Abstract

Arterial stiffness is one of the earliest symptoms of various cardiovascular diseases like the coronary heart disease. Over the course of this project, we have been able to implement an alternate algorithm for detection of the arterial walls of the common carotid artery from the raw ultrasound data. This technique could be further extended to determine the arterial stiffness of common carotid artery. Our method involves noise reduction using discrete wavelet transforms, peak estimation using a novel algorithm and then determination of wall location.

Introduction

One of the best techniques for denoising of a real time signal is by using wavelet transforms. The advantage of wavelet transforms over Fourier transforms is the possibility to capture both temporal and frequency information. Continuous wavelet transforms (CWT) is usually redundant as far as the reconstruction of the signal is concerned. Discrete wavelet transforms (DWT) are sufficient for most applications. DWT consists of discretely sampled wavelets. In biomedical electronic systems, it has found applications in signal processing in gait analysis, design of pacemakers and ECG.[2]

Materials and Methods

MATLAB simulation was used for the entire implementation. The Wavelet Toolbox, in particular the 1-D DWT was used for analysis and denoising.
We have written a MATLAB function to which the raw ultrasound data is passed as an input argument. A single frame is extracted and analysed using Daubechies-4 wavelet at level 5. This wavelet was chosen because of the similarity between the mother wavelet and our signal. This was further confirmed by its application in ECG [1]. After analysing the detail coefficients, D4 was found to have the relevant peaks. Only these coefficients are used to reconstruct the denoised signal. The mesh plot of the denoised signal shows the clear contrast between the pulsating walls and the static echoes.

The denoised signal is smoothed using a suitable weighted average function. This was done to aggregate nearby peaks to maximise efficiency of the peak detection algorithm.

The peak detection algorithm considers a point to be a maxima if it is greater than the preceding value by a threshold set by the user. The peak detection is implemented in a separate function "peakdet.m". This algorithm was used to detect the peaks. The samples corresponding to the peaks were set to a value one and the rest zeros in a separate matrix.

A sliding window of width three was used to sum the values across all frames. The summed values of the static echoes are much larger than the summed values of the walls. This is due to the fact that the static echoes occur in the same window across all frames whereas the walls pulsate in
and out of the window. A threshold of 80 percent of the average of the first 120 frames is used as the upper limit to cancel static echoes. All peaks above this threshold are static echoes. The static echoes in the beginning were identified based on the property that they are very closely spaced. This in conjunction with the threshold was used to eliminate static echoes in the initial samples.

The remaining echoes are sharp and sparsely spaced while the wall peaks are spread over at least 150 samples. This property was exploited to enhance the wall peaks alone by smoothing using a moving window average of width 100 samples. This gives clear peaks at the wall locations and some terminal static echoes.

Using the property that the proximal wall echo is stronger than the distal wall echo, the proximal wall was immediately detected. The proximal and distal walls across all the data files given to us were about 800 samples apart. Using a constraint of minimum separation between the walls to be 500 samples, the closest peak beyond the proximal wall was marked as the distal wall. The sample locations of the proximal and distal wall were returned.

Figure 3: Summed peaks before and after thresholding

Figure 4: Final peak detection
Conclusion

The algorithm was tested for the four data files given to us. It was found to detect the peaks corresponding to the wall locations. This implementation uses all 100 frames in the data. However, with the use of stricter thresholds, it should be possible to optimise the number of frames required for wall detection.

References


