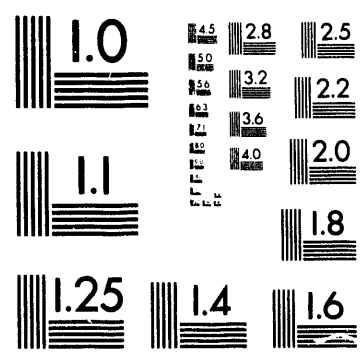

1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0

1 of 1

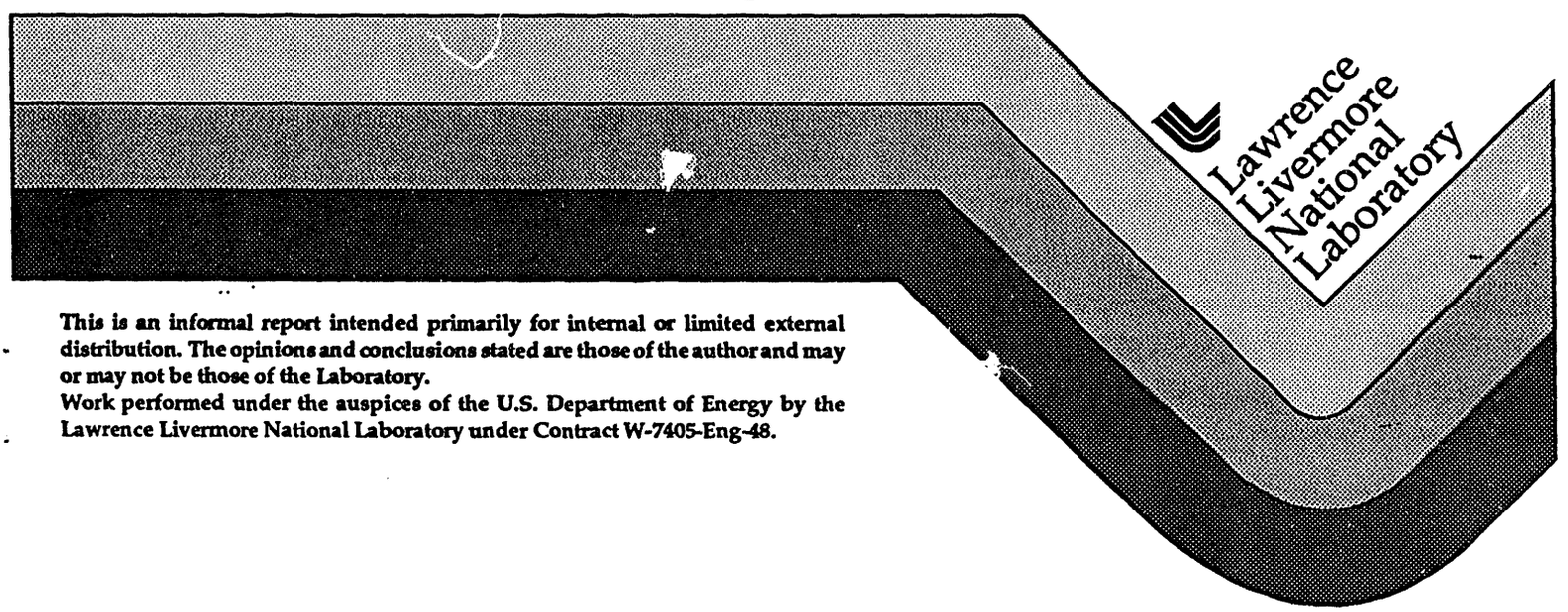
12/8-16-93 SD

UCRL-ID-113760

Signal Processing of Shiley Heart Valve Data for Fracture Detection

Carmen Mullenhoff

April 1, 1993



Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

This report has been reproduced
directly from the best available copy.

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information
P.O. Box 62, Oak Ridge, TN 37831
Prices available from (615) 576-8401, FTS 626-8401.

Available to the public from the
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd.,
Springfield, VA 22161

Price Code	Page Range
A01	Microfiche
Papercopy Prices	
A02	1- 10
A03	11- 50
A04	51- 75
A05	76-100
A06	101-125
A07	126-150
A08	151-175
A09	176-200
A10	201-225
A11	226-250
A12	251-275
A13	276-300
A14	301-325
A15	326-350
A16	351-375
A17	376-400
A18	401-425
A19	426-450
A20	451-475
A21	476-500
A22	501-525
A23	526-550
A24	551-575
A25	576-600
A99	601 & UP

ABSTRACT

Given digital acoustic data emanating from the heart sounds of the beating heart measured from laboratory sheep with implanted Björk-Shiley Convexo-Concave heart valves, it is possible to detect and extract the opening and closing heart beats from the data. Once extracted, spectral or other information can then obtained from the heartbeats and passed on to feature extraction algorithms, neural networks, or pattern recognizers so that the valve condition, either fractured or intact, may be determined.

1.0 INTRODUCTION

Artificial heart valves have provided individuals with heart ailments an extended useful and productive life.[1] Unfortunately, mechanical failure of these valves can be fatal, if not detected early. Once detected, the treatment is an immediate reoperation and replacement of the valve [2]. Therefore, it is necessary to develop valve failure detection schemes that are highly reliable. Currently, there are no sensitive or reliable mechanisms to detect valve failure.

Lawrence Livermore National Laboratory (LLNL) proposes to perform the research and development necessary to construct an instrument capable of non-invasively classifying the condition of implanted Björk-Shiley Convexo-Concave (BSCC) heart valves. This instrument will analyze acoustic signals recorded in vivo generated by a functioning heart valve implanted in sheep. We will first develop the signal processing necessary to extract pertinent information from the acoustic data and then develop BSCC heart valve classification algorithms based on features of the enhanced acoustic signals. These algorithms will automatically classify the condition of BSCC heart valves implanted in sheep.

Classifying the condition of implanted BSCC heart valves is a difficult, complex task. After preliminary analysis of acoustic data, it is readily apparent that a standard signal processing solution would not be possible due to the hostile nature of the biological environment, valve dynamics and noise from various sources. The acoustic signals generated by the BSCC heart valves become distorted while propagating through the biological medium of the sheep and eventually are digitized and recorded at the output of a sensitive microphone also subjected to extraneous noise sources. Since the fundamental physical characteristics (e.g. resonant frequencies) of these acoustic signals vary at each heartbeat, it is necessary to incorporate all of these characteristics or features in a detection scheme aimed at differentiating between fractured and unfractured valves. Based on a careful analysis of the modal structure of the valve performed both at Shiley Inc. and LLNL, simple simulations of this modal response and more importantly, in-situ real-time data/spectral analysis of ovine acoustic measurements gathered by LLNL at a veterinary laboratory, we propose the following to solve the BSCC heart valve classification problem. Our approach to solve the heart valve classification problem is based on Statistical Pattern Recognition which essentially interprets the spectrogram surface as either a signal or an image, extracts so-called features from it and attempts to define various decision regions for detection/classification. As part of this study we have also have investigated "adaptive" type classification schemes using neural networks.

Here algorithms modeled approximately on the human brain, are applied to spectrogram data once the important features are extracted. The network learns the various valve classes by repeated application of data. Both techniques offer much promise, but again large quantities of high quality acoustic data must be processed to quantify their performance with acceptable statistical reliability. Of course, improved signal processing of the spectrogram and/or feature vectors can enhance performance. Our approach is outlined in more detail below in Figure 1. The steps we require to classify the condition of a given BSCC heart valve are to: (1) acquire and digitize the acoustic measurements after the appropriate signal conditioning (performed by Shiley Inc.), (2) perform the necessary signal processing to enhance the particular characteristic or features of the BSCC heart valve (e.g. pre-assigned frequency bins) which will be used for detection/discrimination, (3) extract the selected features of interest in a simplified form (e.g. averaging adjacent frequency bins), (4) perform heart valve classification (fractured/unfractured) using techniques from detection/pattern recognition on either the one-dimensional acoustic signal (feature versus time/frequency) or on the entire two-dimensional structure (feature versus heart beat number) evolving from the data. Note also that we will apply "adaptive" pattern recognition techniques implemented using a so-called *neural-net* algorithm, (5) test the classification algorithm using a simulator first (see Figure below) to ascertain proper performance and then measured acoustic data composed of a training set to adjust algorithm parameters followed by a test set to generate sample statistics (probability of detection, etc.), and finally (6) implement the algorithm along with an accompanying expert system (eventually in the instrument) and process all of the acquired data set to assess overall performance (probability of detection, miss, false alarm etc.). In order to fully develop this approach, it is first necessary to identify the difference between the acoustic signals generated by intact and fractured or single leg separation (SLS) BSCC valves (modeling/experiment). It will be necessary to compensate for distortions caused by biological tissue and by electronic instrumentation (signal processing). Large volumes of acoustic data must be reduced to a manageable size without losing significant information about the functioning heart valve (feature extraction). A classification algorithm with extreme reliability must be developed and proven (pattern recognition/test). The resulting signal processing, feature extraction, and classification algorithms must be combined with other data and algorithms into a sophisticated instrument for application to the clinical setting.

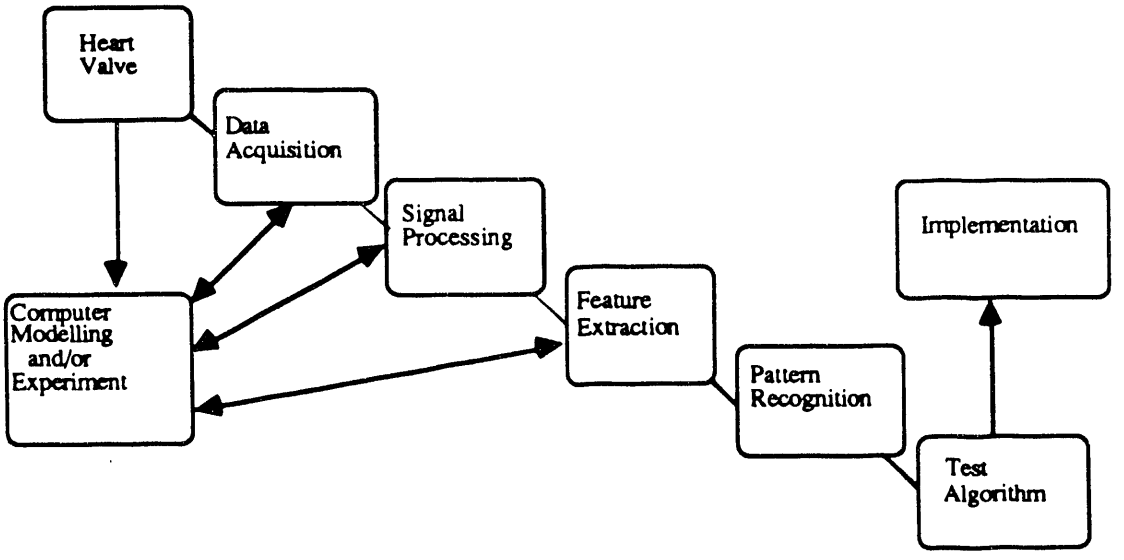


Figure 1. LLNL Approach to BSCC Heart Valve Classification Problem.

The goal of this particular research project is to develop a technique to non-invasively monitor the acoustic signals produced by implanted BSCC heart valves in order to determine the structural condition of the valves. It is postulated that an intact valve radiates a different acoustic signal than a SLS valve. A current feasibility study conducted by LLNL indicates that it is possible to determine valve condition. However, the main problem yet to be solved is to identify all of the differences between the acoustic signatures of intact and SLS valves and construct a classification system capable of automatically recognizing those differences.

Shiley has provided the heart valve and the data acquisition, and so it is the purpose of this report to outline our method for signal processing of the data. Shown in Figure 2 is an expansion of the Signal Processing block of Figure 1.

The main purpose of the Signal Processing of the heart valve data is to enhance signal levels and minimize the effects of noise and obtain results which will be put it into a format that is useful for Feature Extraction. Initially, we must detect the various heartbeats (i.e. openings, closings, etc.), screen for unacceptable beats, and extract them from the data. We then filter the heartbeats into appropriate frequency bands and find their corresponding spectrograms.

Section 2 will discuss our methodology and the algorithms for processing the data The Appendix will have the actual C code description and usage.

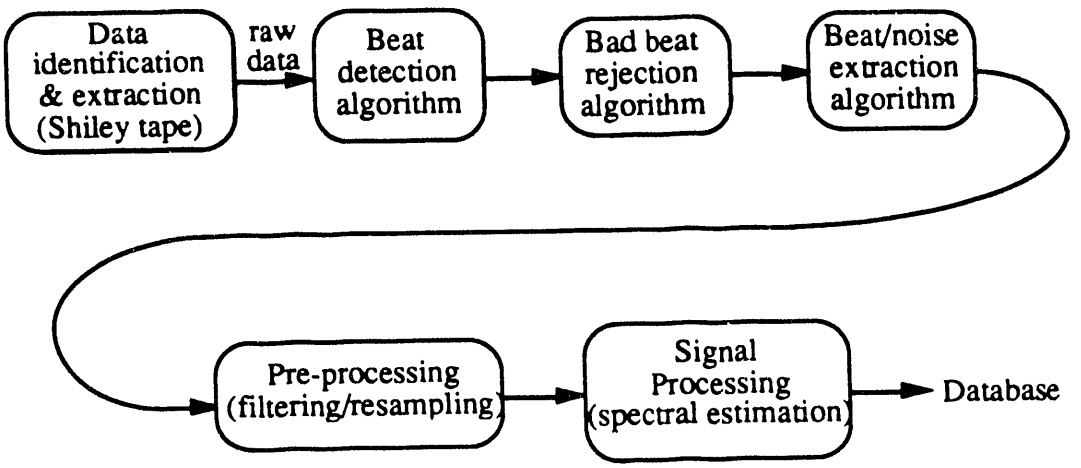


Figure 2: Signal Processing of Heart Valve Acoustic Data

2.0 ALGORITHMS

In this section, we discuss each operation depicted in Figure 2. First we discuss the data identification and extraction procedure. Next, we detail our beat detection algorithm followed by explanation our bad beat rejection algorithm. We then talk about how and why we filter and resample the data into bands followed by discussion of the spectral estimation techniques that we chose. Finally, we list the database created from the signal processing.

2.1 Data Identification and Extraction

The data from Shiley comes in TAR format on Exabyte tapes. Since we have no prior knowledge of the exact files names, we must first list the table of contents of the Shiley tape. Due to large file sizes and limited disk space, we then extract a subset of the files from the tape and process them until all the files have been completed. Once a file is on the hard drive, it must be converted from the Shiley interleaved format to a format that can be readily utilized by our software. This is accomplished by removing the 1025 character header, converting the remainder to a 16 bit integer file followed by conversion to a binary floating point file. We call this binary floating point file the raw data.

2.2 Beat Detection

Once we have the raw data, we can process it with our beat detection algorithm. The purpose of this algorithm is to create a text file, called a "BEATS" file, which contains the location in the raw data file and maximum value of each acceptable opening and closing beat. The block diagram of our beat detection algorithm is shown in Figure 3.

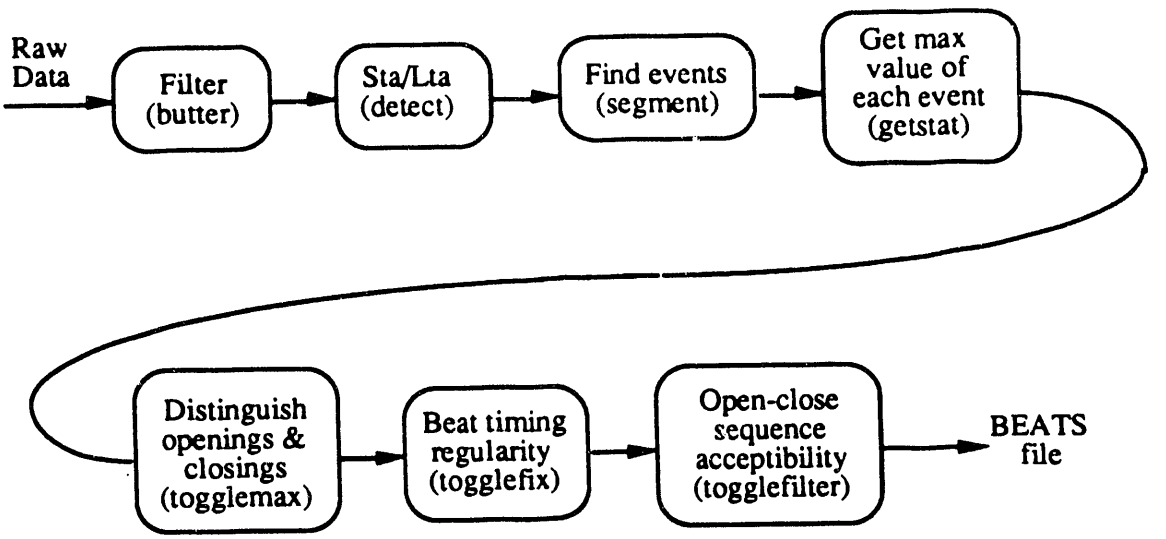


Figure 3: Beat Detection Algorithm

The first operation we perform on the raw data is to bandpass filter it with a third order butterworth filter. This filtering operation helps to smooth the data and suppress noise so that it is better conditioned for the detection process.

Then, from the filtered raw data we compute a signal that consists of the ratio of a sliding short term average to sliding long term average. The output is a signal of potential events, where an event is either an opening or closing. The short term average is computed as,

$$m_{ST}(x)=\frac{1}{N_{ST}}\sum_{k=0}^{N_{ST}} x(t_k)\ , \quad \text{typically } N_{ST} = 50 \text{ samples}$$

and the long term average is computed as,

$$m_{LT}(x)=\frac{1}{N_{LT}}\sum_{k=0}^{N_{LT}} x(t_k), \quad \text{typically } N_{LT} = 500 \text{ samples}$$

and their ratio is then,

$$R_{STLT}=\frac{m_{ST}(x)}{m_{LT}(x)}$$

After this signal of potential events, R_{STLT}, is created, we then pick both a threshold to discriminate between the noise and the signal for event screening, and a windowsize for which events occurring inside will be coalesced together. From looking at R_{STLT} in Figure 4, we see that a good threshold to use is 3.0, and from experience, a good windowsize to use is 2800 samples based on a sampling rate of $2.0833 * 10^{-5}$ seconds. At this point, a text file with the beginning and ending locations of each event in the raw data file is created. Shown here is a portion of what the text file looks like:

11864	13425
16428	16504
54364	54831
73007	73079
94616	95018
112161	113195
135352	135769
152912	154298

Using the text file of events just created, the absolute maximum value of each event is found in the raw data file and then appended to the text file. The text file then looks like:

11864	13425	14359.000000
16428	16504	131.000000
54364	54831	8361.000000
73007	73079	227.000000
94616	95018	17119.000000
112161	113195	133.000000
135352	135769	17063.000000
152912	154298	283.000000

Now that we know the maximum value of each event, we should be able to tell which is an opening and which is a closing. Distinguishing between openings and closings is accomplished by finding the best threshold that maximizes the toggles between the openings and closings. After this threshold value is determined, a 0 or 1 is appended onto the text file, where a 0 indicates an opening, and a 1 indicates a closing. The text file now appears as:

11864	13425	14359.000000	1
16428	16504	131.000000	0
54364	54831	8361.000000	1
73007	73079	227.000000	0
94616	95018	17119.000000	1
112161	113195	133.000000	0
135352	135769	17063.000000	1
152912	154298	283.000000	0

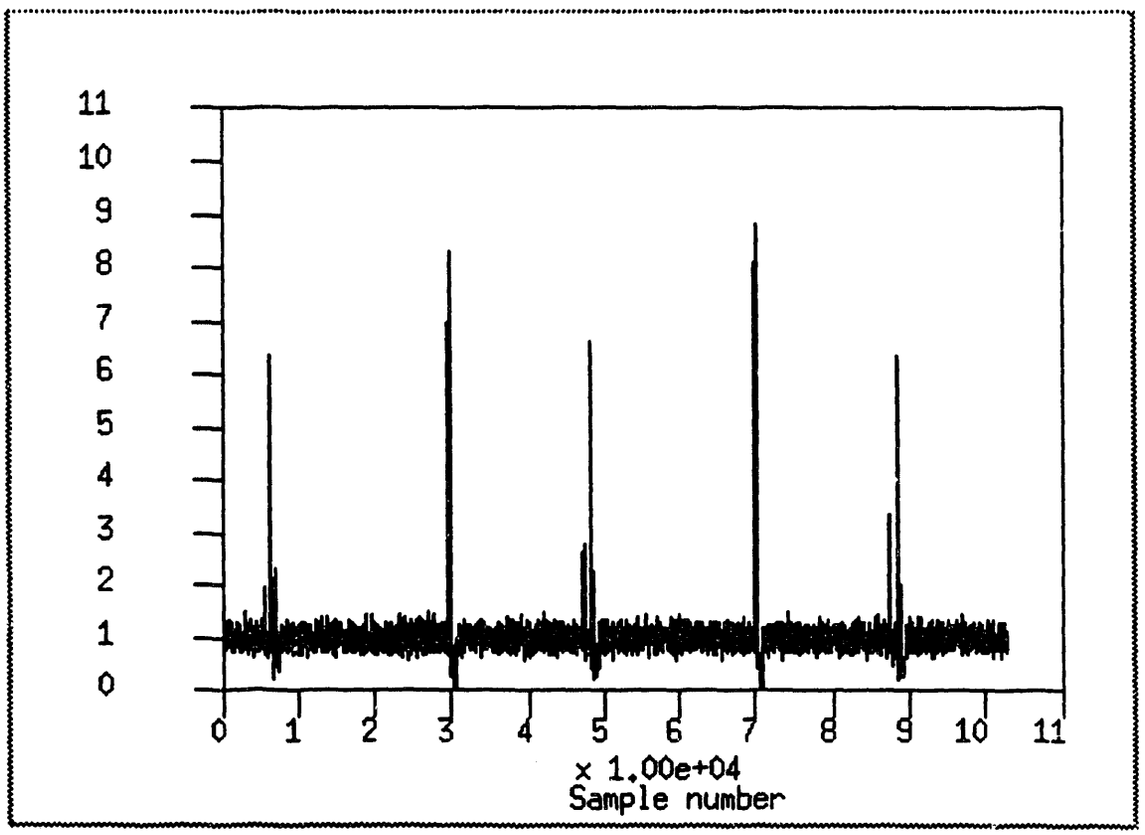


Figure 4: Example of the Sta/Lta Signal

Next, we perform a beat timing check which relies on and ensures the regularity of the beats. The average time between similar events is calculated and a check is made for each event to find out if it is within a certain percentage deviation from that average. When an out of range event is found, a -1 is inserted in place of the 1 or 0 that was already there in the text file. We typically allow a 20% deviation. The marked text file appears like this:

11864	13425	14359.000000	1
16428	16504	131.000000	-1
54364	54831	8361.000000	1
73007	73079	227.000000	0
94616	95018	17119.000000	1
112161	113195	133.000000	0
135352	135769	17063.000000	1
152912	154298	283.000000	0

The final step is to output the corrected text file called the "BEATS" file, which is done by taking into account the events marked with a -1 and keeping only acceptable length open-close sequences. We typically accept sequences with length five or greater. The "BEATS" file would then look like this:

54364	54831	8361.000000	1
73007	73079	227.000000	0
94616	95018	17119.000000	1
112161	113195	133.000000	0
135352	135769	17063.000000	1
152912	154298	283.000000	0

2.3 Bad Beat Rejection

In the bad beat rejection algorithm, we ensure the proper beat ordering, and perform some outlier quality control. Although at this point we could extract the openings and closings based on the "BEATS" file created thus far, because the openings and closings vary somewhat widely in amplitude, we want to reject those that are outside of a certain boundary. In Figure 5 is shown the block diagram of the bad beat rejection algorithm.

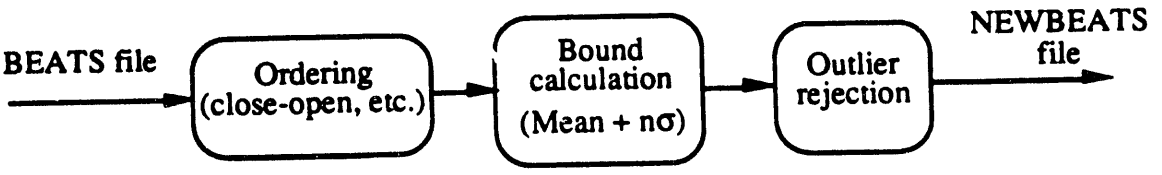


Figure 5: Bad Beat Rejection Algorithm

The first operation performed is to search through the "BEATS" file and ensure that the ordering is close-open-close-open etc. (e.g. If the sequence ordering was close-open-open, then the second opening would be rejected.) A temporary "BEATS" file is created with the proper ordering. Next, from the absolute maximum values of the openings in the temporary "BEATS" file, a mean and standard deviation(σ) are calculated. Then, all close-open sequences whose absolute opening maximum value lies outside the mean $\pm$ $n\sigma$, where n is an input, are rejected. We typically use n= 0.5 if there are more than 100 beats detected, otherwise we use n=1.0 if there are less. A "NEWBEATS" file is then created.

Illustrated in Figure 6 is an example before and after the bad beat rejection algorithm. As can be seen from the example, the first opening is kept, and the other two are rejected.

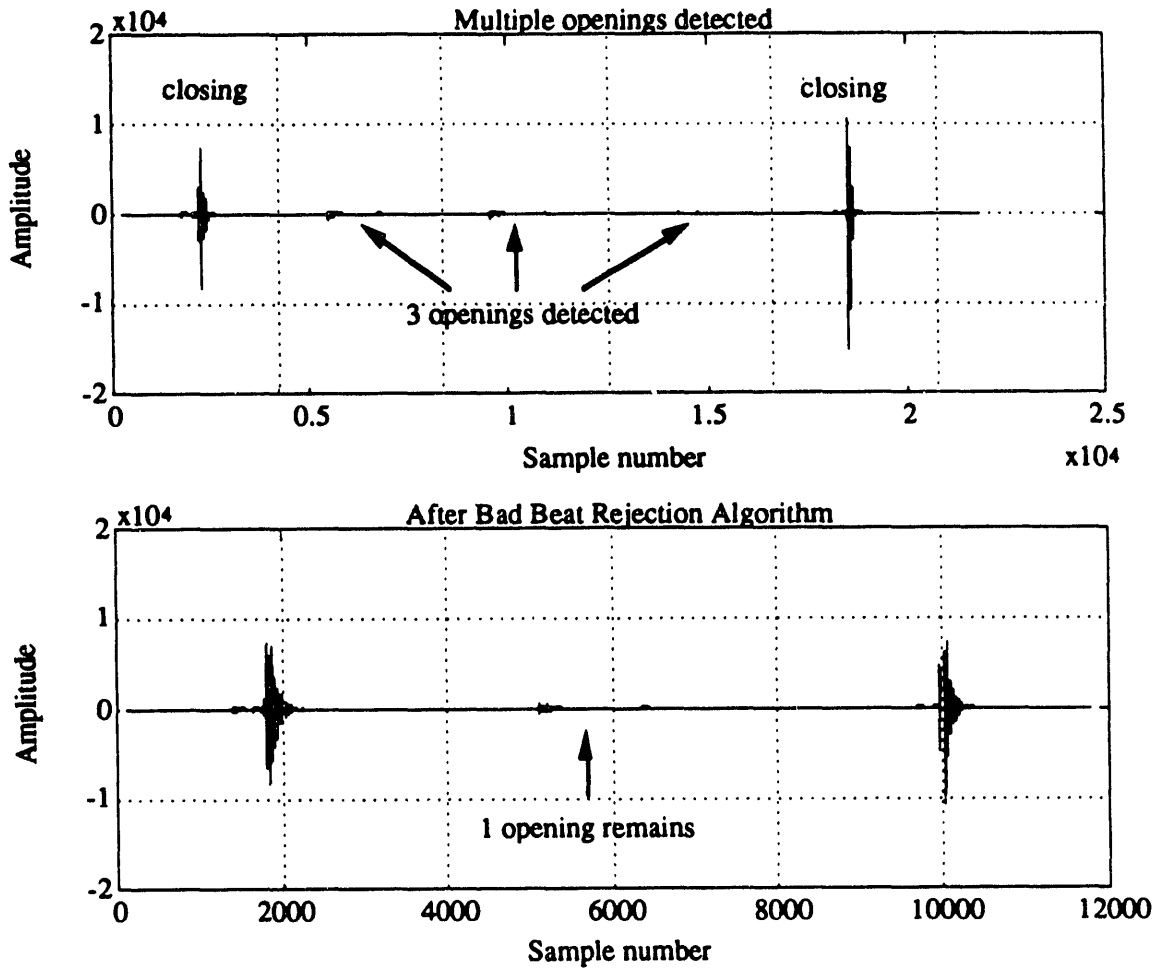


Figure 6: Bad Beat Rejection Example

The corresponding "BEATS" and "NEWBEATS" file sections are shown here:

BEATS file

```
6779049  6779575  8218.000000  1
6792507  6793842  474.000000   0
6811314  6812726  496.000000   0
6830384  6830441  97.000000    0
6834592  6835001  15183.000000 1
```

NEWBEATS file

```
6779049  6779575  8218.000000  1
6792507  6793842  474.000000   0
6834592  6835001  15183.000000 1
```

2.4 Beat Extraction

Now that we know where to find all the good beats in the raw data, we can then extract them. Based on the starting and ending locations in the "NEWBEATS" file, the openings and closings are cut from the raw data file and pasted together into separate opening and closing data files. Each opening and closing is centered in a window of user defined size. We use a windowsize of 4096 samples at a sampling rate of $2.0833 * 10^{-5}$ seconds to ensure that we capture the whole opening beat. The capability exists in the code to put the openings and closings into one file, but we keep them separate for processing purposes.

The noise just prior to each opening is extracted as well. Care is taken to be sure that the noise does not overlap the previous closing. If the code decides that it has run into the previous closing, it writes a noise errors file indicating this.

The result of the beat extraction is three main files, and "openings", "closings", and "noise" file. A block diagram of the process is shown below in Figure 7.

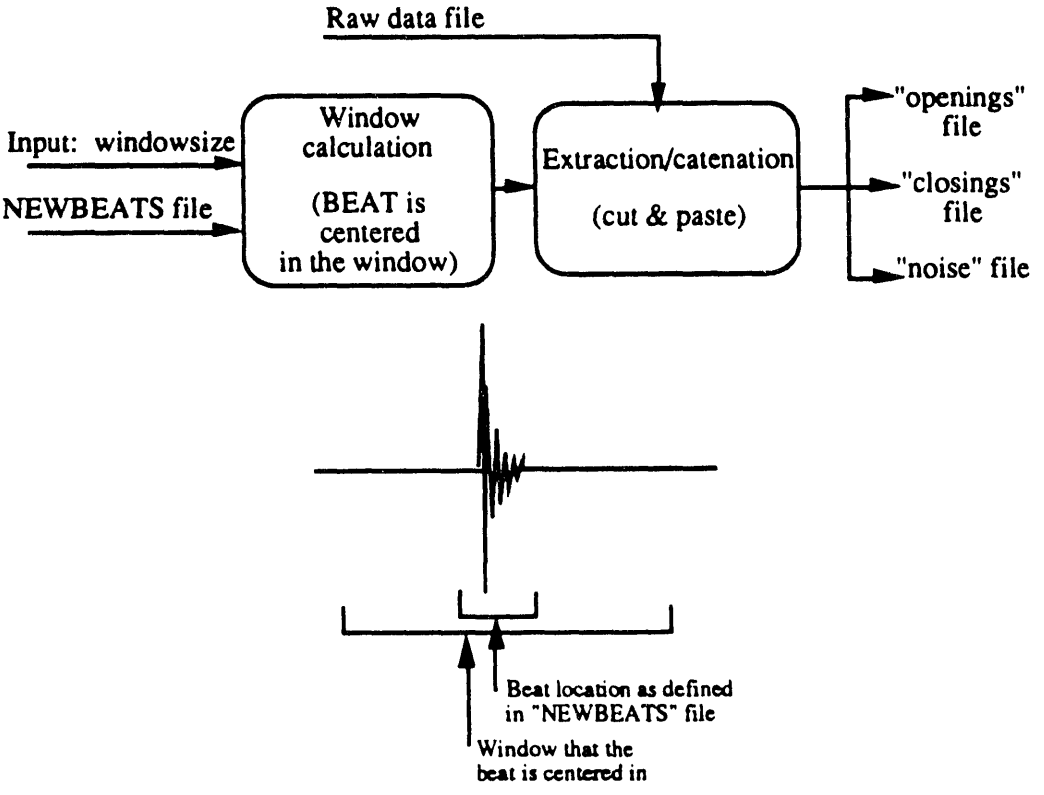


Figure 7: Beat Extraction Algorithm

In Figure 8 examples of the "openings", "closings", and "noise" files are presented.

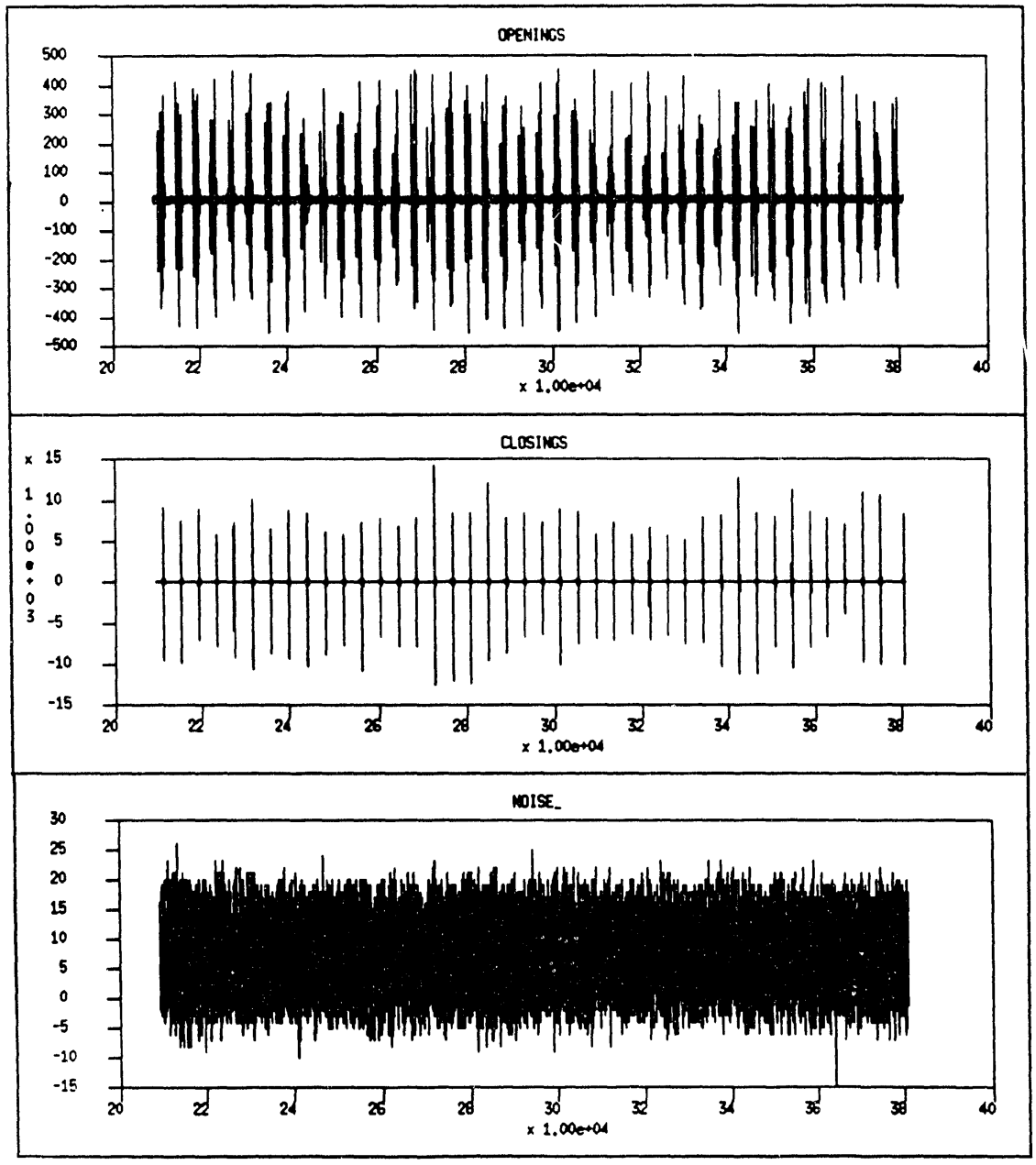


Figure 8: Examples of the "openings", "closings", and "noise" files

2.5 Filtering and Resampling

On a beat by beat basis, the data is filtered into three bands and resampled if necessary. We would not need to do this if the spectrum were flat, but because the spectrum has much more energy in the higher frequencies, we divide it up in such a way that we can look at important parts of the spectrum separately. The three bands we employ currently are a low band from 1-5 kHz, a medium band from 3-7 kHz, and a high band from 7-24 kHz. For the low band we apply a bandpass filter, and then are able to downsample by a factor of four. Likewise for the medium band we apply a bandpass filter and are able to downsample by a factor of two. The high band is merely highpass filtered with no down sampling. Depicted in Figure 9 is an example of the low, medium, and high bands for opening beats.

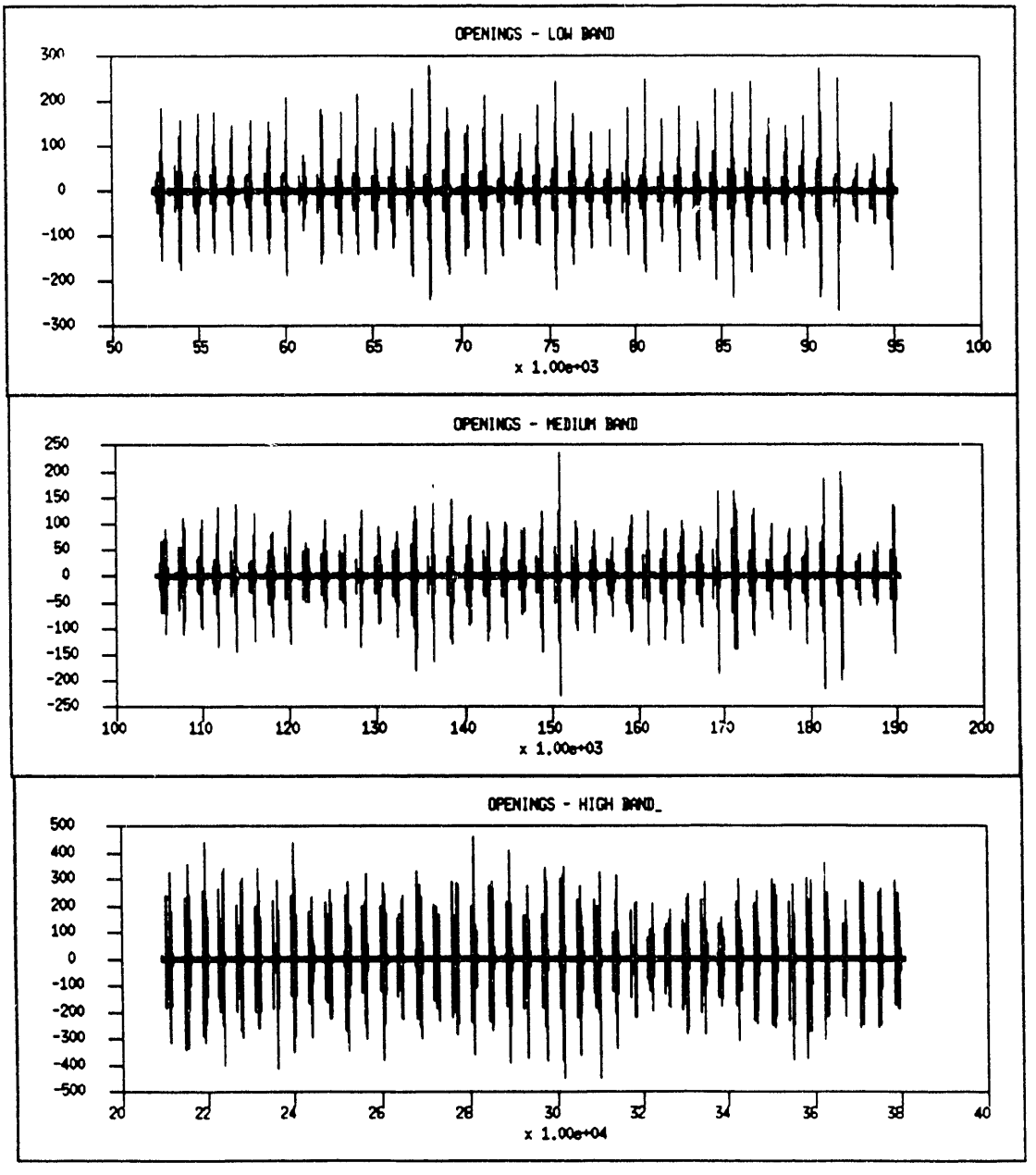


Figure 9: Examples of opening low, medium, and high band files

2.6 Spectral Estimation

Now from the three bands we create spectrograms. The spectrograms are created by taking the spectra of each opening a beat a time and then stacking the spectra together. From this, we should be able to observe spectral lines at various frequencies. Because the high band tends to have a lot of sharp peaks, we found that the AR (Auto-Regressive) model [3] worked well using 100th order. This is because the AR model is an all-pole model, which means that the transfer function has the form,

$$H_{AR}(z) = \frac{\sigma}{A(z)}, \text{ where } \sigma \text{ is the white noise standard deviation}$$

where

$$A(z) = 1 + a_1 z^{-1} + \ldots + a_N z^{-N}$$

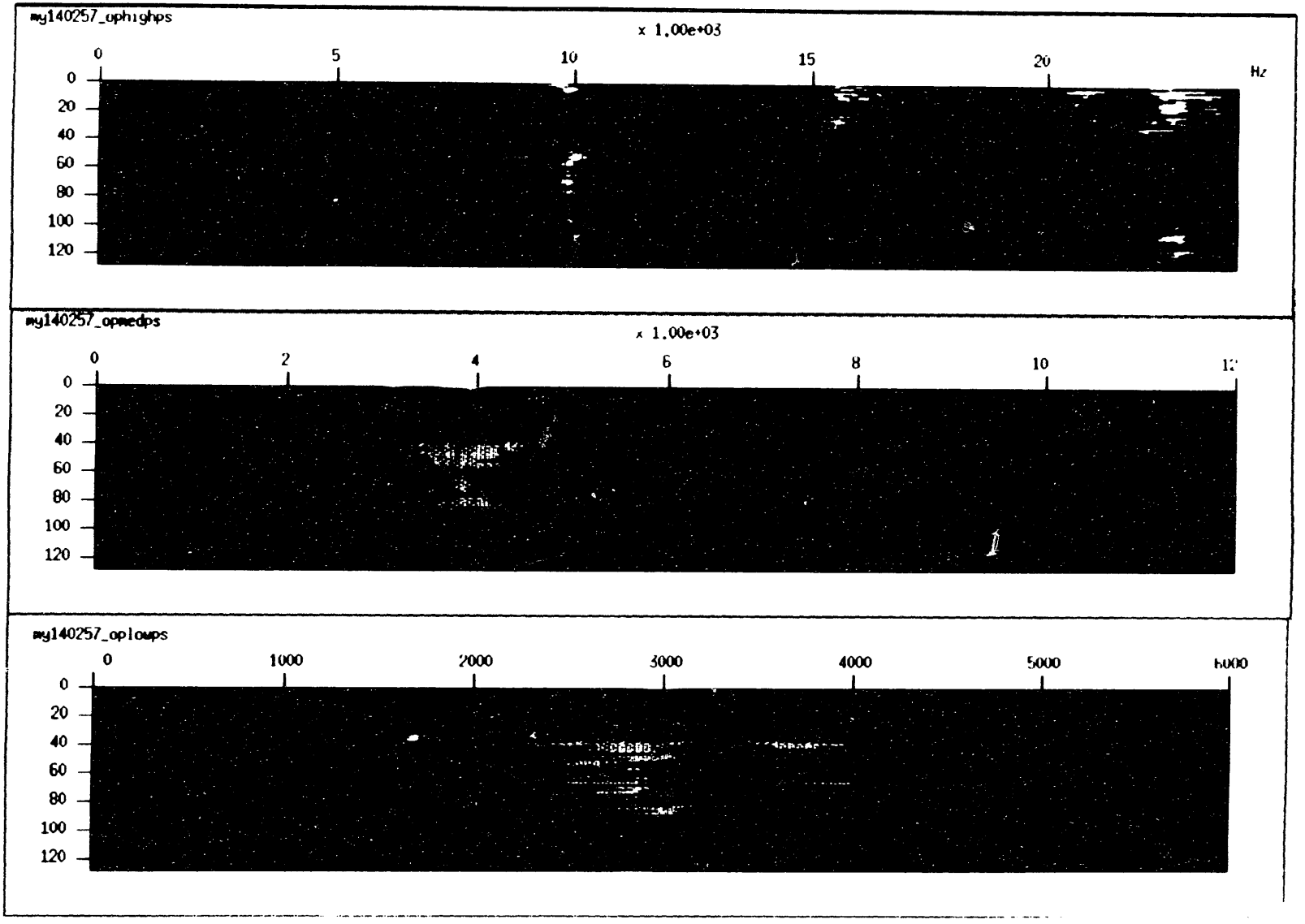
The spectral estimate is then formed as,

$$S_{AR}(\Omega) = \frac{\sigma^2 \Delta T}{\left|\hat{A}\left(e^{j\Omega}\right)\right|^2}$$

For the low and medium bands, the spectrum is rather broad band and does not appear to have any sharp peaks. Because of this, we found that the MVDR (Minimum Variance Distortionless Response) model[3] for spectral estimation worked satisfactorily using 25th order. The MVDR spectral estimate is formed by averaging all the lower order AR models in the following fashion,

$$\frac{1}{S_{MVDR}(\Omega)} = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{S_{AR}(\Omega, i)}$$

Along with the spectrum, when using the MVDR model, we create a file of the reflection coefficients, which can be used to reconstruct the entire spectrogram and can also be used to derive other features directly[4]. Or, when using the AR model, we create a file of the AR coefficients. In both cases we write the variance, σ^2, for each beat to a file. See Figure 10 for examples of the low, medium, and high spectrums.



Example of the Spectrograms for Low, Medium, and High Bands

FIGURE 10

2.7 Database

The result of these algorithms is a database of useful information about the heart valves. The files that are currently stored in the database have the Shiley filename followed by a suffix. The file suffixes we store are:

_beats- text file containing start sample, stop sample, max value, and event indicator
_newbeats - updated and screened "_beats" text file
_openings - binary floating point data file with the extracted and concatenated openings
_closings - binary floating point data file with the extracted and concatenated closings
_noise - binary floating point data file with the extracted and concatenated noise chunks
_noiserrs - text file with location of possible noise error locations
_oplow - low band openings file
_oplowps, _oplowvar, _oplowrefl - low band opening spectrum, variance, and reflection coefficients files
_opmed - medium band openings file
_opmedps, _opmedvar, _opmedrefl - medium band opening spectrum, variance, and reflection coefficients files
_ophigh - high band opening file
_ophighps, _ophighvar, _ophighar - high band opening spectrum, variance, and reflection coefficients files

3.0 SUMMARY

We have developed the signal processing capabilities to detect, identify, and extract opening and closing heartbeats from a data file containing them. Likewise, we can obtain information about the heartbeats, such as the power spectrum, which is useful for feature extraction. With this variety of information about the heartbeats from the two types of valves (SLS and intact), we are confident that with the proper features, we should be able to distinguish between them.

ACKNOWLEDGMENTS

We would like to thank Tony DeGroot and Bob Searfus who wrote the detection and extraction codes, and Ray Chia and Becky Interbitzen of Shiley who provided us with technical guidance as well as vast amounts heart valve data on Exabyte tapes.

REFERENCES

[1] Graham Thomas, "Signal Processing and Classification of Acoustic Signatures from Björk-Shiley Convexo-Concave Heart Valves"

[2] Hiratzka, 1988

[3] Candy, J. V. (1988) *Signal Processing: The Modern Approach* McGraw-Hill, New York.

[4] J. D. Markel, A. H. Gray, Jr. (1976) *Linear Prediction of Speech* Springer-Verlag, New York.

APPENDIX

Unless otherwise indicated, all programs take their input file from standard input and write their output files to standard output.

A.1 Tape File Conversion

tail +1025c filename - to get rid of the header

splitsheep.c - reads interleaved file from Shiley and creates a 16 bit integer file.
Inputs: starting index, optional finish index
Example: splitsheep 0 < input > output

stof.c - converts 16 bit integer file to binary floating point
Example: stof < input > output.float

A.2 Beat Detection

butter.c - filtering
Inputs: filter order, filter type, low frequency cutoff, high frequency cutoff, and delta t
where filter type = (0 Lowpass)(1 Highpass)(2 Bandpass)(3 Bandreject)
Example: butter 3 2 0.2 0.45 1.0 < input > output

detect.c - sta to lta ratio
Inputs: short term average window length(sec), long term average window length(sec), and delta t (sec)
Note: If delta t is entered as 1.0, then the short term and long term average window lengths can be entered as the number of samples.
Example: detect 50.0 500.0 1.0 < input > output

segment.c - thresholding and event location text file
Inputs: threshold between signal and noise, windowsize(sec), delta t
Note: If delta t is entered as 1.0, then the windowsize can be entered as the number of samples.
Example: segment 3.0 2800 1.0 < input > output.text

getstat.c - statistic appended to each line of text file
Inputs: binary floating point data file name, operation, windowsize
where operation = (0 get absolute maximum value of event)(1 get average value of event), and the windowsize should be the same as in segment.c
Example: getstat input.float 0 2800 < input.text > output.text

togglemax.c - maximize open and close toggling and append 0 or 1 to each line in the text file
Example: togglemax < input.text > output.text

togglefix.c - timing regularity check and marking of the text file if bad
Inputs: percentage deviation from average event spacing to allow
Example: togglefix 0.2 < input.text > output.text

togglefilter.c - outputs corrected text file and ensures at least a certain length sequence of openings and closings
Inputs: length of shortest acceptable open-close sequence
Example: togglefilter 5 < input.text > output_beats

A.3 Bad Beat Rejection

fixbeats.c - modifies text file to make ordering close-open etc., rejects out of bound openings along with their previous opening.
Inputs: beats file name, newbeats filename, percentage of the standard deviation to use for the boundaries
Example: fixbeats input_beats output_newbeats 50

A.4 Beat Extraction

extract.c - based on the text file of start and stop locations, this extracts the beats out of the raw data and centers them in a window.
Inputs: raw data filename, extraction flag(0 = openings, 1= closings, 2=both), windowsize in samples
Example: For this routine and the following routines, see A.8 for examples.

extractn.c - extracts noise just prior to each opening

Inputs: raw data file name, filename for noise errors, windowsize in samples

A.5 Filtering and Resampling

resamp.c - On a window by window basis, filters and resamples the data
Inputs: Downsample factor, sampling rate(sec), windowsize in samples

A.6 Spectral Estimation

powerspec_mvdr.c - On a window by window basis, it finds the power spectrum using the Minimum Variance Distortionless Response method.
Inputs: model order, sample rate in seconds, windowsize in samples, reflection coefficient filename, variance filename

powerspec_ar.c - On a window by window basis, it finds the power spectrum using the Auto Regressive (Levinson Durbins) method.
Inputs: model order, sample rate (sec), windowsize in samples, ar coefficient filename, variance filename

A.7 Printing

printsheep.c - This routine converts a power spectrum output file into a spectrogram View file and prints it out on our Tektronics printer .
Inputs: Filename, windowsize

A.8 Shellscript "go1"

This UNIX shellscript automates the signal processing of the heart valve data for a particular file.

```
#!/usr/local/bin/bash
export DISPLAY=${HOST}:0

tail +1025c $1 | splitsheep 0 | stof > $2
butter 3 2 0.2 0.45 1.0 < $2 | detect 50.0 500.0 1.0 | segment 3.0 $4 1.0 | getstat $2 0 $4 |
togglemax | togglefix 0.2 | togglefilter 5 > $3_beats
```

```
fixbeats $3_beats $3_newbeats 50
extract $2 0 4096 < $3_newbeats > $3_openings
extract $2 1 4096 < $3_newbeats > $3_closings
extractn $2 $3_noiserrs 4096 < $3_newbeats > $3_noise

resamp 4 1000 5000 .0000208333 4096 < $3_openings > $3_oplow
resamp 2 3000 7000 .0000208333 4096 < $3_openings > $3_opmed
resamp 1 7000 24000 .0000208333 4096 < $3_openings > $3_ophigh

powspec_mvdr 25 .000083333 1024 $3_oplowrefl $3_oplowvar < $3_oplow >
$3_oplowps
printsheep $3_oplowps 1024

powspec_mvdr 25 .0000416667 2048 $3_opmedrefl $3_opmedvar < $3_opmed >
$3_opmedps
printsheep $3_opmedps 2048

powspec_ar 100 .0000208333 4096 $3_ophighar $3_ophighvar < $3_ophigh >
$3_ophighps
printsheep $3_ophighps 4096
```

DATE
FILMED
10 / 14 / 93
END