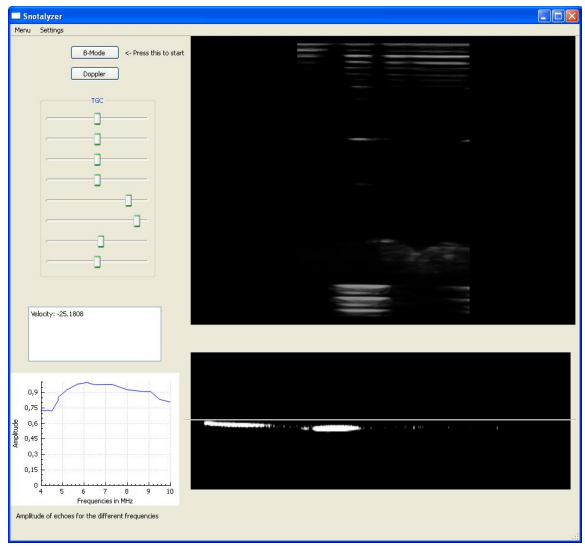


Figure 27: The Doppler runs for 10 seconds. After this time, the highest velocity found is displayed and written to the patient file.

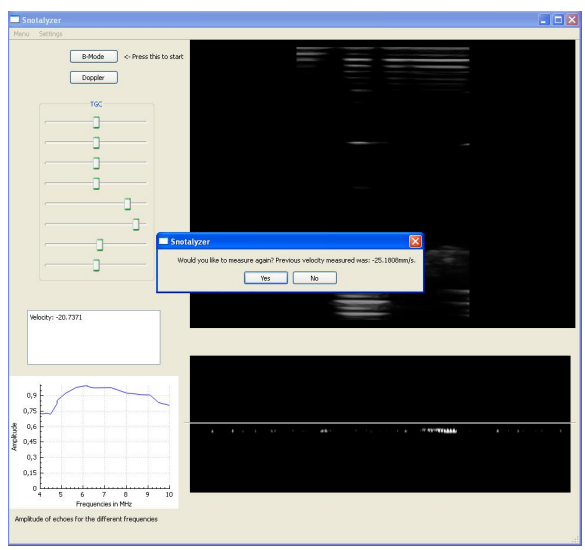


Figure 28: The pop-up that appears when the Doppler measurement is finished.

To switch to another Patient, press the New Patient button from the drop down Menu.

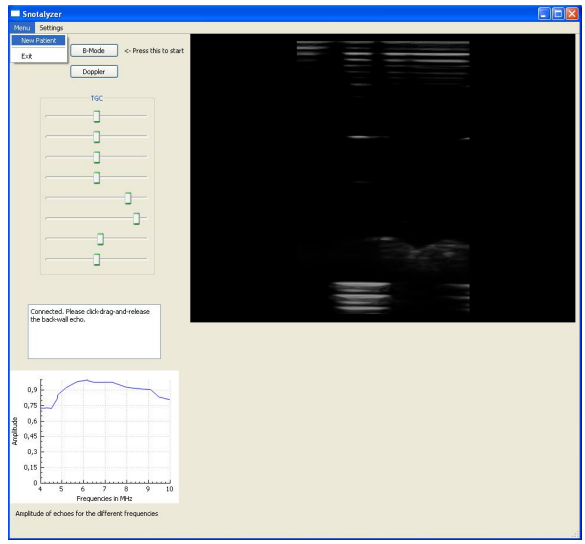


Figure 29: New Patient from the drop-down Menu.

If the program crashes for any reason, restart Snotalyzer and the Ultrasonix Exam Software.

Patient information files are saved in Snotalyzer/Patient Data.

B Source Code

B.1 main.cpp

```
#include <QtGui/QApplication>
#include "mainwindow.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

B.2 mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "keypresswidget.h"
#include "worker.h"
#include <QtGui>
#include <QtCore>
//#include <QTimer>
#include "qcustomplot.h"
#include "patientdialog.h"
#include "settingsdialog.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    static void setSelection1(int x,int y);
    static void setSelection2(int x,int y);
    void drawGraph(QVector<double> y, QVector<double> x, int
        start, int stepsize, int stop, int maxVal);
```

```
public slots:
    void measureAgain(double velocity);

private slots:
    //void on_Disconnect_clicked();
    void on_Connect_clicked();
    void on_Doppler_clicked();
    void updateBDisplay(QImage* ptr);
    void updateSpect(QImage* ptr, double velocity);

    void on_actionNew_Patient_triggered();

    void on_actionExit_triggered();

    void on_actionAbout_triggered();

    void on_actionSettings_triggered();

private:
   // void on_prpslider_sliderMoved(int position);
    bool MainWindow::fileWrite(QVector<double> plotValues,
        QVector<double> frequencies, int start, int stepSize,
         int endVal, int maxVal, int bestFreq);
    void updateTGC();
    Ui::MainWindow *ui;
    QPixmap *image;
    static keyPressWidget* bDisplay;
    static keyPressWidget* selected;
    QString get_date();
    QString FirstName;
    QString LastName;
    QString PNumber;
    QString otherInfo;

//    static SettingsDialog setDiag;
//    static PatientDialog pDialog;

};

#endif // MAINWINDOW_H
```

B.3 mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "ushandler.h"
```

```
#include <cmath>
#include <cstdlib>
#include <time.h>
/*
#include <QDebug>
#include <QDesktopWidget>
#include <QMessageBox>
#include <QDateTime>
#include <QString>
#include <QFile>
#include <QCoreApplication>
#include <QTextStream>
*/

#define MAX_DATE 12
//USHandler handler;
USHandler* hptr;

keyPressWidget* MainWindow::bDisplay;
keyPressWidget* MainWindow::selected;
MainWindow* mptr;
Worker *wThread;

//Dialogs



//Coordinates for selected to view area
int x1,x2,y11,y2;
bool selectionMade;
bool written;


MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    /* Setting up the GUI and creating other class objects
       */
    mptr = this;
    ui->setupUi(mptr);
    mptr->removeToolBar(ui->mainToolBar);
    mptr->setWindowTitle("Snotalyzer");
    mptr->ui->line->setVisible(false);
    hptr= new USHandler();

    written=false;
```

```
    mptr->ui->customPlot->setVisible(false);
    mptr->ui->echoSumLabel->setVisible(false);

    selectionMade=false;

    MainWindow::bDisplay = new keyPressWidget(bDisplay);
    MainWindow::selected = new keyPressWidget(selected);

    ui->vlayout->addWidget(bDisplay);
    ui->MaxValText->setText("Not Connected.");
    ui->keyP->addWidget(selected);

    QImage frame(":/images/disconnected.jpg");
    bDisplay->setPixmap(QPixmap::fromImage(frame));
    bDisplay->setEnabled(true);

    wThread = new Worker(mptr,hptr);

    QObject::connect(hptr,SIGNAL(newBFrame()),wThread,SLOT(
        BWork()));

    //QObject::connect(hptr,SIGNAL(newBDisplay(QImage)),mptr
        ,SLOT(updateBDisplay(QImage)));
    QObject::connect(hptr,SIGNAL(newBDisplay(QImage*)),mptr,
        SLOT(updateBDisplay(QImage*)));

    QObject::connect(hptr,SIGNAL(newDoppler()),wThread,SLOT(
        dopplerWork()));

    //QObject::connect(hptr,SIGNAL(newSpect(QImage, double))
        ,mptr,SLOT(updateSpect(QImage, double)));
    QObject::connect(hptr,SIGNAL(newSpect(QImage*, double)),
        mptr,SLOT(updateSpect(QImage*, double)));

    QObject::connect(hptr,SIGNAL(measureFinished(double)),
        mptr,SLOT(measureAgain(double)));

    FirstName = "FirstName";
    LastName="LastName";

    //MainWindow::setDiag.setModal(true);
   // MainWindow::pDialog.setModal(true);
    //MainWindow::pDialog.setWindowTitle("Patient Info");
}

MainWindow::~MainWindow()
{
```

```
    hptr->Disconnect();
    delete hptr;
    delete selected;
    delete bDisplay;
    delete wThread;
    delete ui;
}

/* Function sets up the connection to the SonixRP */
void MainWindow::on_Connect_clicked()
{
    if(hptr->Connect())
    {
        mptr->ui->MaxValText->setText("Connected.");
        if(hptr->BMode()){
            mptr->selected->setVisible(false);
            mptr->ui->MaxValText->setText("Connected. Please
                click-drag-and-release the back-wall echo.")
                ;
            ui->line->setVisible(false);
            selectionMade=false;
            return;
        }
    }
    mptr->ui->MaxValText->setText("Failed to connect.");
}

/* Switch to PWD */
void MainWindow::on_Doppler_clicked()
{
    hptr->Doppler();
    ui->line->setVisible(true);
    mptr->selected->setVisible(true);
    mptr->ui->MaxValText->setText("PW Doppler.");
}

/* Update the images on QLabels
   and display frequency progress on a
   QTextEdit */
void MainWindow::updateBDisplay(QImage *ptr)
{
    if(selectionMade){//visa selected area ocksa
        selected->setPixmap(QPixmap::fromImage(hptr->
            selectedArea));
        if(hptr->bestFound){
            if(!written){
                ui->MaxValText->setText("Best Frequency for
```

```
                    the area: " + QString::number(hptr->
                        getbestFreq()));
                mptr->drawGraph(hptr->getSums(),hptr->
                    getFreqs(),hptr->getStartFreq(),hptr->
                    getStepSize(),hptr->getMaxFreq(),hptr->
                    getMaxVal());
                written=true;
            }
        }else{
            ui->MaxValText->setText("Calculating optimum
                frequency.\nCurrently at:"+QString::number(
                hptr->getCurrFreq()) + "\nBest Frequency so
                far:" + QString::number(hptr->getbestFreq())
                );
            written=false;
        }
        //}
    }
    mptr->updateTGC();

    //bDisplay->setPixmap(QPixmap::fromImage(frame));
    bDisplay->setPixmap(QPixmap::fromImage(*(ptr)));
}

/* Update spectral image on QLabel */
void MainWindow::updateSpect(QImage *ptr, double velocity)
{
    selected->setPixmap(QPixmap::fromImage(*(ptr)));
    ui->MaxValText->setText("Velocity: " + QString::number(
        velocity));
}

/* Function to set one corner of the rectangular area */
void MainWindow::setSelection1(int x,int y){
    x1=x;
    y11=y;

    x2=x1+1;
    y2=y11+1;
}

/* Function to set the other corner of the rectangular area,
   Also switches coords if needed (Click, draw left or Click
       , draw right)*/
void MainWindow::setSelection2(int x,int y){
    if(x>x1){x2=x;}else{x2=x1;x1=x;}
    if(y>y11){y2=y;}else{y2=y11;y11=y;}
```

```
    selectionMade=true;
    hptr->setSelArea(y11,y2,x1,x2);

    printf("X1: %d, Y1: %d, X2: %d, Y2: %d", x1, y11, x2, y2
        );
    fflush(stdout);

    mptr->ui->MaxValText->setText("Area Selected, 
        calculating optimum frequency.");
    mptr->selected->setVisible(true);
}

// Set the Time Gain Control
void MainWindow::updateTGC(){
    hptr->updateTGC(ui->hs1->value(),
            ui->hs2->value(),
            ui->hs3->value(),
            ui->hs4->value(),
            ui->hs5->value(),
            ui->hs6->value(),
            ui->hs7->value(),
            ui->hs8->value());
}


/* After the frequency scan is complete, this function is
    run to draw a graph of the normalized frequency response
  of the selected area */
void MainWindow::drawGraph(QVector<double> y, QVector<double
    > x, int start, int stepsize, int stop,int maxVal)
{
    mptr->ui->customPlot->setVisible(true);
    mptr->ui->echoSumLabel->setVisible(true);
    int s=(stop-start)/stepsize -1;
    printf("Size: %d. y1: %f. y2: %f. Maxval: %d", s , y[4],
        y[8], maxVal);
    fflush(stdout);
     // initialize with entries 0..100
    s=sizeof(x)/sizeof(x[0]);
   for (int i=0; i<21; ++i)
   {
        x[i] = x[i]/1000000;
        y[i]=y[i]/maxVal; //take away the first number and
            count up to the second last
   }
```

```
    // create graph and assign data to it:
    mptr->ui->customPlot->addGraph();
    printf("y1: %f, y2: %f, x1: %f, x2: %f", y[1],y[2], x
        [1],x[2]);
    fflush(stdout);
    // QCPDataMap::
    mptr->ui->customPlot->graph(0)->setData(x,y);
    // give the axis some labels:
    mptr->ui->customPlot->xAxis->setLabel("Frequencies in
        MHz");
    mptr->ui->customPlot->yAxis->setLabel("Amplitude");
    // set axis ranges, so we see all data:
    mptr->ui->customPlot->xAxis->setRange(4, 10);
    mptr->ui->customPlot->yAxis->setRange(0, 1.01);//maxVal
        +100);

    mptr->ui->customPlot->replot();

    fileWrite(y,x,start,stepsize,stop, maxVal, hptr->
        getbestFreq());
}

/* Funciton to save Patient data to disc */
bool MainWindow::fileWrite(QVector<double> plotValues,
    QVector<double> frequencies, int start, int stepSize, int
     endVal, int maxVal, int bestFreq){
    time_t rawtime;

    time ( &rawtime );
    //printf ( "The current local time is: %s", ctime (&
        rawtime) );

    //fflush(stdout);
    if(FirstName=="FirstName"){
        mptr->on_actionNew_Patient_triggered();
    }
    //QString filnamn=ctime(&rawtime);
    if(!QDir("Patient Data").exists()){
        QDir().mkdir("Patient Data");
    }
    QFile file("Patient Data/"+FirstName + "_"+ LastName +"_
        "+PNumber+"_at_" +get_date() + ".txt");
    file.open(QIODevice::Append | QIODevice::Text);
    QTextStream out(&file);
    out <<"\n
        ////////////////////////////////////////////////////////////
        "<< " \nName: " << FirstName <<", "<< LastName <<". "
```

```
        Id number: "<<PNumber<<".\nOther info: "<<otherInfo<<
        ".\nThe date and time is: "<< ctime(&rawtime) <<
        "..................................................
            "<<"\nFrequencies Used: \n";
    for(int k=0;k<frequencies.size();k++)
    {
        out << frequencies[k]<<", "; //Outputs array to the
            text file
    }
    out <<".\nIntensity values: \n";
    for(int k=0;k<plotValues.size();k++)
    {
        out << plotValues[k]<<", "; //Outputs array to the
            text file
    }
    out << ".\n End of plotValues. MaxVal: "<<maxVal<<". 
        Best Frequency was: "<<bestFreq<< "Hz.  \n
        ..................................................";
    file.close();
    return true;
}

// Function to get the date of examination
QString MainWindow::get_date()
{
    time_t now;
    char the_date[MAX_DATE];

    the_date[0] = '\0';

    now = time(NULL);

    if (now != -1)
    {
        strftime(the_date, MAX_DATE, "%d_%m_%Y", gmtime(&now
            ));
    }else{
        return "NoDate";
    }
    return QString(the_date);
}

/* Function that allows repeated measurements in Doppler
    mode */
void MainWindow::measureAgain(double velocity){
    QMessageBox again;
    again.setText("Would you like to measure again? Previous
```

```
        velocity measured was: " + QString::number(velocity)
        +"mm/s.");

    if(!QDir("Patient Data").exists()){
        QDir().mkdir("Patient Data");
    }
    QFile file("Patient Data/"+FirstName + "_"+ LastName +"_
        "+PNumber+"_at_" +get_date() + ".txt");
    file.open(QIODevice::Append | QIODevice::Text);
    QTextStream out(&file);
    out<<"\nVelocity found: " << velocity<<".";
    file.close();

    again.setStandardButtons(QMessageBox::Yes | QMessageBox
        ::No);
    if(again.exec()==QMessageBox::Yes){ //Yes
        hptr->Doppler();
    }else{//No, go back to B-Mode
        mptr->on_Connect_clicked();
    }
}

/* Set up the PatientDialog and pop up the Dialog to
  retreive patient data */
void MainWindow::on_actionNew_Patient_triggered()
{
    PatientDialog pDialog;
    pDialog.setModal(true);
    pDialog.setWindowTitle("Patient Info");
    if (pDialog.exec()==QDialog::Accepted) {
        FirstName = pDialog.getItem(0);
        LastName=pDialog.getItem(1);
        PNumber =pDialog.getItem(2);
        otherInfo=pDialog.getItem(3);
    }
}

//Quit
void MainWindow::on_actionExit_triggered()
{
    //hptr->Disconnect();
    exit(0);
}

void MainWindow::on_actionAbout_triggered()
{
    QMessageBox about;
```

```
    about.setText("This program was written by Daniel
        Andersson & Peter Zanden. For use on sinusitis
        patients. It was done by request from Tomas Jansson
        and Pernilla Sahlstrand Johnson");
    about.exec();
}

/* Pop up the SettingsDialog, allows change of IP, pulse
   frequency and measurement time*/
void MainWindow::on_actionSettings_triggered()
{
        SettingsDialog setDiag;
        setDiag.setModal(true);
        if (setDiag.exec()==QDialog::Accepted){
            QString address=setDiag.getIP();
            hptr->setIP(address);
            hptr->setPRP(setDiag.getPRP());
            hptr->setTime(setDiag.getTime());
        }
}
```

B.4 mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>947</width>
    <height>859</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralWidget">
   <widget class="QWidget" name="verticalLayoutWidget">
    <property name="geometry">
     <rect>
      <x>300</x>
      <y>0</y>
      <width>641</width>
      <height>481</height>
     </rect>
    </property>
```

```
 <layout class="QVBoxLayout" name="vlayout">
  <property name="spacing">
   <number>6</number>
  </property>
  <property name="sizeConstraint">
   <enum>QLayout::SetFixedSize</enum>
  </property>
 </layout>
</widget>
<widget class="QWidget" name="verticalLayoutWidget_2">
 <property name="geometry">
  <rect>
   <x>100</x>
   <y>10</y>
   <width>81</width>
   <height>71</height>
  </rect>
 </property>
 <layout class="QVBoxLayout" name="verticalLayout">
  <item>
   <widget class="QPushButton" name="Connect">
    <property name="text">
     <string>B-Mode</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QPushButton" name="Doppler">
    <property name="text">
     <string>Doppler</string>
    </property>
   </widget>
  </item>
 </layout>
</widget>
<widget class="QWidget" name="verticalLayoutWidget_4">
 <property name="geometry">
  <rect>
   <x>300</x>
   <y>490</y>
   <width>641</width>
   <height>301</height>
  </rect>
 </property>
 <layout class="QVBoxLayout" name="keyP"/>
</widget>
<widget class="QTextEdit" name="MaxValText">
```

```
 <property name="geometry">
  <rect>
   <x>30</x>
   <y>450</y>
   <width>221</width>
   <height>91</height>
  </rect>
 </property>
</widget>
<widget class="QGroupBox" name="groupBox">
 <property name="geometry">
  <rect>
   <x>50</x>
   <y>100</y>
   <width>191</width>
   <height>301</height>
  </rect>
 </property>
 <property name="title">
  <string>TGC</string>
 </property>
 <property name="alignment">
  <set>Qt::AlignCenter</set>
 </property>
 <widget class="QWidget" name="verticalLayoutWidget_3">
  <property name="geometry">
   <rect>
    <x>9</x>
    <y>20</y>
    <width>171</width>
    <height>281</height>
   </rect>
  </property>
  <layout class="QVBoxLayout" name="verticalLayout_2">
   <item>
    <widget class="QSlider" name="hs1">
     <property name="maximum">
      <number>255</number>
     </property>
     <property name="value">
      <number>127</number>
     </property>
     <property name="orientation">
      <enum>Qt::Horizontal</enum>
     </property>
    </widget>
   </item>
```

```
<item>
 <widget class="QSlider" name="hs2">
  <property name="maximum">
   <number>255</number>
  </property>
  <property name="value">
   <number>127</number>
  </property>
  <property name="orientation">
   <enum>Qt::Horizontal</enum>
  </property>
 </widget>
</item>
<item>
 <widget class="QSlider" name="hs3">
  <property name="maximum">
   <number>255</number>
  </property>
  <property name="value">
   <number>127</number>
  </property>
  <property name="orientation">
   <enum>Qt::Horizontal</enum>
  </property>
 </widget>
</item>
<item>
 <widget class="QSlider" name="hs4">
  <property name="maximum">
   <number>255</number>
  </property>
  <property name="value">
   <number>127</number>
  </property>
  <property name="orientation">
   <enum>Qt::Horizontal</enum>
  </property>
 </widget>
</item>
<item>
 <widget class="QSlider" name="hs5">
  <property name="maximum">
   <number>255</number>
  </property>
  <property name="value">
   <number>127</number>
  </property>
```

```
      <property name="orientation">
       <enum>Qt::Horizontal</enum>
      </property>
     </widget>
    </item>
    <item>
     <widget class="QSlider" name="hs6">
      <property name="maximum">
       <number>255</number>
      </property>
      <property name="value">
       <number>127</number>
      </property>
      <property name="orientation">
       <enum>Qt::Horizontal</enum>
      </property>
     </widget>
    </item>
    <item>
     <widget class="QSlider" name="hs7">
      <property name="maximum">
       <number>255</number>
      </property>
      <property name="value">
       <number>127</number>
      </property>
      <property name="orientation">
       <enum>Qt::Horizontal</enum>
      </property>
     </widget>
    </item>
    <item>
     <widget class="QSlider" name="hs8">
      <property name="maximum">
       <number>255</number>
      </property>
      <property name="value">
       <number>127</number>
      </property>
      <property name="orientation">
       <enum>Qt::Horizontal</enum>
      </property>
     </widget>
    </item>
   </layout>
  </widget>
 </widget>
```

```
<widget class="QLabel" name="label_3">
 <property name="geometry">
  <rect>
   <x>190</x>
   <y>20</y>
   <width>111</width>
   <height>16</height>
  </rect>
 </property>
 <property name="text">
  <string>&lt;- Press this to start</string>
 </property>
</widget>
<widget class="Line" name="line">
 <property name="geometry">
  <rect>
   <x>290</x>
   <y>630</y>
   <width>651</width>
   <height>16</height>
  </rect>
 </property>
 <property name="orientation">
  <enum>Qt::Horizontal</enum>
 </property>
</widget>
<widget class="QCustomPlot" name="customPlot" native="
    true">
 <property name="geometry">
  <rect>
   <x>0</x>
   <y>560</y>
   <width>281</width>
   <height>221</height>
  </rect>
 </property>
</widget>
<widget class="QLabel" name="echoSumLabel">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>790</y>
   <width>251</width>
   <height>16</height>
  </rect>
 </property>
 <property name="text">
```

```
    <string>Amplitude of echoes for the different
        frequencies</string>
   </property>
  </widget>
  <zorder>groupBox</zorder>
  <zorder>verticalLayoutWidget</zorder>
  <zorder>verticalLayoutWidget_2</zorder>
  <zorder>verticalLayoutWidget_4</zorder>
  <zorder>MaxValText</zorder>
  <zorder>label_3</zorder>
  <zorder>line</zorder>
  <zorder>customPlot</zorder>
  <zorder>echoSumLabel</zorder>
 </widget>
 <widget class="QMenuBar" name="menuBar">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>947</width>
    <height>21</height>
   </rect>
  </property>
  <widget class="QMenu" name="menuMenu">
   <property name="title">
    <string>Menu</string>
   </property>
   <addaction name="actionNew_Patient"/>
   <addaction name="separator"/>
   <addaction name="actionExit"/>
  </widget>
  <widget class="QMenu" name="menuSettings">
   <property name="title">
    <string>Settings</string>
   </property>
   <addaction name="actionSettings"/>
   <addaction name="separator"/>
   <addaction name="actionAbout"/>
  </widget>
  <addaction name="menuMenu"/>
  <addaction name="menuSettings"/>
 </widget>
 <widget class="QToolBar" name="mainToolBar">
  <attribute name="toolBarArea">
   <enum>TopToolBarArea</enum>
  </attribute>
  <attribute name="toolBarBreak">
```

```
      <bool>false</bool>
     </attribute>
    </widget>
    <widget class="QStatusBar" name="statusBar"/>
    <action name="actionNew_Patient">
     <property name="text">
      <string>New Patient</string>
     </property>
    </action>
    <action name="actionExit">
     <property name="text">
      <string>Exit</string>
     </property>
    </action>
    <action name="actionSettings">
     <property name="text">
      <string>Settings</string>
     </property>
    </action>
    <action name="actionAbout">
     <property name="text">
      <string>About</string>
     </property>
    </action>
   </widget>
   <layoutdefault spacing="6" margin="11"/>
   <customwidgets>
    <customwidget>
     <class>QCustomPlot</class>
     <extends>QWidget</extends>
     <header>../qcustomplot.h</header>
     <container>1</container>
    </customwidget>
   </customwidgets>
   <resources/>
   <connections/>
  </ui>
```

B.5 ushandler.h

```
#ifndef USHANDLER_H
#define USHANDLER_H
#include "ulterius.h"
#include <stdio.h>
#include <QtCore>
#include <QtGui>
#include <QTimer>
#include <QWaitCondition>
```

```
class USHandler : public QObject
{
    Q_OBJECT

public:
    USHandler();
    ~USHandler();

    static bool Connect();
    static void Disconnect();
    static bool BMode();
    static void setSelArea(int selTop, int selBot, int
        selLeft, int selRight);
    static bool isSel();
    static void updateBFrame();
    static void Doppler();
    static void updateDoppler();
    static QImage selectedArea;
    void setSelected();
    void updateTGC(int tgc1, int tgc2,int tgc3,int tgc4,int
        tgc5,int tgc6,int tgc7,int tgc8);
    void changePRP(int prpPos);

    QVector<double> getSums();
    QVector<double> getFreqs();
    int getStartFreq();
    int getMaxFreq();
    int getStepSize();
    int getMaxVal();
    int getCurrFreq();


    double getVelocity();
    int getbestFreq();

    static bool bestFound;

    static void setIP(QString address);
    void setPRP(int prp);
    void setTime(int time);


signals:
    void newBFrame();
    void newDoppler();
```

```
    void newBDisplay(QImage* ptr);
    void newSpect(QImage* ptr, double velocity);
    void measureFinished(double velocity);

private:
    static ulterius ult;
    static uchar* buffer;
    static uchar* bufferDoppler;
    //static QMutex* mutex;
    static void freqChanger(int selectionMax);
    static double adjustPRP();

    static QVector<double> selectionSums;
    //static bool localhost;

    static bool processBFrame(void* data, int type, int size
        , bool fromCine, int frameNo);
    static bool processDoppler(void* data, int type, int
        size, bool fromCine, int frameNo);


private slots:
    void stopDoppler();


};

#endif // USHANDLER_H
```

B.6 ushandler.cpp

```
#include "ushandler.h"

uchar* USHandler::buffer;
uchar* USHandler::bufferDoppler;
QVector<double> USHandler::selectionSums;

QImage frame;
QImage* imPtr= &frame;


ulterius USHandler::ult;
static int imSize= 640*480*4; //Dunno what the actual
    maxsize is supposed to be...
QMutex mutex;
static USHandler* usptr;
uDataDesc desc;
QVector<QRgb> v(256);
```

```
QImage USHandler::selectedArea;
bool USHandler::bestFound;

// Global image info
int wid=632;
int hei = 228;
int sumSpect [1000];
int baseline;
//int velocity;

//-------------------------------------------------

//Frequency and velocity variables
double maxVal;
int bestFreq;
int maxFreq;
int currFreq;
int startFreq;
int currMax;
int stepSize;
int spectSumCount;
double maxVelocity;
int maxSum;
int pulseRepeat;
                // Steps : 4-4.2                    4.3-4.7
                                          4.8-5.3
                                             5.4-6.1
                                             6.2-7.2
                                         7.3-8.8
                      8.9-10
int freqVect[] = {4000000, 4150000, 4290000, 4400000,
    4500000, 4790000, 4800000, 5200000, 5390000, 5400000,
    5700000, 6190000, 6200000, 6500000, 7200000, 7300000,
    8000000, 8700000, 9100000, 9500000, 10000000};
int freqCount;
int freqVectSize;

uTGC tgc;
int PRPsetting;
int measurementTime;


//Coordinates for selected to view area
int xx1,xx2,yy11,yy2;
bool selMade;

//Remote & localhost support
```

```
QString ipAddr;
bool localhost;


USHandler::USHandler()
{//Constructor. Initiates variables and sets their values.

    //A pointer to this is needed in some static functions.
    usptr=this;

    //Static buffer only needs to be initialized once.
    USHandler::buffer = (uchar *)malloc(imSize * sizeof(
        uchar));

    //Selection variables initialized.
    selMade=false;
    xx1=10;xx2=15;yy11=5;yy2=10;
    maxVal=0;

    //Frequencies are in Hz.
    maxFreq=10000000;
    startFreq=4000000;
    //bestFreq=startFreq;
    freqCount=0;
    freqVectSize=sizeof( freqVect ) / sizeof( freqVect[0] );
    bestFreq=freqVect[0];
    stepSize=1000000;
    //currFreq=startFreq;
    currFreq=freqVect[0];
    PRPsetting=400;
    measurementTime = 10;

    //Initializing the vector of doubles which will be used
        to plot the amplitudes of the different echoes.
    //selectionSums= QVector<double>((maxFreq-startFreq)/
        stepSize);
    selectionSums= QVector<double>(freqVectSize);
    //printf("size: %d.\n", sizeof( freqVect ) / sizeof(
        freqVect[0] ) );
    //fflush(stdout);
    bestFound=false;
    spectSumCount=0;

    //For remote och local use
    ipAddr="127.0.0.1"; // default localhost
    localhost=true;
```

```
    //Creating an RGB vector for doppler usage.
    for (int i = 0; i < 256; ++i)
    {
        v[i] = qRgb(i, i, i);
    }
}


USHandler::~USHandler()
{   //Deconstructor of USHandler.
   // if(ult.isConnected()){ult.disconnect();}
   free(buffer);
    free(bufferDoppler);
    //if(selectedSums){     delete selectionSums;
   /* if(sumSpect){     delete [] sumSpect;}
        if(sumSpect){delete sumSpect;}
    if(buffer){delete buffer;}
    if(bufferDoppler){delete bufferDoppler;}*/
}



bool USHandler::Connect()
{
    /*Connect functionality
      If one is already connected, nothing needs to be done.
      The probe is set to the first slot if the connectin is
          successful.
      */

    //boolean for the callbackfunctions
    if (!QString::compare(ipAddr,"127.0.0.1")) {
        localhost=true;
    } else {
        localhost=false;
    }

    //QString to ip
    QByteArray bArr = ipAddr.toLocal8Bit();
    const char *cStr = bArr.data();

    if(ult.isConnected()){return true;}
    //if(ult.connect("130.235.53.125"))
    if(ult.connect(cStr))
    {
```

```
        ult.selectProbe(0);
        return true;
    }
    else
        return false;
}

void USHandler::Disconnect()
{//If one simply wants to disconnect.
    ult.disconnect();
}

bool USHandler::BMode()
{
    /*Starting in B-mode (0).
      Callback function needs to be set.
      We know the depth is about 6 cm, and so 60 mm.
      Start with the best frequency found, this is the same
          as the starting one
      if no calculation has been made.
      */
    mutex.lock();
    ult.selectMode(0);
    ult.setCallback(&USHandler::processBFrame);
    ult.setDataToAcquire(udtBPost32);
    ult.setParamValue("b-depth",60);
    ult.setParamValue("b-freq",bestFreq);
    mutex.unlock();
    return true;

}

bool USHandler::processBFrame(void* data, int type, int size
    , bool fromCine, int frameNo){
    /* This function is being called as a callback. Each
        time the Sonix RP has created a new frame,
      this function is called.
    If the program runs as local host, we need to skip
        frames, otherwise the program can't keep up.
      Copys the data from the Sonix RP to 'buffer' and emits
          a signal that a new frame has been recieved.

      */
    if(data!=NULL && localhost && frameNo % 3==0 || data!=
        NULL && localhost==false){
        mutex.lock();
        memcpy(buffer,data,size);
```

```
        mutex.unlock();

        emit usptr->newBFrame();
        return true;
    }else {
        return false;
    }
}

bool USHandler::processDoppler(void* data, int type, int
    size, bool fromCine, int frameNo){
    /*First determining how one is connected to the machine.
      If the program is run at localhost, images are being
          send to rapidly,
      hence the % as modulo 3. So every third image is used.
      */

    if(data!=NULL && localhost && frameNo % 3==0 || data!=
        NULL && localhost==false){
        mutex.lock();
        memcpy(bufferDoppler,data,size);
        mutex.unlock();

        emit usptr->newDoppler();
        return true;
    }else {
        return false;
    }

}

void USHandler::updateBFrame()
{
    /* This function updates the image to that which has
       been sent from the UltraSonix RP
      A descriptor is used in order to determine the
          characteristics of the image recieved.
      A temporary image is created and '.bits()' forces a
          deep copy.
      Mutex is used to prevent the image from being changed
          while working.
      */
        mutex.lock();
        ult.getDataDescriptor(udtBPost32,desc);


        {
```

```
QImage tmpFrame(buffer, desc.w, desc.h, QImage::
    Format_RGB32);
frame = tmpFrame;
frame.bits(); // Should force frame to perform a
    deep copy of the buffer.
} // At the end of the block, tmpFrame dies and so
    the buffer can now be freed.




//
    ************************************************************

//Selection making if an area has been selected.
if(selMade){
    int w = xx2-xx1;
    int h=yy2-yy11;
    int selectiontop = yy11;
    int selectionbottom = yy2;
    int selectionleft=xx1;
    int selectionright=xx2;

    /*Calculates the sum of the combined pixel
        values, of the selected area.
    First, the selected area is extracted from the B
        -mode image and put into the 'selectedArea'.
    Second, 'selectedArea' is used to calculate the
        sum called currMax.
    */
    currMax=0;
    uchar currVal;
    selectedArea = frame.copy(selectionleft,
        selectiontop, selectionright - selectionleft,
         selectionbottom - selectiontop);
    for(int m = 0;m<h;++m){
        for(int n=0;n<w;++n){
            currVal = selectedArea.pixel(n,m);
            if(currVal>10){
                currMax+=currVal;
            }
        }
    }
    //When the calculation is done, freqChanger
        changes the transmitting frequency and stores
         the calculated sum.
    freqChanger(currMax);
```

```
        }

        //A pointer to the updated image is sent in a signal
            .
        emit usptr->newBDisplay(imPtr);
        mutex.unlock();

}

void USHandler::updateDoppler()
{
    /* Similar to the updateBFrame, this function handles a
       new Dopplerframe.
      A colortable is set, in case this is needed.
      Using the buffer sent by the UltraSonix RP, an image
         is created through the use of memcpy.
      This because the the image is created differently from
          the B-mode frame.
      */
        mutex.lock();
        USHandler::ult.getDataDescriptor(udtPWSpectrum,desc)
            ;

        QImage tempIm(QSize(desc.w, desc.h), QImage::
            Format_Indexed8);

        tempIm.setColorTable(v);//kanske inte maste satta

        // Copying from the buffer, line by line.
        for (int line = 0; line < tempIm.height(); ++line)
        {
            std::memcpy(tempIm.scanLine(line), bufferDoppler
                + line * tempIm.width(), tempIm.width());
        }

        //Creating a pointer to the image's position in the
            memory.
        QImage* tmpPtr = &tempIm;


        //
            ************************************************************

        /*Here starts the calculation of the velocity from
            the buffer data.
          The sumSpect vector is filled with zeros, and
```

```
    other needed variables are initiated.
  The double loop goes through

  */

    memset(sumSpect,0,1000);
    int currsum=0;
    double tomte=0;
    double velocity=0;

    for(int row = 0;row<desc.h;++row){
        for(int column=20;column<desc.w;++column){

            currsum = qRed(tempIm.pixel(column,row))
                ; //detta maste ersattas.
            // c=c+1;

            if(currsum>10){
                //c++;
                sumSpect[row]+=currsum;

                //if(c>velocity){velocity=c;}//
                //      if(static_cast<double>(m)>
                    velocity){velocity=static_cast<
                    double>(m);}

            }
        }
        tomte=(double)sumSpect[row]/(double)desc.w;
        if(tomte>5){
            if((double)row>velocity){velocity=(
                double)row;}

        }
    }



/*
for (int rows=0;rows<desc.h;rows++){
    for (int columns =0;columns<desc.w;columns++) {
        currsum = static_cast<double>(*(
            bufferDoppler + (rows*desc.w+columns)*
            sizeof(uchar)));
        if (currsum>10) {
            sumSpect[rows]+=currsum;
```

```
                }
            }


            tomte=sumSpect[rows]/(double)desc.w;
            if(tomte>5){
                if((double)rows>velocity){velocity=(double)
                    rows;}

            }
        }*/


        //These are calculations to adjust for prp and image
            properties.
        double noll=0;
        double hojd = (double)desc.h;
        if(velocity!=noll){
            velocity=hojd/2-velocity ;
            double delta=USHandler::adjustPRP()/hojd;
            velocity=velocity*delta;
            //  printf("Delta: %f", delta);
            //  fflush(stdout);
            if(velocity<maxVelocity){
                maxVelocity=velocity;
            }
        }else{velocity=0;}


        //emit usptr->newSpect(tempIm, velocity);
        emit usptr->newSpect(tmpPtr, velocity);
        mutex.unlock();
}

void USHandler::freqChanger(int selectionMax){

    /*This function handles the change in frequency once the
        pixel sums of a selected area has been calculated
       for
      the current frequency.
      The selectionSums vector needs to hold doubles, the
          static_cast ensures that selectionMax value is
          converted into a
      double. This is for comparison reasons.
      */
    selectionSums[spectSumCount]=static_cast<double>(
       selectionMax);
```

```
        spectSumCount++;

        //Checks to see if the new sum found is the largest one
            yet.
        if(selectionMax>maxVal){
            maxVal=selectionMax;
            bestFreq=currFreq;
        }

        /*Updates the transmitting frequency. It increases this
            by the stepsize set in the constructor
             until the maximum frequency is reached. */

        if(freqCount<freqVectSize-1){
            //currFreq=currFreq+stepSize;


            //printf("freqcount = %d \n", freqCount);
            //fflush(stdout);

            currFreq=freqVect[freqCount];

            ult.setParamValue("b-freq",currFreq);
            int val =0;
            while(val!=currFreq){
                    ult.getParamValue("b-freq",val);
            }

        }
        else{//The frequency sweep is done. Reseting some
            variables and set the frequency to the optimum found.
            spectSumCount=0;
            //printf("Am I here? \n");
            //fflush(stdout);
            freqCount=0;
            ult.setParamValue("b-freq",bestFreq);
            bestFound=true;
            currFreq=freqVect[0];
            selMade=false;
        }
        freqCount++;
}

void USHandler::Doppler()
{
        /*This function changes the mode to pulsed doppler (3)
```

```
        and adjusts some parameters.
      If the machine is not yet connected, it simply does
          nothing.
      Once all the parameters are set, a timer starts. When
          the time is up, 'stopDoppler' is called.
      */
    if(!ult.isConnected()){return;}
    if(ult.getFreezeState()){
        ult.toggleFreeze();
    }
    USHandler::bufferDoppler = (uchar *)malloc(imSize *
        sizeof(uchar));
    ult.selectMode(3);
    ult.setParamValue("b-freq",bestFreq);
    ult.setParamValue("pw-freq", bestFreq);
    ult.setParamValue("pw-voltage [-] rng1",15);
    ult.setParamValue("pw-voltage [-] rng2",15);
    ult.setParamValue("pw-voltage [-] rng3",15);
    ult.setParamValue("pw-gate pos",100);
    ult.setParamValue("pw-gate depth",16000);
    ult.setParamValue("pw-gate size",13000);
    ult.setParamValue("pw-wf",1);
    ult.setParamValue("pw-steer",0);
    ult.setParamValue("pw-prp",PRPsetting);
    ult.setParamValue("pw-noise level",50);
    ult.setParamValue("pw/color-pulse repeat",4);
    ult.setParamValue("pw-gain audio",0);
    ult.setParamValue("pw-LRP",4000);
    ult.setParamValue("pw-baseline",0);

    ult.setDataToAcquire(udtPWSpectrum);
    ult.setCallback(&USHandler::processDoppler);

    //The top velocity found is set to 0.
    maxVelocity=0;

    /*A timer is started. The velocity measurement goes on
        for 10 seconds.
    When the time is up, an interrupt is sent out, forcing '
        stopDoppler' to execute.
    The maximum velocity found during the 10 second interval
         is then registred and written to the patient file.
    */
    int time = measurementTime * 1000;
    QTimer::singleShot(time, usptr, SLOT(stopDoppler()));

}
```

```
bool USHandler::isSel()
{
    return selMade;
}

void USHandler::setSelected(){
    selMade=true;

}
void USHandler::setSelArea(int selTop, int selBot, int
   selLeft, int selRight)
{
    /* This functin is called once an area has been selected
        in the B-mode image.
      It sets the required parameters in order to start a
         frequency optimization.
      */
    selMade=true;
    yy11=selTop;
    yy2=selBot;
    xx1=selLeft;
    xx2=selRight;
    maxVal=0;
    bestFound=false;
    currFreq=startFreq;
    spectSumCount=0;

}


double USHandler::adjustPRP(){
    /*This function returns the multiplier to velocity
        determined from PRF and Frequency
      It first fetches the prp value from the ultrasonix
         machine, and puts the value into prp.
      Then it calculates the proper value.
      The equation might look odd, but this is in order to
         avoid going outside the range of a double.
      */
    int prp;
    ult.getParamValue("pw-prp",prp);

    double val = (15400000/(double)bestFreq)*2*100000/(4*(
       double)prp);//mm/s //*2 fran desc.h
    return val;

}
```

```
void USHandler::updateTGC(int tgc1, int tgc2,int tgc3,int
    tgc4,int tgc5,int tgc6,int tgc7,int tgc8){
    //Simply a funcion to update the Time Gain Control from
        the sliders in the user interface.

    tgc.v1=tgc1;
    tgc.v2=tgc2;
    tgc.v3=tgc3;
    tgc.v4=tgc4;
    tgc.v5=tgc5;
    tgc.v6=tgc6;
    tgc.v7=tgc7;
    tgc.v8=tgc8;

    ult.setParamValue("gain_curve",tgc);
}

/* Help function to change Pulse frequency depending on
    slider in SettingsDialog */
void USHandler::changePRP(int prpPos){

    switch (prpPos){
    case 1:
        ult.setParamValue("pw-prp",200);
        PRPsetting=200;
        break;
    case 2:
        ult.setParamValue("pw-prp",250);
        PRPsetting=250;
        break;
    case 3:
        ult.setParamValue("pw-prp",300);
        PRPsetting=300;
        break;
    case 4:
        ult.setParamValue("pw-prp",400);
        PRPsetting=400;
        break;
    case 5:
        ult.setParamValue("pw-prp",600);
        PRPsetting=600;
        break;
    case 6:
        ult.setParamValue("pw-prp",800);
        PRPsetting=800;
        break;
    }
```

```
}

int USHandler::getbestFreq(){
    return bestFreq;
}
int USHandler::getCurrFreq(){
    return currFreq;
}

QVector<double> USHandler::getSums(){
    return selectionSums;
}
QVector<double> USHandler::getFreqs(){
    QVector<double> x(freqVectSize);
    for(int i=0;i<freqVectSize;i++){
        x[i]=(double)freqVect[i];
    }
    return x;
}

int USHandler::getStartFreq(){
    return startFreq;
}

int USHandler::getMaxFreq(){
    return maxFreq;
}

int USHandler::getStepSize(){
    return stepSize;
}

int USHandler::getMaxVal(){
    return maxVal;
}

/* Function that makes it possible to interrupt a doppler
   measurement mid run */
void USHandler::stopDoppler(){
    if(ult.getActiveImagingMode() == 3){
        ult.toggleFreeze();
        emit usptr->measureFinished(maxVelocity);
    }
}

void USHandler::setIP(QString address){
    ipAddr=address;
```

```
}

// Set the pulse frequency
void USHandler::setPRP(int prp){
        if(ult.isConnected()){
        if(ult.getActiveImagingMode()==3){
            changePRP(prp);
            int val;
            ult.getParamValue("pw-prp",val);
            printf("Prp set to: %d", val);
            fflush(stdout);
        }
    }
}

void USHandler::setTime(int time){measurementTime=time;}
```

B.7 patientdialog.h

```
#ifndef PATIENTDIALOG_H
#define PATIENTDIALOG_H

#include <QDialog>

namespace Ui {
class PatientDialog;
}

class PatientDialog : public QDialog
{
    Q_OBJECT

public:
    explicit PatientDialog(QWidget *parent = 0);
    ~PatientDialog();
    QString getItem(int index);

private slots:

    void on_firstName_lostFocus();

    void on_lastName_lostFocus();

    void on_ssNbr_lostFocus();


    void on_other_lostFocus();
```

```
private:
    Ui::PatientDialog *ui;
    QString* array;
};

#endif // PATIENTDIALOG_H
```

B.8 patientdialog.cpp

```
#include "patientdialog.h"
#include "ui_patientdialog.h"

PatientDialog::PatientDialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::PatientDialog)
{
    ui->setupUi(this);
    array = new QString[4];
}

PatientDialog::~PatientDialog()
{
    delete ui;
}


QString PatientDialog::getItem(int index) {
    return array[index];
}

void PatientDialog::on_firstName_lostFocus()
{
        array[0]=ui->firstName->text();
}

void PatientDialog::on_lastName_lostFocus()
{
        array[1]=ui->lastName->text();
}

void PatientDialog::on_ssNbr_lostFocus()
{
        array[2]=ui->ssNbr->text();
}


void PatientDialog::on_other_lostFocus()
{
```

```
    array[3]=ui->other->text();
}
```

B.9 settingsdialog.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>SettingsDialog</class>
 <widget class="QDialog" name="SettingsDialog">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>400</width>
    <height>300</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Dialog</string>
  </property>
  <widget class="QWidget" name="horizontalLayoutWidget">
   <property name="geometry">
    <rect>
     <x>30</x>
     <y>60</y>
     <width>111</width>
     <height>31</height>
    </rect>
   </property>
   <layout class="QHBoxLayout" name="horizontalLayout">
    <item>
     <widget class="QLabel" name="ipLabel">
      <property name="text">
       <string>IP:</string>
      </property>
     </widget>
    </item>
    <item>
     <widget class="QLineEdit" name="ipLine">
      <property name="text">
       <string>127.0.0.1</string>
      </property>
     </widget>
    </item>
   </layout>
  </widget>
  <widget class="QLabel" name="ipDesc">
   <property name="geometry">
```

```
    <rect>
     <x>20</x>
     <y>10</y>
     <width>161</width>
     <height>41</height>
    </rect>
   </property>
   <property name="text">
    <string>IP address of SonixRP
127.0.0.1 if the program
is run on the SonixRP(Default)</string>
   </property>
  </widget>
  <widget class="QSlider" name="prpSlider">
   <property name="geometry">
    <rect>
     <x>20</x>
     <y>170</y>
     <width>191</width>
     <height>20</height>
    </rect>
   </property>
   <property name="minimum">
    <number>1</number>
   </property>
   <property name="maximum">
    <number>6</number>
   </property>
   <property name="singleStep">
    <number>1</number>
   </property>
   <property name="pageStep">
    <number>10</number>
   </property>
   <property name="value">
    <number>4</number>
   </property>
   <property name="tracking">
    <bool>true</bool>
   </property>
   <property name="orientation">
    <enum>Qt::Horizontal</enum>
   </property>
   <property name="tickPosition">
    <enum>QSlider::TicksBelow</enum>
   </property>
   <property name="tickInterval">
```

```
   <number>1</number>
  </property>
 </widget>
 <widget class="QLabel" name="label">
  <property name="geometry">
   <rect>
    <x>40</x>
    <y>140</y>
    <width>161</width>
    <height>16</height>
   </rect>
  </property>
  <property name="text">
   <string>PRP Adjustment.</string>
  </property>
 </widget>
 <widget class="QLabel" name="label_2">
  <property name="geometry">
   <rect>
    <x>240</x>
    <y>120</y>
    <width>141</width>
    <height>51</height>
   </rect>
  </property>
  <property name="text">
   <string>Doppler Measurement Time</string>
  </property>
 </widget>
 <widget class="QLabel" name="label_3">
  <property name="geometry">
   <rect>
    <x>20</x>
    <y>200</y>
    <width>201</width>
    <height>16</height>
   </rect>
  </property>
  <property name="text">
   <string>200      250      300      400      600
       800</string>
  </property>
 </widget>
 <widget class="QLineEdit" name="timeLine">
  <property name="geometry">
   <rect>
    <x>260</x>
```

```
    <y>170</y>
    <width>89</width>
    <height>20</height>
   </rect>
  </property>
  <property name="text">
   <string>10</string>
  </property>
 </widget>
 <widget class="QLabel" name="label_4">
  <property name="geometry">
   <rect>
    <x>350</x>
    <y>170</y>
    <width>20</width>
    <height>20</height>
   </rect>
  </property>
  <property name="text">
   <string>s</string>
  </property>
 </widget>
 <widget class="QWidget" name="horizontalLayoutWidget_2">
  <property name="geometry">
   <rect>
    <x>170</x>
    <y>259</y>
    <width>160</width>
    <height>31</height>
   </rect>
  </property>
  <layout class="QHBoxLayout" name="hLayout">
   <item>
    <widget class="QPushButton" name="OKButton">
     <property name="text">
      <string>OK</string>
     </property>
    </widget>
   </item>
   <item>
    <widget class="QPushButton" name="CancelButton">
     <property name="text">
      <string>Cancel</string>
     </property>
    </widget>
   </item>
  </layout>
```

```
  </widget>
 </widget>
 <resources/>
 <connections/>
</ui>
```

B.10 settingsdialog.h

```
#ifndef SETTINGSDIALOG_H
#define SETTINGSDIALOG_H

#include <QDialog>
#include <QMessageBox>
#include <QRegexp>

namespace Ui {
class SettingsDialog;
}

class SettingsDialog : public QDialog
{
    Q_OBJECT

public:
    explicit SettingsDialog(QWidget *parent = 0);
    ~SettingsDialog();
    QString getIP();
    int getPRP();
    int getTime();

private slots:
    void on_ipLine_lostFocus();
    void checkValues();

    void on_CancelButton_clicked();

private:
    Ui::SettingsDialog *ui;
    QString string;
    QRegExp rx;
    bool validIPv4(QString string);
};

#endif // SETTINGSDIALOG_H
```

B.11 settingsdialog.cpp

```
#include "settingsdialog.h"
```

```
#include "ui_settingsdialog.h"

SettingsDialog::SettingsDialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::SettingsDialog)
{
    ui->setupUi(this);
    ui->ipLine->setText("127.0.0.1");
    //ui->ipLine->setText("130.235.53.125");
    ui->OKButton->setAutoDefault(false);
    ui->CancelButton->setAutoDefault(false);

    //Set Regular expression to check for invalid IPv4
        address
    rx.setPattern("((25[0-5]|2[0-4]\\d|1\\d\\d|[1-9]\\d|\\d)
        \\.){3}(25[0-5]|2[0-4]\\d|1\\d\\d|[1-9]\\d|\\d)");

    connect(ui->OKButton,SIGNAL(clicked()),this, SLOT(
        checkValues()));
}

SettingsDialog::~SettingsDialog()
{
    delete ui;
}

//Get IP
QString SettingsDialog::getIP()
{
    return string;
}

void SettingsDialog::on_ipLine_lostFocus()
{
    string = ui->ipLine->text();
}

/* Get pulse frequency */
int SettingsDialog::getPRP(){
    int val = (ui->prpSlider->value());
    return val;
}

/* Fetch time for Doppler measurement */
int SettingsDialog::getTime(){
    QString timeString = ui->timeLine->text();
    int time = timeString.toInt();
```

```
    return time;
}

/* Verify valid IPv4 address */
void SettingsDialog::checkValues()
{
    if (rx.exactMatch(string)) {
        accept();
    } else {
        QMessageBox::information(this,"Fail","Invalid IPv4
            address, try again.");
    }
}


void SettingsDialog::on_CancelButton_clicked()
{
    reject();
}
```

B.12 settingsdialog.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>SettingsDialog</class>
 <widget class="QDialog" name="SettingsDialog">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>400</width>
    <height>300</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Dialog</string>
  </property>
  <widget class="QWidget" name="horizontalLayoutWidget">
   <property name="geometry">
    <rect>
     <x>30</x>
     <y>60</y>
     <width>111</width>
     <height>31</height>
    </rect>
   </property>
   <layout class="QHBoxLayout" name="horizontalLayout">
    <item>
```

```
      <widget class="QLabel" name="ipLabel">
       <property name="text">
        <string>IP:</string>
       </property>
      </widget>
     </item>
     <item>
      <widget class="QLineEdit" name="ipLine">
       <property name="text">
        <string>127.0.0.1</string>
       </property>
      </widget>
     </item>
    </layout>
   </widget>
   <widget class="QLabel" name="ipDesc">
    <property name="geometry">
     <rect>
      <x>20</x>
      <y>10</y>
      <width>161</width>
      <height>41</height>
     </rect>
    </property>
    <property name="text">
     <string>IP address of SonixRP
127.0.0.1 if the program
is run on the SonixRP(Default)</string>
    </property>
   </widget>
   <widget class="QSlider" name="prpSlider">
    <property name="geometry">
     <rect>
      <x>20</x>
      <y>170</y>
      <width>191</width>
      <height>20</height>
     </rect>
    </property>
    <property name="minimum">
     <number>1</number>
    </property>
    <property name="maximum">
     <number>6</number>
    </property>
    <property name="singleStep">
     <number>1</number>
```

```
  </property>
  <property name="pageStep">
   <number>10</number>
  </property>
  <property name="value">
   <number>4</number>
  </property>
  <property name="tracking">
   <bool>true</bool>
  </property>
  <property name="orientation">
   <enum>Qt::Horizontal</enum>
  </property>
  <property name="tickPosition">
   <enum>QSlider::TicksBelow</enum>
  </property>
  <property name="tickInterval">
   <number>1</number>
  </property>
 </widget>
 <widget class="QLabel" name="label">
  <property name="geometry">
   <rect>
    <x>40</x>
    <y>140</y>
    <width>161</width>
    <height>16</height>
   </rect>
  </property>
  <property name="text">
   <string>PRP Adjustment.</string>
  </property>
 </widget>
 <widget class="QLabel" name="label_2">
  <property name="geometry">
   <rect>
    <x>240</x>
    <y>120</y>
    <width>141</width>
    <height>51</height>
   </rect>
  </property>
  <property name="text">
   <string>Doppler Measurement Time</string>
  </property>
 </widget>
 <widget class="QLabel" name="label_3">
```

```
 <property name="geometry">
  <rect>
   <x>20</x>
   <y>200</y>
   <width>201</width>
   <height>16</height>
  </rect>
 </property>
 <property name="text">
  <string>200      250      300      400      600
      800</string>
 </property>
</widget>
<widget class="QLineEdit" name="timeLine">
 <property name="geometry">
  <rect>
   <x>260</x>
   <y>170</y>
   <width>89</width>
   <height>20</height>
  </rect>
 </property>
 <property name="text">
  <string>10</string>
 </property>
</widget>
<widget class="QLabel" name="label_4">
 <property name="geometry">
  <rect>
   <x>350</x>
   <y>170</y>
   <width>20</width>
   <height>20</height>
  </rect>
 </property>
 <property name="text">
  <string>s</string>
 </property>
</widget>
<widget class="QWidget" name="horizontalLayoutWidget_2">
 <property name="geometry">
  <rect>
   <x>170</x>
   <y>259</y>
   <width>160</width>
   <height>31</height>
  </rect>
```

```
    </property>
    <layout class="QHBoxLayout" name="hLayout">
     <item>
      <widget class="QPushButton" name="OKButton">
       <property name="text">
        <string>OK</string>
       </property>
      </widget>
     </item>
     <item>
      <widget class="QPushButton" name="CancelButton">
       <property name="text">
        <string>Cancel</string>
       </property>
      </widget>
     </item>
    </layout>
   </widget>
  </widget>
  <resources/>
  <connections/>
</ui>
```

B.13 keypresswidget.h

```
#ifndef KEYPRESSWIDGET_H
#define KEYPRESSWIDGET_H

#include <QLabel>
#include <QMouseEvent>
#include <QPainter>

class keyPressWidget : public QLabel
{
    Q_OBJECT
public:
    explicit keyPressWidget(QLabel *parent = 0);
    void mousePressEvent(QMouseEvent *e);
    void mouseReleaseEvent(QMouseEvent *e);

signals:

public slots:

};

#endif // KEYPRESSWIDGET_H
```

B.14 keypresswidget.cpp

```
#include "keypresswidget.h"
#include "mainwindow.h"

QPainter painter;

    QPainter paint();
keyPressWidget::keyPressWidget(QLabel *parent) :
    QLabel(parent)
{
    keyPressWidget::setEnabled(true);
  QPainter painter(this);
}

/* Function to get mouse coords for where the mouse button
    is
  pressed down */
void keyPressWidget::mousePressEvent(QMouseEvent *e){
    MainWindow::setSelection1(e->x(), e->y());
}

/* Function to get mouse coords for where the mouse button
    is
  released */
void keyPressWidget::mouseReleaseEvent(QMouseEvent *e){
    MainWindow::setSelection2(e->x(), e->y());
}
```

B.15 worker.h

```
#ifndef WORKER_H
#define WORKER_H
#include <QtGui>
#include "ushandler.h"

class Worker : public QThread
{
    Q_OBJECT

public:
    Worker(QObject *parent = 0, USHandler *ptr = 0);
    ~Worker();
    void run();

private:
    USHandler *usptr;
```

```
private slots:
    void BWork();
    void dopplerWork();

signals:
    void BFrameDone();
    void SpectDone();

};

#endif // WORKER_H
```

B.16 worker.cpp

```
#include "worker.h"

Worker::Worker(QObject *parent, USHandler *ptr)
    :QThread(parent)
{
    usptr=ptr;
}

Worker::~Worker()
{

}

void Worker::run()
{
}

/* Start heavy computations in this worker
   thread instead of MainWindow */
void Worker::BWork()
{
    usptr->updateBFrame();
    emit BFrameDone();
}

/* Start heavy computations in this worker
   thread instead of MainWindow */
void Worker::dopplerWork()
{
    usptr->updateDoppler();
    emit SpectDone();
}
```